Indice

1 Prerequisiti

1.1 Blockchain

Una blockchain è un registro aperto e distribuito di dati, strutturato come una catena di blocchi contenenti le transazioni. I dati risiedono su unità computazionali chiamati *nodi*, questi, come mostrato nell'immagine 1, sono interconnessi e comunicano tra loro per mantenere i dati di tutti i nodi aggiornati.[9]

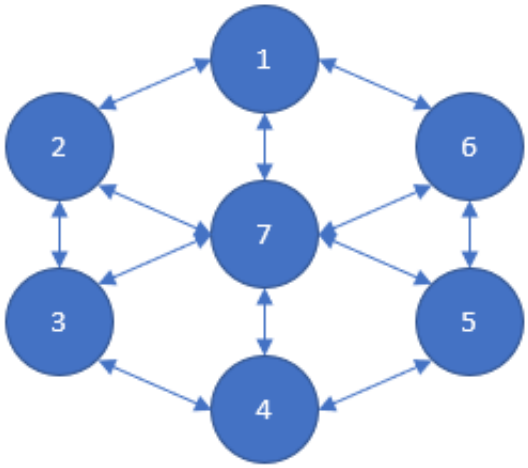

Figura 1: Rete di nodi

1.1.1 I nodi

Le responsabilità di un nodo sono principalmente:

- Controllo della validità di un nuovo record di dati, chiamato anche transazione, e acctettarlo o rifiutarlo;
- Nel caso di un record valido, salvataggio della transazione nel registro locale del nodo;

- Comunicazione e distribuzione della transazione agli altri nodi. In questo modo tutti i nodi hanno la stessa versione del registro.

1.1.2 Le caratteristiche di una blockchain

Le caratteristiche principali della tecnologia blockchain sono:

- Decentralizzazione: le informazioni contenute nel registro digitale vengono distribuite tra più nodi, così da garantire sicurezza e resilienza dei sistemi anche in caso di attacco ad uno dei nodi o in caso di perdita di un nodo.

- Tracciabilità: ogni elemento salvato nel registro è tracciabile in ogni sua parte e se ne può risalire all'esatta provenienza e alle eventuali modifiche apportate nel corso del tempo, con una precisione assoluta.

- Disintermediazione: i singoli nodi della blockchain certificano le informazioni distribuite, rendendo quindi del tutto inutile la presenza di enti centrali o di aziende per la certificazione dei dati.

- Trasparenza: il contenuto del registro è visibile a tutti ed è facilmente consultabile e verificabile da ogni nodo della rete ma anche tramite servizi che interrogano la blockchain senza apportare modifiche. Nessuno può nascondere o modificare dati senza che l'intera rete venga a saperlo.

- Solidità del registro: dopo aver aggiunto un'informazione al registro, essa non può essere modificata senza il consenso di tutta la rete.

- Programmabilità: le operazioni di transazione possono anche essere programmate nel tempo, così da poter attendere il verificarsi di determinate condizioni prima di procedere con l'inserimento o la modifica.[10]

1.1.3 Smart contracts

Le blockchain permettono di implementare codici e funzioni all'interno di esse, questi sono chiamati smart contracts e permettono l'esecuzione di operazioni quando predeterminate condizioni si avverano. Sono tipicamente usate per automatizzare l'esecuzione di un accordo, in questo modo tutti i partecipanti possono verificarne immediatamente i risultati, senza aver bisogno di un itermediario. Possono, inoltre, essere utilizzate per automatizzare workflow, innescando azioni successive al raggiungimento di certe condizioni.[8]

1.2 Ethereum

Ethereum è una piattaforma blockchain decentralizzata che stabilisce una rete peer-to-peer che esegue e verifica smart contracts in modo sicuro. I smart contracts permettono transazioni tra gli utenti senza la necessità di un autorità centrale. Le transazioni sono immutabili, verificabili, e distribuiti in modo sicuro sulla rete. Le transazioni sono inviate e ricevute da account Ethereum creati dagli utenti. Come costo per il processamento di una transazione sulla rete, l'utente deve spendere Ether (ETH), la criptovaluta nativa di Ethereum.[1]

1.2.1 Token Ethereum

Ethereum permette la creazione di token crittografici all'interno della sua rete. Questi token non sono altro che smart contracts scritti seguendo specificazioni stabilite dagli sviluppatori della rete. I token possono essere di tipi di differenti, a seconda delle specificazioni seguite. In Ethereum ci sono pricipalmente due tipi di token: token ERC-20 e NFT.

1.2.1.1 Token ERC-20

I token ERC-20 sono il tipo di token più diffuso sulla rete Ethereum, sono lo standard per la definizione di token fungibili[3], ossia i singoli token sono indistinguibili ed itercambiabili tra loro. Un token ERC-20 implementa le specificazioni indicate nell'EIP-20 che richiede nello smart contract del token la presenza di diversi metodi. I più importanti metodi richiesti sono:

- `balanceOf(address _owner)`: restituisce la quantità di token posseduto da `_owner`;
- `transfer(address _to, uint256 _value)`: trasferisce una quantità di token indicata da `_value` all'indirizzo `_to`;
- `transferFrom(address _from, address _to, uint256 _value)`: trasferisce una quantità `_value` di token dall'indirizzo `_from` all'indirizzo `_to`;
- `approve(address _spender, uint256 _value)`: permette all'indirizzo `_spender` di ritirare fino a `_value` token dall'account.[7]

1.2.1.2 NFT

I Non Fungible Tokens sono, appunto, token non fungibili, ossia ogni token è univoco e non intercambiabile con un altro. Sono utilizzati per replicare le proprietà tipiche di un oggetto fisico come la scarsità, l'unicità e la possibilità di dimostrare la proprietà del token.[6] Data la natura del token, l'uso più comune di questi token è la creazione di arte digitale.[2]
Gli NFT seguono lo standard dettato dall'EIP-721.

1.2.1.3 Smart contracts in Ethereum

In Ethereum gli smart contracts sono considerati come account, questo significa che hanno un saldo e possono inviare transazioni sulla rete. A differenza di un normale account, però, gli smart contracts non sono controllati dagli utenti ma eseguono il codice con cui sono stati programmati. Gli account user possono interagire con uno smart contract inviando una transazione che effettua una chiamata ad una funzione definita nello smart contract. Di default questi contratti non possono essere eliminati e le interazioni con essi sono irreversibili.

Il codice di uno smart contract si scrive utilizzando *Solidity*, un linguaggio object-oriented usato per implementare smart contracts su diverse piattaforme blockchain. Di seguito un esempio di uno smart contract semplice:

```
1   pragma solidity ^0.5.2;
2
3   contract EsempioContratto() {
4     address public owner;
5
6     event EsempioEvento(
7       uint256 parametroEvento
8     )
9
10    function esempioFunzione(
11      uint256 _parametro
12    ) public returns (uint256) {
13      emit EsempioEvento(_parametro)
```

```
14            return _parametro
15          }
16        }
```

Esempio codice di uno smart contract

Solitamente uno smart contract inizia con la dichiarazione della versione di solidity usata per la scrittura del codice, in questo modo il compilatore, se di versione superiore, rifiuta la compilazione del codice. Per far questo si scrive: `pragma solidity [versione]`. Nell'esempio precedente la versione è dichiarata nella prima riga, in questo caso la versione indicata è superiore a 0.5.2.
Una volta dichiarata la versione, inzia il codice dello smart contract, questo viene indicato con `contract [nome contratto]()`. Prendendo sempre come riferimento l'esempio precedente, alla riga 4 viene dichiarata una variabile chiamata owner con visibilità `public` e tipo `address`, ossia un indirizzo Ethereum. Ci sono 4 livelli di visibilità per le variabili e le funzioni:

- `public`: le funzioni possono essere chiamate anche da contratti esterni, per le variabili vengono generate in automatico delle funzioni getter implicite;
- `external`: le funzioni e le variabili possono essere accedute solo dall'esterno e non internamente nel contratto;
- `internal`: le funzioni e le variabili possono essere accedute dentro il contratto stesso e i contratti derivati;
- `private`: visibili solo nel contratto in cui le variabli e le funzioni sono definite.[5]

Solidity fornisce numerosi tipi per le variabili, di cui quelle usate in questa tesi sono:

- `bool`: per indicare una variabile booleana;
- `uint`: per indicare un intero senza segno, può avere diverse dimensioni aggiungendo il numero di bits, ad esempio `uint256`;
- `address`: per indicare un indirizzo Ethereum;
- `mapping`: per indicare un dizionario con chiavi di ricerca e valori associati.[4]

Alla riga 6 è stato dichiarato un evento, ossia uno strumento utile per fare logging delle transazioni e per permettere agli utenti di mettersi in ascolto di questi eventi. Un evento può contenere dati aggiuntivi, in questo caso l'evento `EsempioEvento` contiene il dato `parametroEvento` di tipo uin256. Per emettere un evento si utilizza `emit [nome evento](dati evento)`, come mostrato alla riga 13.
Alla riga 10 è stata dichiarata una funzione di nome `esempioFunzione`, con visibilità `public`, con parametro `_parametro` e che restituisce un valore di tipo `uin256`. Questa funzione esegue solo due operazioni: emette l'evento `EsempioEvento` e restituisce un valore.

1.3 OpenZeppelin

OpenZeppelin è una libreria per lo sviluppo di smart contracts sicuri. Le principali funzionalità fornite da OpenZeppelin sono:

- Implementazione dei diversi standard dei token Ethereum;
- Gestione del controllo degli accessi agli smart contracts;

- Componenti Solidity riutilizzabili per creare smart contracts.

1.4 Metamask

Metamask è un'estensione del web browser. Questo software permette di connettere il browser con applicazioni decentralizzate basate sulla piattaforma Ethereum. Metamask permette la gestione di wallet Ethereum, la ricezione e l'invio di criptomonete basate su Ethereum, e l'interazione con applicazioni decentralizzate. L'estensione, inoltre, fornisce le API Ethereum web3, in questo modo le applicazioni sono in grado di leggere dati sulla blockchain.

1.5 ReactJS

React è una libreria JavaScript open-source per lo sviluppo di interfacce utente.

1.5.1 Caratteristiche di ReactJS

1.5.1.1 Componenti

I Componenti permettono di suddividere la UI in parti indipendenti, riutilizzabili e di pensare ad ognuna di esse in modo isolato. Per definire un componente è necessario implementare una funzione JavaScript, ad esempio:

```
1 function Saluto(props) {
2     return <h1>Ciao, {props.nome}</h1>;
3 }
```

Esempio componente

Questo componente accetta un oggetto parametro contenente dati sotto forma di una singola "props", il quale è un oggetto parametro avente dati al suo interno. Per renderizzare un componente bisogna utilizzare la funzione `ReactDOM.render()`, passandole come parametri il componente da visualizzare e il riferimento al componente padre. Se si volesse, quindi, renderizzare il componente `Saluto`, passando "*Martina*" come parametro `nome`, il codice potrebbe essere:

```
1       ReactDOM.render(
2         <Saluto nome="Martina"/>,
3         document.getElementById('root')
4       );
```

Esempio composizione di componenti

I componenti, inoltre, possono essere composte da altri componenti. In questo caso, renderizzando il componente padre, verranno visulizzati anche i componenti figli. Ad esempio, si potrebbe avere un componente `Convenevoli` che contiene multipli componenti `Saluto`:

```
1       function Convenevoli() {
2         return (
3           <div>
4             <Saluto nome="Sara" />
5             <Saluto nome="Cahal" />
6             <Saluto nome="Edite" />
7           </div>
```

```
8         );
9       }
```

Esempio renderizzazione componente

1.5.1.2 Hook state

Un componente React di default è stateless. Usando la funzione `useState()` si può aggiungere uno stato interno ad un componente, React preserverà questo stato tra le ri-renderizzazioni. `useState` ritorna una coppia: il valore dello stato corrente ed una funzione che ci permette di aggiornarlo. La funzione ha un unico parametro ed è il suo stato iniziale. Ad esempio, se si volesse realizzare un contatore con un bottone che, alla sua pressione, aumenti il valore del contatore, si potrebbe scrivere il seguente codice:

```
1       function Contatore() {
2         const [contatore, setContatore] = useState(0);
3         return (
4           <div>
5             <p>Hai cliccato {contatore} volte</p>
6             <button
7               onClick={() => setContatore(contatore + 1)}>
8               Cliccami
9             </button>
10          </div>
11        );
12      }
```

```
13
```

Esempio contatore con stato interno

1.5.1.3 Hook effect

Il costrutto `useEffect()` permette l'esecuzione di funzioni ad ogni renderizzazione da parte di React. Questa funzione viene utilizzata per effettuare operazioni nei vari stati del ciclo di vita di un componente.
Nel seguente esempio il titolo del documento viene aggiornato all'aumentare del valore del contatore, infatti, ad ogni aggiornamento del DOM da parte di React, viene chiamata la funzione passata a `useEffect()`.

```
1         function ContatoreConTitolo() {
2           const [contatore, setContatore] = useState(0);
3
4           useEffect(() => {
5             document.title = `Hai cliccato ${contatore} volte`;
6           });
7
8           return (
9             <div>
10              <p>Hai cliccato {contatore} volte</p>
11              <button
12                onClick={() => setContatore(contatore + 1)}>
13                Cliccami
14              </button>
```

```
15              </div>
16            );
17          }
```

Esempio uso di useEffect()

1.6 Material-UI

Material-UI è una libreria per ReactJS per creare interfacce utente. La libreria contiene al suo interno numerosi componenti grafici, questi sono forniti di un tema di default, per modificare l'aspetto di un componente si può utilizzare la sua proprietà `className`.

Nell'esempio seguente vengono modificati le dimensione di un componente `<Button>`:

```
1           .Button {
2             width: "100px",
3             height: "100px"
4           }
5
6           <Button className="Button">
7
```

Esempio modifica aspetto di un componente

1.7 Truffle e Ganache

Truffle e Ganache sono entrambi strumenti contenuti all'interno della suite software Truffle. Ganache permette la creazione di una blockchain Ethereum che viene eseguita in locale, semplificando, così, il deploy ed il test degli smart contracts. Truffle è un software che facilita lo sviluppo di smart contracts, i principali comandi di truffle utilizzati sono stati:

- `truffle compile`, per compilare gli smart contracts;
- `truffle test`, per eseguire i file di test;
- `truffle deploy`, per eseguire il deploy degli smart contracts.

2 Scrittura e test dei smart contracts

3 Implementazione dell'interfaccia grafica

L'interfaccia utente che implementa le funzionalità di scambio di token è stata divisa in due pagine. La prima è chiamata *Create Sale* e, scegliendo i token da vendere e quelli da accettare, permette la creazione di una vendita. La seconda è chiamata *Sales List* e permette di visualizzare una lista di vendite, sia quelle in corso che quelle terminate.

3.1 Create Sale

Il processo di creazione di una vendita è diviso in tre step. Nel primo step viene permessso all'utente di scegliere i token in suo possesso da mettere in vendita. Nel secondo step l'utente sceglie i token che accetta come pagamento per la vendita dei token scelti nel primo step. Nel terzo step viene mostrato all'utente un riassunto delle scelte fatte nei due step precedenti, in questo passo, inoltre, l'utente può impostare una data di scadenza della vendita. Per mostrare graficamente il progresso nei diversi step è presente, in cima alla pagina, un componente `<stepper>`, questo indica all'utente gli step terminati e quelli ancora da completare.

3.1.1 Step 1: scelta dei token da vendere

La pagina contiene una lista, creata usando il componente `<List>`, ed una barra di ricerca, creata usando il componente `<TextField>`.
Ogni elemento della lista contiene le informazioni di un token posseduto dall'utente e un componente `<TextField>`, quest'ultimo permette l'inserimento della quantità desiderata del token da mettere in vendita. Gli eventi a cui la pagina reagisce possono essere divisi in tre categorie: caricamento della

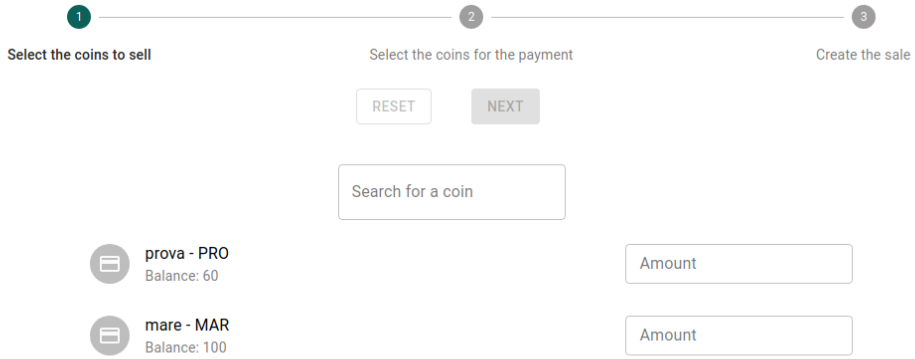

Figura 2: Pagina per la scelta dei token in vendita

pagina, inserimento di quantità dei token e inserimento di testo nella barra di ricerca.

3.1.1.1 Caricamento della pagina

Al caricamento della pagina vengono istanziate diverse variabili che gestiscono lo stato della stessa.

Riferimenti bibliografici

[1] Amazon. *What is Ethereum?* URL: https://aws.amazon.com/it/blockchain/what-is-ethereum/.

[2] Andrew Carroll. *The art world needs blockchain.* URL: https://irishtechnews.ie/the-art-world-needs-blockchain/.

[3] Ethereum.org. *Ethereum Docs.* URL: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/.

[4] Ethereum.org. *Ethereum Docs Types.* URL: https://docs.soliditylang.org/en/v0.4.24/types.html.

[5] Ethereum.org. *Ethereum Docs Visibility and getters.* URL: https://docs.soliditylang.org/en/v0.4.24/contracts.html#visibility-and-getters.

[6] Ethereum.org. *Non-fungible tokens (NFT).* URL: https://ethereum.org/en/nft/.

[7] Vitalik Buterin Fabian Vogelsteller. *EIP-20: Token Standard.* URL: https://eips.ethereum.org/EIPS/eip-20.

[8] IBM. *What are smart contracts on blockchain?* URL: https://www.ibm.com/topics/smart-contracts.

[9] Jimi S. *Blockchain: What are nodes and masternodes?* URL: https://medium.com/coinmonks/blockchain-what-is-a-node-or-masternode-and-what-does-it-do-4d9a4200938f.

[10] Giuseppe Vanni. *Blockchain: cos'è, come funziona, tecnologia e applicazioni.* URL: https://www.punto-informatico.it/blockchain-spiegazione/#h222751-0.