

# Indice

<b>1</b>	<b>Background</b>	<b>3</b>
1.1	CommonsHood . . . . .	3
1.2	Blockchain . . . . .	4
1.2.1	I nodi . . . . .	4
1.2.2	Le caratteristiche di una blockchain . . . . .	5
1.2.3	Smart contracts . . . . .	6
1.3	Ethereum . . . . .	6
1.3.1	Token Ethereum . . . . .	6
1.3.1.1	Token ERC-20 . . . . .	7
1.3.1.2	NFT . . . . .	7
1.3.2	Smart contracts in Ethereum . . . . .	8
1.4	OpenZeppelin . . . . .	10
1.5	Metamask . . . . .	11
1.6	ReactJS . . . . .	11
1.6.1	Caratteristiche di ReactJS . . . . .	11
1.6.1.1	Componenti . . . . .	11
1.6.1.2	Hook state . . . . .	13
1.6.1.3	Hook effect . . . . .	14
1.7	Material-UI . . . . .	15
1.8	Truffle e Ganache . . . . .	16
<b>2</b>	<b>Scrittura e test dei smart contracts</b>	<b>17</b>
<b>3</b>	<b>Implementazione dell'interfaccia grafica</b>	<b>18</b>
3.1	Create Sale . . . . .	18

3.1.1	Caricamento della pagina . . . . .	19
3.1.2	Box di ricerca . . . . .	21
3.1.3	Step 1: scelta dei token da vendere . . . . .	22
3.1.3.1	Caricamento della pagina . . . . .	23

# 1 Background

## 1.1 CommonsHood

CommonsHood è un'applicazione web basata su smart contract per blockchain Ethereum che ha lo scopo di fornire alla comunità strumenti per l'inclusione finanziaria e per supportare l'economia locale delle comunità di cittadini. Gli utenti, una volta registrati sulla piattaforma, possono creare monete, ossia token crittografici Ethereum basati sullo standard ERC-20, questi possono rappresentare beni di valore o servizi. Un altro asset che gli utenti possono creare e possedere sono i coupon, questi sono rappresentati da token non fungibili basati sullo standard ERC-721, in questo modo ogni singolo coupon è univoco e diverso dagli altri. Gli utenti possono utilizzare i coupon per ottenere prestazioni e servizi.

Un'altra funzionalità offerta dall'applicazione è la possibilità di creare crowdsales, questi sono usati per ottenere fondi per finanziare progetti o eventi, dando, in cambio, ai finanziatori monete oppure coupons. Un esempio di un'interazione con la piattaforma potrebbe essere: il negozio di attrezzature da sub *"Sotto il Mar"* crea un account su CommonsHood. Crea, inoltre, una moneta chiamata *"Doblone"*, questa può essere spesa in negozio per acquistare le attrezzature. Oltre alla vendita, il negozio noleggia anche le attrezzature, perciò crea dei coupon che possono essere utilizzati per ottenere i prodotti a noleggio. Inoltre, una stanza ha bisogno di ristrutturazioni perciò viene creata una crowdsale per ottenere il finanziamento necessario, in cambio vengono offerti dei coupon del negozio.

## 1.2 Blockchain

Una blockchain è un registro aperto e distribuito di dati, strutturato come una catena di blocchi contenenti le transazioni. Le transazioni solitamente rappresentano uno scambio di monete, chiamate token. Ogni blockchain ha un token proprio. 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[11]. Un account su una blockchain è costituito da una coppia di chiavi:

- pubblico: è un indirizzo sulla blockchain, i token nella rete sono registrati come appartenenti ad un indirizzo;
- privato: è come una password che l'utente utilizza per accedere ai propri fondi.

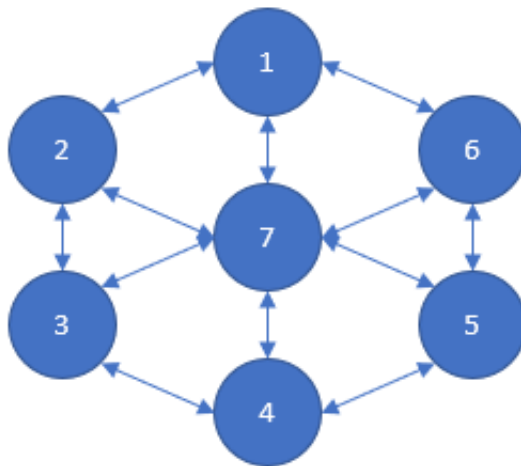


Figura 1: Rete di nodi

### 1.2.1 I nodi

Le responsabilità di un nodo sono principalmente:

- Controllo della validità di un nuovo record di dati, chiamato anche transazione, e accettarlo 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.

### **1.2.2 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 a 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[13].

### **1.2.3 Smart contracts**

Le blockchain permettono d’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 intermediario. Possono, inoltre, essere utilizzate per automatizzare workflow, innescando azioni successive al raggiungimento di certe condizioni[7].

## **1.3 Ethereum**

Ethereum è una piattaforma blockchain decentralizzata che stabilisce una rete peer-to-peer che esegue e verifica smart contracts in modo sicuro. Gli 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].

### **1.3.1 Token Ethereum**

Ethereum permette la creazione di token crittografici all’interno della sua rete. Questi token non sono altro che smart contracts scritti seguendo specifiche stabilite dagli sviluppatori della rete. I token possono essere di

tipi di differenti, a seconda delle specifiche seguite. In Ethereum ci sono principalmente due tipi di token: token ERC-20 e NFT.

#### 1.3.1.1 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 e intercambiabili tra loro. Un token ERC-20 implementa le specifiche 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[14].

#### 1.3.1.2 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.

### 1.3.2 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 a 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, inizia 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 variabili 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 **uint256**. 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 **uint256**. Questa funzione esegue solo due operazioni: emette l'evento **EsempioEvento** e restituisce un valore.

## 1.4 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[10].

## 1.5 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[8].

## 1.6 ReactJS

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

### 1.6.1 Caratteristiche di ReactJS

React fornisce numerosi strumenti che facilitano lo sviluppo di un'interfaccia grafica. Di seguito sono descritti quelli utilizzati in questa tesi.

#### 1.6.1.1 Componenti

I Componenti permettono di suddividere la UI in parti indipendenti, riutilizzabili e di pensare a 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 visualizzati 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

### 1.6.1.2 Hook state

Un componente React di default è stateless. Usando la funzione `useState()` si può aggiungere uno stato interno a un componente, React preserverà questo stato tra le ri-renderizzazioni. `useState` ritorna una coppia: il valore dello stato corrente e 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 }

```

Esempio contatore con stato interno

### 1.6.1.3 Hook effect

Il costrutto `useEffect()` permette l'esecuzione di funzioni a 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, a 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()

## 1.7 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, preso da [9], vengono modificati le dimensione di un componente `<Button>`:

```

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

```

Esempio modifica aspetto di un componente

## 1.8 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 e 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[12].



## **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 la lista di tutte le 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 l'utente sceglie 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.

In tutti e tre gli step sono presenti alcuni componenti grafici comuni, questi sono visibili nell'immagine 2.

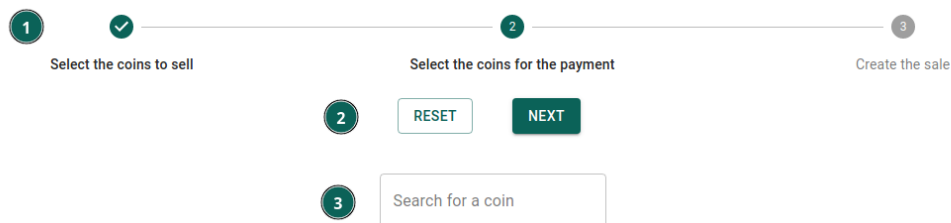


Figura 2: Componenti comuni

Per mostrare graficamente il progresso nei diversi step è presente, in cima alla pagina, un componente `<stepper>`, questo, come mostrato nell'immagine 2 al punto 1, indica all'utente gli step terminati e quelli ancora da completare. Al di sotto, al punto 2, sono presenti due pulsanti per la navigazione nei vari step. Il primo permette di tornare allo step 1 azzerando tutti i parametri inseriti. Il secondo serve per avanzare allo step successivo. Al punto 3 è presente un campo di inserimento per cercare una moneta in particolare.

### 3.1.1 Caricamento della pagina

Al caricamento della pagina vengono inizializzate alcune variabili, tra cui le più significative sono le seguenti:

```
1      const {web3Instance, userAccount} = props;
2      const [coinsToSellAmounts,
3            setCoinsToSellAmounts] = useState(new Map());
4      const [coinsToAcceptAmounts,
5            setCoinsToAcceptAmounts] = useState(new Map());
6      const [coinsList, setCoinsList] = useState([]);
7      const [activeCoinsList,
8            setActiveCoinsList] = useState([]);
9      const [activeStep, setActiveStep] = useState(0);
```

Inizializzazione delle variabili al caricamento della pagina Create Sale

`web3Instance` contiene l'istanza di web3 mentre `userAccount` contiene l'indirizzo Ethereum dell'utente corrente, questi vengono passati alla pagina

come props.

`coinsToSellAmounts` e `coinsToAcceptAmounts` sono variabili dotate di stato ed entrambi sono di tipo `Map`, ossia un dizionario con chiavi di ricerca e valori associati. Vengono usate per mantenere le quantità delle monete scelte da mettere in vendita, per la prima variabile, oppure quelle da accettare come pagamento, nel caso della seconda variabile. Hanno come chiavi di ricerca le monete mentre i valori sono le quantità di queste.

`coinList` è un array usato per contenere la lista di tutte le monete possedute dall'account utente.

`activeCoinsList` è l'array preso come riferimento per mostrare graficamente la lista delle monete.

`activeStep` è un intero che indica lo step corrente.

Le variabili di stato hanno, naturalmente, associate le funzioni per modificarne il loro valore.

Una volta caricata la pagina, con l'uso di `useEffect()`, viene richiamata la funzione `fetchCoins`.

```
1      const fetchCoins = async () => {  
2          setLoadingCoinList(true)  
3          const newCoinsList = await coinGetListOnlyOwned(  
4              web3Instance,userAccount);  
5          setLoadingCoinList(false);  
6          setCoinsList(newCoinsList);  
7          setActiveCoinsList(newCoinsList);  
8      }
```

Funzione `fetchCoins`

### 3.1.2 Box di ricerca

```
1      const handleSearch = (event) => {
2          const searchInput = event.target.value.toLowerCase();
3
4          setActiveCoinsList(coinsList.filter(coin => {
5              const coinName = coin.name.toLowerCase();
6              const coinSymbol = coin.symbol.toLowerCase();
7              if(coinName.includes(searchInput) ||
8                  coinSymbol.includes(searchInput)) return coin;
9          }));
10     }
11
12     <TextField
13         id="searchBox"
14         label="Search for a coin"
15         variant="outlined"
16         onChange={handleSearch}/>
```

Funzione fetchCoins

Il box di ricerca è realizzato usando il componente `<TextField>`, ad ogni inserimento/eliminazione di caratteri viene invocata la funzione `handleSearch`. Questa funzione, come visibile alla riga 1, prende come parametro l'evento lanciato. Quest'ultimo contiene il testo presente nel `<TextField>`, la stringa viene, quindi, convertita in caratteri minuscoli ed assegnata alla variabile `searchInput`. Alla riga 4 viene chiamata la funzione `filter` sull'array

`coinsList`, questo metodo ritorna un nuovo array dopo aver filtrato gli elementi della lista secondo un certo criterio. In questo caso, il criterio, visibile alle righe 7 e 8, è che il nome o il simbolo della moneta che si sta controllando contenga la stringa da cercare contenuta in `searchInput`. In caso positivo la moneta viene aggiunta all'array da ritornare. Una volta terminata la funzione di filtro si ha quindi una lista contenente solo monete che includono il termine di ricerca. Questa lista viene, quindi, assegnata alla variabile di stato `activeCoinsList` usando, perciò, la funzione `setActiveCoinsList()`. In questo modo verranno mostrate a video le monete filtrate.

### 3.1.3 Step 1: scelta dei token da vendere

1 Select the coins to sell 2 Select the coins for the payment 3 Create the sale

RESET NEXT

Search for a coin



 prova - PRO Balance: 60	<input type="text" value="Amount"/>
 mare - MAR Balance: 100	<input type="text" value="Amount"/>

Figura 3: Pagina per la scelta dei token in vendita

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

#### **3.1.3.1 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] IBM. *What are smart contracts on blockchain?* URL: <https://www.ibm.com/topics/smart-contracts>.
- [8] Metamask. *Metamask docs*. URL: <https://docs.metamask.io/guide/>.
- [9] Mui-org. *Material-UI docs*. URL: <https://v4.mui.com/customization/components/>.
- [10] OpenZeppelin. *OpenZeppelin docs*. URL: <https://docs.openzeppelin.com/contracts/4.x/>.
- [11] 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>.



- [12] Truffle suite. *Truffle suite docs*. URL: <https://www.trufflesuite.com/docs>.
- [13] Giuseppe Vanni. *Blockchain: cos'è, come funziona, tecnologia e applicazioni*. URL: <https://www.punto-informatico.it/blockchain-spiegazione/#h222751-0>.
- [14] Fabian Vogelsteller e Vitalik Buterin. *EIP-20: Token Standard*. URL: <https://eips.ethereum.org/EIPS/eip-20>.