Indice

1 Prerequisiti

1.1 Ethereum

1.2 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.

1.3 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.

1.4 ReactJS

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

1.4.1 Caratteristiche di ReactJS

1.4.1.1 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`:

```
function Convenevoli() {
  return (
    <div>
      <Saluto nome="Sara" />
      <Saluto nome="Cahal" />
      <Saluto nome="Edite" />
    </div>
  );
}
```

Esempio renderizzazione componente

1.4.1.2 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:

```
function Contatore() {
  const [contatore, setContatore] = useState(0);
  return (
    <div>
      <p>Hai cliccato {contatore} volte</p>
      <button
        onClick={() => setContatore(contatore + 1)}>
        Cliccami
      </button>
    </div>
  );
}

```

Esempio contatore con stato interno

1.4.1.3 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()`.

```
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()

1.5 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>`:

```
.Button {
  width: "100px",
  height: "100px"
}

<Button className="Button">

```

Esempio modifica aspetto di un componente

1.6 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

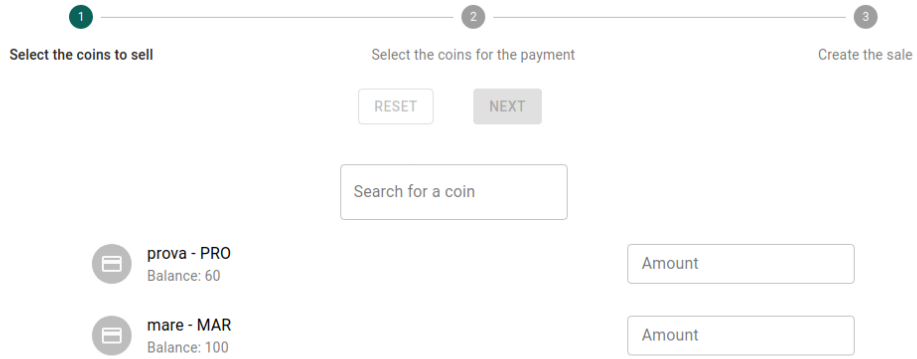

Figura 1: Pagina per la scelta dei token in vendita

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.