Relazione del Progetto relativo l'insegnamento di Sistemi Distribuiti per la sessione autunnale a.a. 2022/2023.

Corso di Laurea Magistrale in Informatica Applicata
Università degli studi di Urbino "Carlo Bo"

Relatori:
Matteo Marco Montanari (Matr. 323293)
Giacomo Di Fabrizio (Matr. 322856)

Docente:
Prof. Stefano Ferretti

1) Specifica di progetto

Si vuole creare un'applicazione distribuita (DApp), basata su un sistema Blockchain, per gestire la compravendita di biglietti per degli eventi. In questo modo la vendita risulta tracciabile, verificabile e non necessita di affidarsi a terze parti per la fruizione del servizio. Tale applicazione è strutturata come segue:

- Il lato backend si basa su uno smart contract per la Blockchain di Ethereum ed è scritto in linguaggio Solidity.
- Il lato frontend è invece una Web App standard (HTML, CSS, JavaScript) realizzata tramite il framework React.JS.
- Questa DApp prevede la possibilità, da parte del gestore dell'app (che accede tramite account di ammistratore), di creare dei nuovi eventi e di modificarli in qualsiasi momento. È possibile specificare il nome e la descrizione dell'evento ma anche il numero di biglietti acquistabili e il loro prezzo.
- Tramite il lato frontend, accessibile da un comune browser Web, un utente può acquistare un qualsiasi numero di biglietti per un determinato evento, a patto di pagare tramite la criptovaluta di Ethereum (Ether) il prezzo dei relativi biglietti e il costo delle commissioni per le transazioni blockchain.
- Sempre da browser Web è possibile, per chi ha già acquistato dei biglietti, rivenderli ad altri utenti al prezzo attuale specificato dal gestore dell'evento.
- È anche possibile ottenere il rimborso dei biglietti acquistati tramite apposita interfaccia Web (al prezzo attuale specificato dal gestore dell'evento).
- L'utente può accedere ad una serie di altre informazioni utili come il numero dei biglietti che ha acquistato per un determinato evento, il prezzo di tali biglietti, il numero totale di biglietti venduti e di biglietti ancora disponibili.
- L'amministratore può inoltre consultare il saldo proveniente dalla vendita dei biglietti e ritirarlo, inviandolo al proprio indirizzo Ethereum.

2) Funzionamento di una DApp in breve

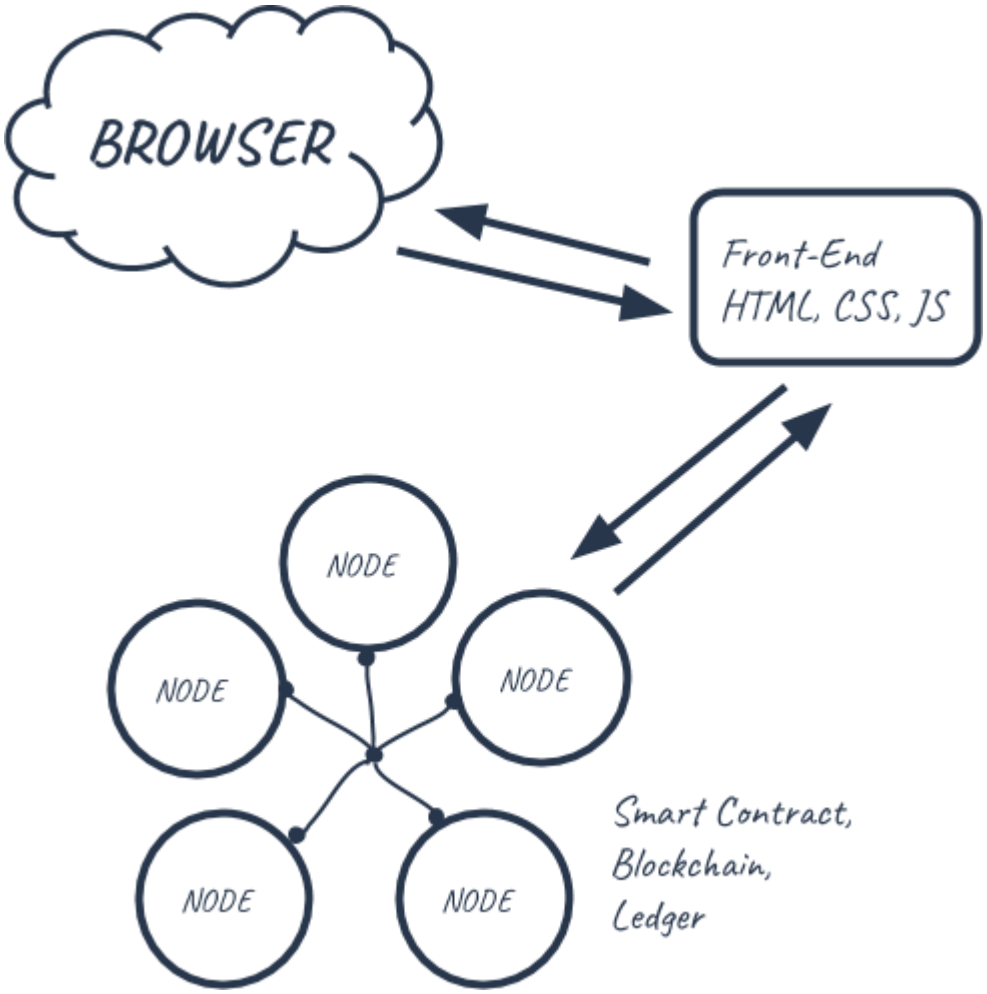
Le tecnologie blockchain come Bitcoin ed Ethereum si basano su reti di calcolatori di tipo Peer-to-Peer, e quindi su sistemi e protocolli distribuiti. Esse sono un esempio di Distributed Ledger Technologies, cioè tecnologie a registro distribuito, che in questo caso è rappresentato dalla blockchain ("catena di blocchi"). Quest'ultima è una struttura dati di tipo lista costruita tramite hash pointers piuttosto che semplici puntatori. Un hash pointer è una struttura dati costituita da un puntatore a dei dati e l'hash crittografico di tali dati, che ne garantisce l'integrità. Una blockchain è quindi una lista in cui ogni nodo mantiene l'hash del nodo precedente, il puntatore ad esso associato e dei dati interni, ciò garantisce che sia molto difficile modificare i dati che vengono memorizzati al suo interno. Questo accade perché una eventuale modifica di un blocco verrebbe subito individuata dai Peers in quanto invaliderebbe gli hash pointer di tutti i blocchi successivi. Una struttura dati di questo tipo garantisce dunque integrità, verificabilità e trasparenza dei dati in essa contenuti.

In un sistema basato su blockchain, i Peers lavorano insieme per verificare, validare e registrare le transazioni scrivendole sulla blockchain e implementano meccanismi di consenso per avere tutti una visione consistente della blockchain pur essendo programmi concorrenti che eseguono su nodi distinti collegati tramite Internet (ogni nodo ha la propria copia locale della blockchain).

Una evoluzione del sistema Bitcoin è Ethereum, una tecnologia a registro distribuito che permette di creare programmi Turing-completi che vengono codificati direttamente sulla blockchain e vengono eseguiti localmente da ciascun Peer su una macchina virtuale chiamata Ethereum Virtual Machine (EVM). Questi software sono chiamati Smart Contracts e possono essere programmati per interagire con la blockchain dietro pagamento di commissioni tramite la criptovaluta di Ethereum, l'Ether.

Una DApp (Distributed Application) è un esempio di applicazione distribuita il cui lato backend si basa su una Distributed Ledger Technology (DLT) su cui esegue uno Smart Contract, mentre l'accesso alla DApp avviene solitamente tramite una interfaccia utente fornita da un frontend Web o Mobile.

La DApp che vogliamo realizzare ha dunque la struttura seguente:



Tramite browser Web è possibile accedere al frontend dell'applicazione, che risiederà su un Server Web e che gestirà l'interfaccia utente. Il frontend sarà poi collegato tramite Internet ad una rete blockchain distribuita come Ethereum che, tramite uno specifico smart contract, fungerà da backend per l'applicazione scrivendo i dati direttamente sulla blockchain.

3) Tecnologie utilizzate

L'applicazione utilizza una serie di tecnologie tra cui:

- Per realizzare il frontend:
    - Lato Client si è scelto di creare una semplice pagina web come interfaccia utente tramite i linguaggi HTML, CSS e JavaScript. Per realizzarla si è fatto uso del framework React.JS.

- Per realizzare il backend:
    - Per poter eseguire e testare il backend dell'applicazione senza utilizzare direttamente la mainchain di Ethereum si è scelto di eseguire in locale una blockchain di test sulla quale sviluppare utilizzando il software Ganache.
    - Per codificare lo smart contract per EVM si è scelto di utilizzare il linguaggio Solidity e il framework di sviluppo Truffle.
    - Si è scelto di utilizzare il software Metamask, tramite estensione per browser, per gestire gli account Ethereum relativi agli utenti, confermare transazioni e comunicare con la blockchain.
    - Si è scelto di utilizzare la libreria JavaScript Web3.JS come interfaccia per comunicare con la blockchain locale fornita da Ganache e connettere il frontend al backend.

4) Avvio dell'applicativo (su Windows)

1. Installare i seguenti software:
    a. Node.JS (e NPM): https://nodejs.org
    b. Truffle: https://github.com/trufflesuite/truffle
    c. Ganache: http://truffleframework.com/ganache/
    d. Metamask (estensione per browser): https://metamask.io/
    e. Git (opzionale): https://git-scm.com/
2. Clonare il repository Git da Github:
    a. git clone https://github.com/GiacomoDiFa/ProgettoSD
3. Eseguire da terminale i seguenti comandi:
    a. $ cd progettoSD
    b. $ cd client
    c. $ npm install
4. Avviare il software Ganache tramite interfaccia grafica.
5. Eseguire da terminale i comandi:
    a. $ cd ..
    b. $ cd truffle
    c. $ truffle migrate
6. Eseguire da terminale i comandi per lanciare l'interfaccia utente:
    a. $ cd ..
    b. $ cd client
    c. $ npm start
7. Visitare l'URL: http://localhost:8080 tramite Google Chrome (per evitare bug).
8. Configurare Metamask (estensione per Google Chrome):
    a. Sbloccare l'account Metamask locale tramite password.
    b. Connettere Metamask alla blockchain Ethereum locale fornita da Ganache.
    c. Importare un account fornito da Ganache tramite chiave privata.
    d. Connettere l'account all'endpoint: http://localhost:8080 (utilizzare l'opzione nella sezione: Siti connessi > connettersi manualmente al sito).
9. Utilizzare l'interfaccia fornita dal browser per interagire con la DApp.

5) Codici dell'applicativo

5.1) Codice dello smart contract (metodi principali)

- *Funzione per creare un evento*

```
function createEvent(string memory name, string memory description, uint256 _totalTickets, uint256
ticketPrice) external onlyOwner {
        require(_totalTickets > 0, "Number of total tickets must be > 0");

        // totalEvents e' l'ID evento
        events[totalEvents] = Event(name, description, _totalTickets, 0, ticketPrice);
        totalEvents++;
    }
```

- *Funzione per modificare un evento*

```
function editEvent(uint256 eventId, string memory name, string memory description, uint256
_totalTickets, uint256 _ticketPrice) external onlyOwner {
        Event storage currentEvent = events[eventId];
        require(currentEvent.totalTickets > 0, "Event does not exist");
        require(_totalTickets >= currentEvent.ticketsSold, "Total tickets must be more than tickets
sold");

        currentEvent.name = name;
        currentEvent.description = description;
        currentEvent.totalTickets = _totalTickets;
        currentEvent.ticketPrice = _ticketPrice;
    }
```

- *Funzione per comprare dei biglietti*

```
function buyTickets(uint256 eventId, uint256 numTickets) external payable {
        Event storage currentEvent = events[eventId];
        require(currentEvent.totalTickets > 0, "Event does not exist");
        require(numTickets > 0, "You must buy at least one ticket");
        require(currentEvent.ticketsSold + numTickets <= currentEvent.totalTickets, "Not enough
tickets available for this event");
        require(msg.value >= currentEvent.ticketPrice * numTickets, "Insufficient payment");

        ticketsPurchased[msg.sender][eventId] += numTickets;
        currentEvent.ticketsSold += numTickets;

        uint amountToSend = currentEvent.ticketPrice * numTickets;
        uint change = msg.value - amountToSend;
        // gli ether dentro msg.value sono tutti inviati allo SC (default)
        // il resto al mittente
        payable(msg.sender).transfer(change);
```

```
        emit TicketPurchased(msg.sender, eventId, numTickets);
    }
```

- *Funzione per rivendere biglietti*

```
// chi chiama la funzione paga ed e' il buyer
function resellTickets(address seller, uint256 eventId, uint256 numTickets) external payable {
        Event storage currentEvent = events[eventId];
        require(currentEvent.totalTickets > 0, "Event does not exist");
        require(msg.sender != seller, "Seller and Buyer must be different");
        require(numTickets > 0, "You must buy at least one ticket");
        require(ticketsPurchased[seller][eventId] >= numTickets, "Not enough tickets available");
        require(msg.value >= currentEvent.ticketPrice * numTickets, "Insufficient payment");

        // vedi funzioni successive
        require(ticketsToResell[seller][eventId] >= numTickets, "Not enough tickets to resell");

        ticketsToResell[seller][eventId] -= numTickets;

        ticketsPurchased[seller][eventId] -= numTickets;
        ticketsPurchased[msg.sender][eventId] += numTickets;

        // msg.value va tutto allo SC che poi inoltra i soldi
        uint amountToSend = currentEvent.ticketPrice * numTickets;
        uint change = msg.value - amountToSend;
        // Transfer the amount to the seller
        payable(seller).transfer(amountToSend);
        // il resto al mittente
        payable(msg.sender).transfer(change);

        emit TicketReselled(seller, msg.sender, eventId, numTickets);
    }
```

- *Funzioni per esprimere la volontà di rivendere biglietti*

```
// flag per sapere quanti biglietti voglio vendere
// ticketsToResell[msg.sender][eventId] == numTicketsToResell
mapping(address => mapping(uint256 => uint256)) public ticketsToResell;
event WantToResell(address indexed reseller, uint256 eventId, uint256 numTickets);

function wantToResell(uint256 eventId, uint256 _numTickets) external {
   require(ticketsPurchased[msg.sender][eventId] >= _numTickets, "Not enough tickets to resell");
   ticketsToResell[msg.sender][eventId] += _numTickets;

   emit WantToResell(msg.sender, eventId, _numTickets);
}
```

```
// getter per il msg.sender
function getMyTicketsToResell(uint256 eventId) external view returns (uint256) {
    return ticketsToResell[msg.sender][eventId];
}
```

- *Funzione per ottenere il rimborso dei biglietti acquistati*

```
// rimborso ticket
function refundTickets(uint256 eventId, uint256 numTickets) external {
    Event storage currentEvent = events[eventId];
    require(currentEvent.totalTickets > 0, "Event does not exist");
    require(numTickets > 0, "You must refund at least one ticket");
    require(ticketsPurchased[msg.sender][eventId] >= numTickets, "You don't have enough tickets to
refund");

    uint256 balance = address(this).balance;
    uint256 refundAmount = currentEvent.ticketPrice * numTickets;
    require(balance >= refundAmount, "Smart Contract balance is not sufficient");

    ticketsPurchased[msg.sender][eventId] -= numTickets;
    currentEvent.ticketsSold -= numTickets;

    // Transfer the refund amount to the buyer (paga lo SC)
    payable(msg.sender).transfer(refundAmount);
}
```

5.2) Codice del frontend

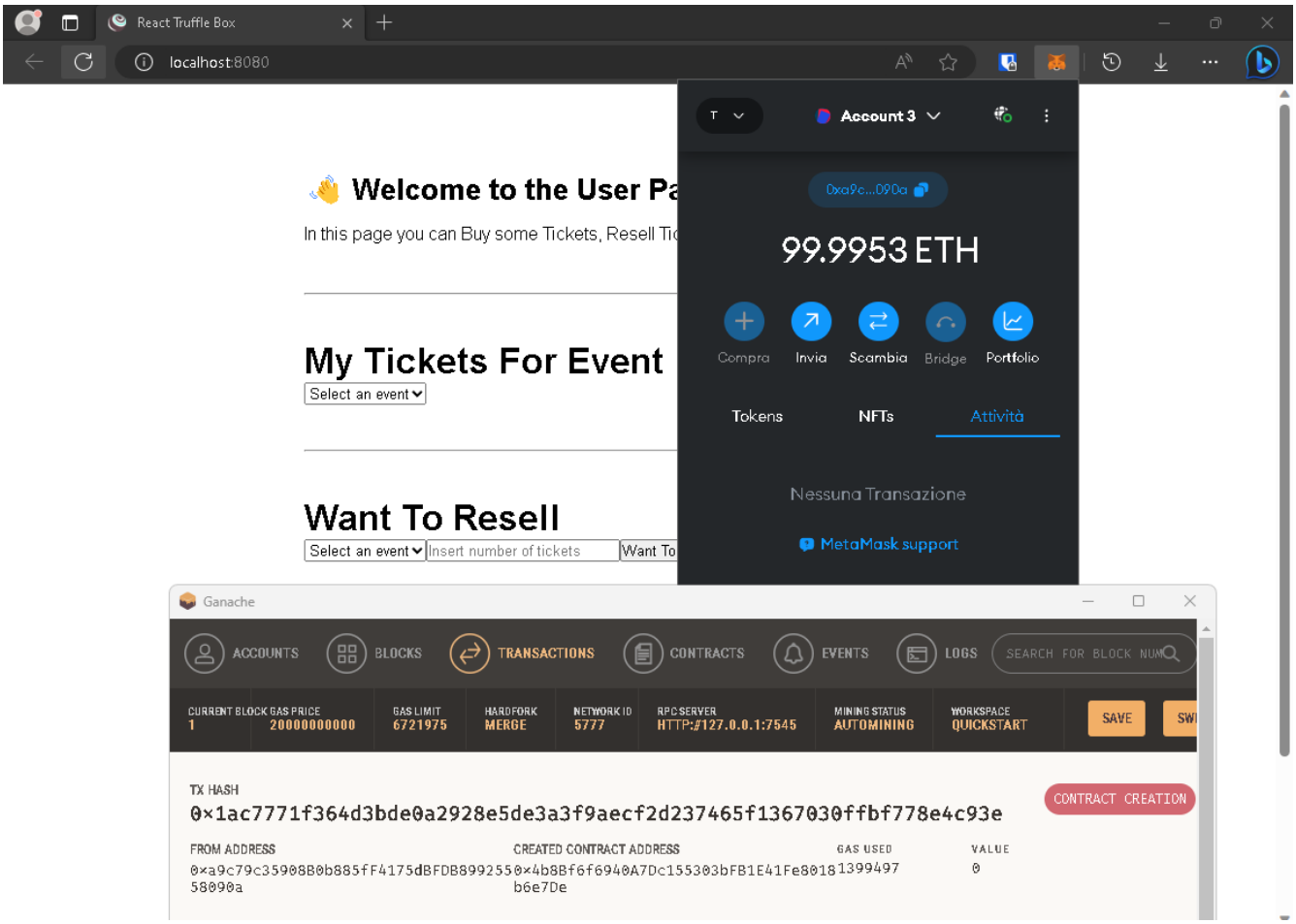
Il codice per il frontend dell'applicazione, realizzato tramite il framework React.JS (HTML, CSS, JavaScript) è reperibile al repository pubblico: https://github.com/GiacomoDiFa/ProgettoSD.

6) Dimostrazione tramite esempi

1. Deploy dello smart contract

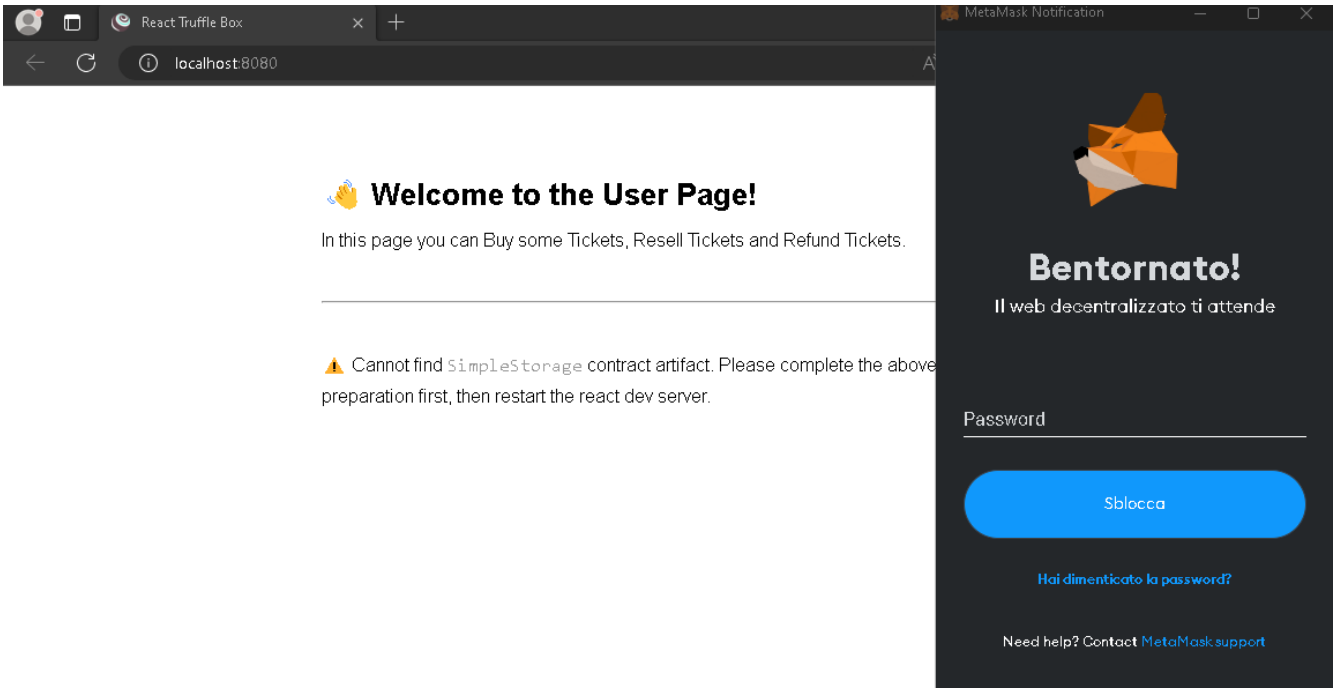
All'avvio dell'applicazione viene fatto il deploy dello smart contract sulla blockchain da parte dell'account di amministratore (account 3, **0xa9c79c35908B0b885fF4175dBFDB89925558090a**) che deve quindi pagare delle commissioni per scrivere la relativa transazione sulla blockchain.

L'indirizzo dello smart contract è **0x4b8Bf6f6940A7Dc155303bFB1E41Fe8018b6e7De**.



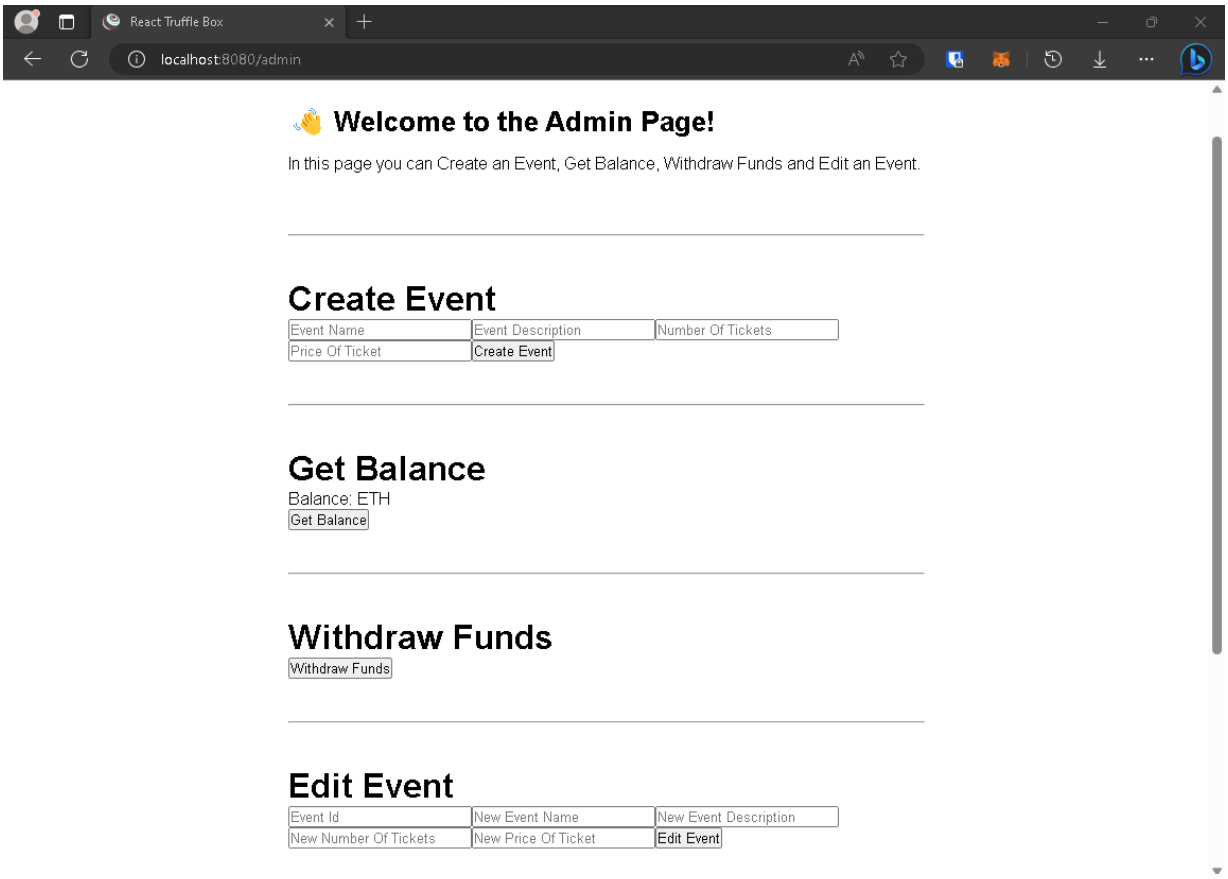
2. *Schermata di accesso all'applicazione*

Al primo accesso all'app (tramite browser) viene richiesto l'accesso all'account Metamask per gestire gli account Ethereum e confermare le transazioni blockchain.



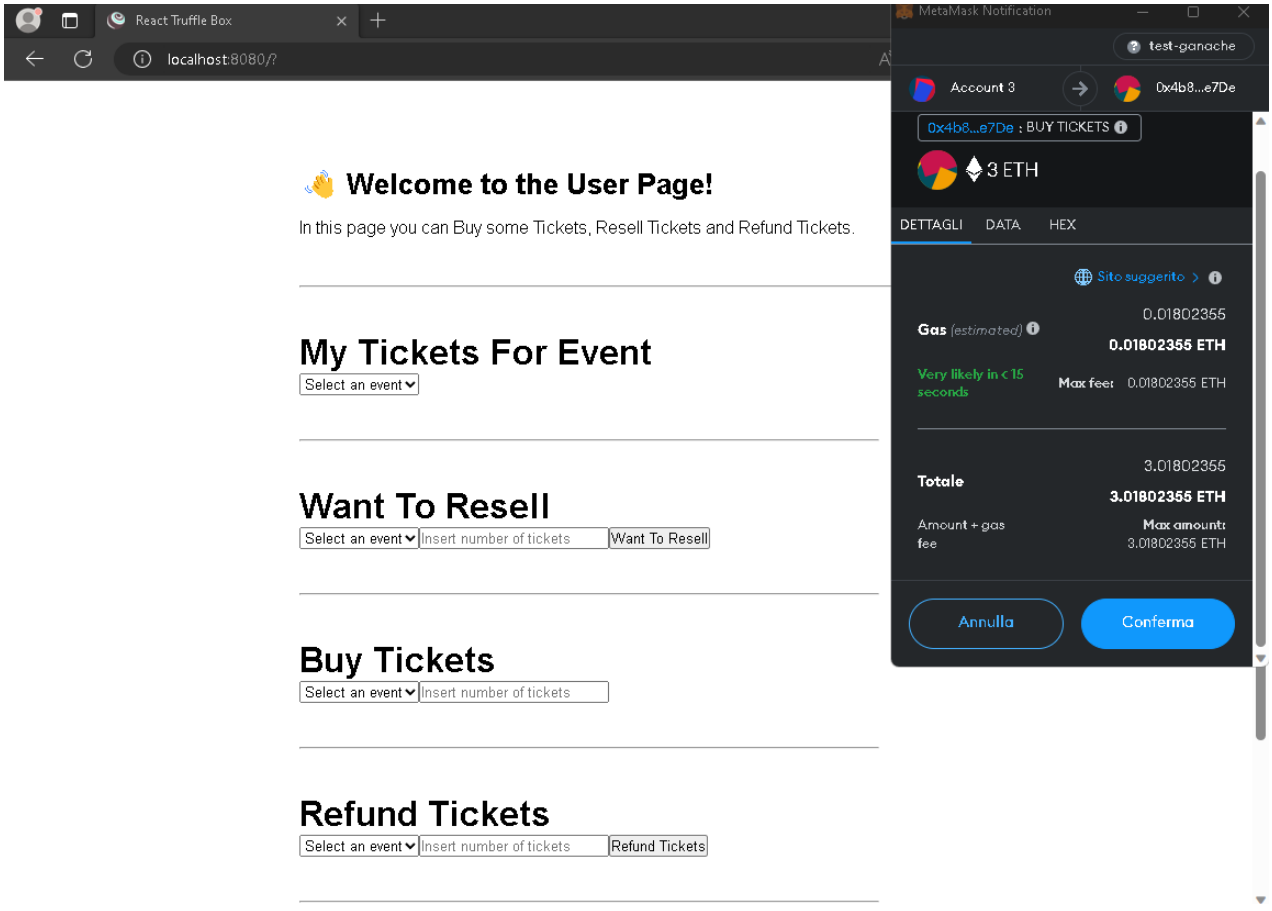
3. *Interfaccia utente della pagina di amministratore*

L'interfaccia utente utilizzata dall'amministratore è raggiungibile tramite browser all'indirizzo: [http://localhost:8080/admin.](http://localhost:8080/admin)



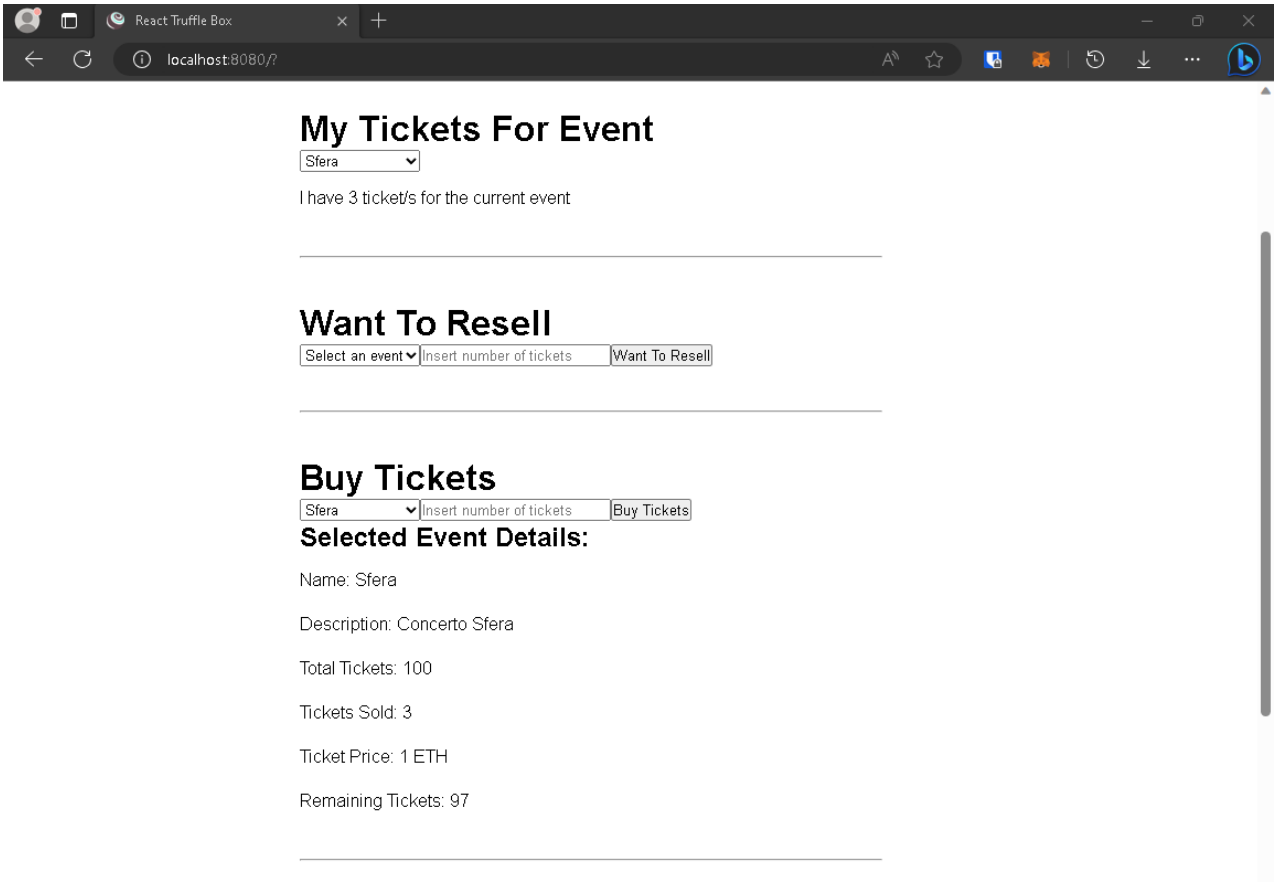
4. *Pop-up di conferma della transazione al costo di 3 ETH.*

L'account 3 (**0xa9c79c35908B0b885fF4175dBFDB89925558090a**) procede all'acquisto di 3 biglietti per l'evento "Concerto Sfera". È necessario dare la conferma tramite Metamask pagando 3 ETH.



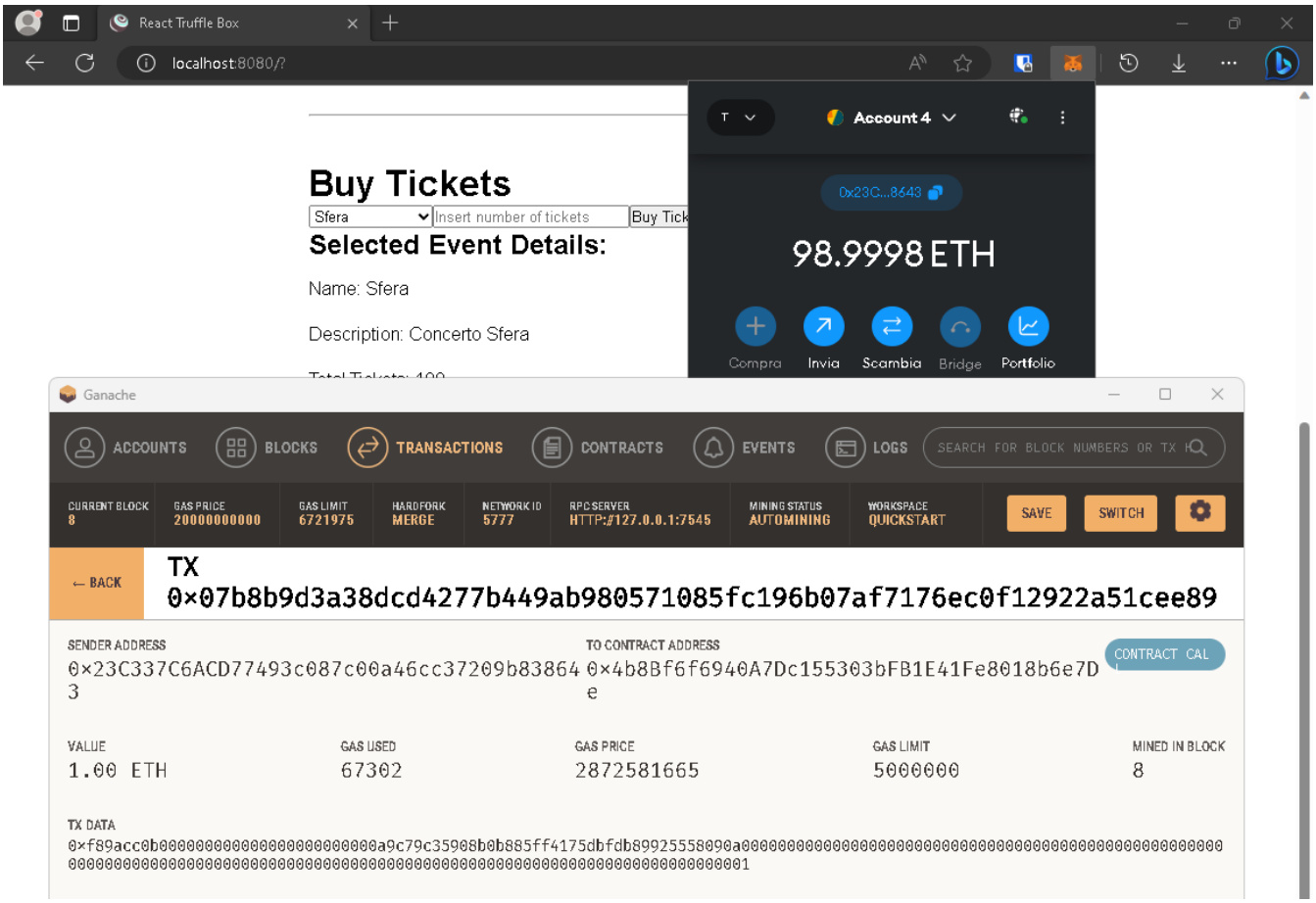
5. *Interfaccia utente per visualizzare i dettagli sull'evento selezionato*

Nella sezione "Buy Tickets" è possibile selezionare l'evento "Sfera" per visualizzare i dettagli dell'evento tra cui il numero di biglietti venduti, disponibili e il prezzo.



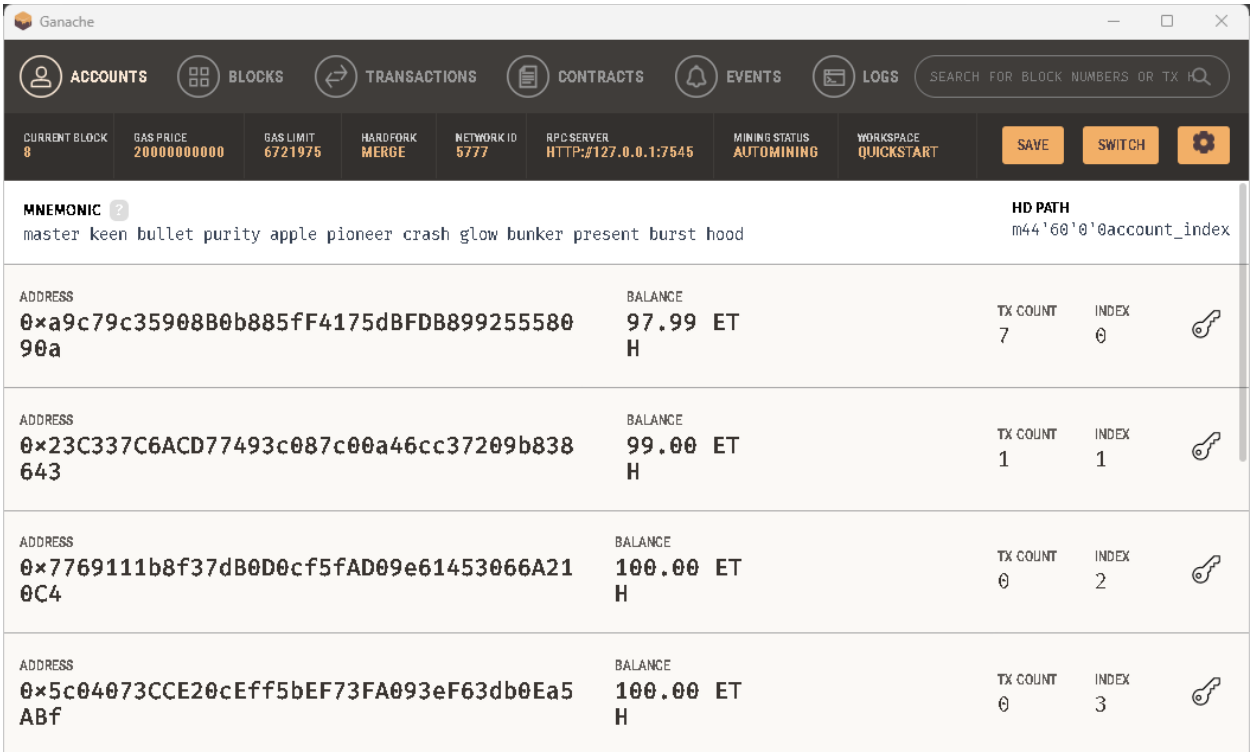
6. *Transazione per comprare un biglietto da un altro utente*

Tramite l'apposita interfaccia utente l'account 4 (**0x23C337C6ACD77493c087c00a46cc37209b838643**) chiede di poter comprare 1 biglietto per l'evento "Sfera" (al prezzo di 1 ETH) messo in vendita precedentemente dall'account 3 (**0xa9c79c35908B0b885fF4175dBFDB89925558090a**). Invia quindi una transazione, assieme ad 1ETH, allo smart contract (**0x4b8Bf6f6940A7Dc155303bFB1E41Fe8018b6e7De**) che procede a inoltrare il biglietto e pagare il venditore.



7. *Saldo dei vari account su blockchain*

Schermata di Ganache con i bilanci espressi in ETH per i due account coinvolti in questo esempio: l'account 3 (indirizzo index 0) e l'account 4 (indirizzo index 1). L'account 3 ha acquistato 3 biglietti (3 ETH) e ne ha rivenduto 1 all'account 4 (1 ETH). I saldi iniziali erano di 100 ETH per ciascun account.



7) Riferimenti Bibliografici

[The Ultimate Ethereum Dapp Tutorial (How to Build a Full Stack Decentralized Application Step-By-Step) | Dapp University]

Termine della relazione