TickeChain

1. Descrizione Generale

TickeChain è una piattaforma **decentralizzata** basata su **Ethereum** che offre una soluzione innovativa per la gestione di eventi e la distribuzione di biglietti digitali in forma di **NFT (Non-Fungible Token)**. Il sistema consente agli utenti di **creare eventi**, **emettere biglietti digitali**, **gestire vendite e rimborsi**, e **verificare la validità dei biglietti** in modo **sicuro, trasparente e senza intermediari**.

Grazie alla **blockchain**, ogni operazione all'interno della piattaforma è **immutabile e verificabile**, garantendo **fiducia e protezione** sia per gli organizzatori che per i partecipanti. TickeChain elimina la necessità di terze parti, offrendo un **modello completamente decentralizzato** in cui tutte le transazioni avvengono **on-chain**, senza il rischio di frodi, duplicazioni o scalping illegale dei biglietti.

L'architettura del sistema è progettata per essere:

- **Sicura**: grazie all'uso di smart contract Solidity verificati, che garantiscono la correttezza e la protezione delle transazioni.
- **Scalabile**: supportando una gestione efficiente degli eventi con possibilità di espandere il supporto ad altre blockchain Layer 2 (Polygon, Arbitrum) per ridurre i costi di transazione.
- **Flessibile**: con un'integrazione semplice e modulare, che permette agli sviluppatori di estendere le funzionalità.
- **Accessibile**: basata su un'interfaccia utente intuitiva che permette a qualsiasi utente, anche senza conoscenze tecniche, di creare e gestire eventi in pochi passaggi.

TickeChain rappresenta quindi una **soluzione all'avanguardia per l'industria degli eventi**, offrendo **trasparenza, efficienza e decentralizzazione**, eliminando le limitazioni dei tradizionali sistemi di ticketing e creando un ecosistema affidabile e innovativo per gli organizzatori e i partecipanti.

---

2. Funzionalità Implementate

2.01 Creazione e Gestione degli Eventi

Gli utenti possono creare eventi personalizzati fornendo i seguenti dettagli:

- **Nome dell'evento**
- **Luogo**
- **Descrizione**
- **Data (timestamp UNIX, solo futura)**
- **Prezzo dei biglietti in ETH**
- **Numero totale di biglietti disponibili**

Ogni evento segue un flusso controllato tramite **State Machine**:

- **CREATED** → evento appena creato, non ancora aperto per la vendita.
- **OPEN** → biglietti disponibili per l'acquisto.
- **CLOSED** → evento chiuso, non più disponibile.
- **CANCELLED** → evento annullato, con rimborsi per gli utenti.

2.01.01 Controlli di sicurezza:

- Un evento può essere **modificato o cancellato solo dal suo creatore**.
- Un evento può passare a **OPEN solo se la data non è passata**.
- Se troppi eventi vengono annullati in poco tempo, il sistema **attiva automaticamente la modalità di emergenza (Circuit Breaker)**.

2.02 Emissione e Gestione di Biglietti NFT

I biglietti vengono gestiti come **token NFT ERC-721**, garantendo:

- **Unicità e autenticità**
- **Tracciabilità trasparente su blockchain**

2.02.01 Operazioni supportate:

- **Minting:** Generazione di biglietti NFT collegati a un evento.
- **Validazione:** Verifica on-chain dei biglietti all'ingresso.
- **Rimborso:** Gli utenti possono bruciare i loro biglietti per ottenere un rimborso in caso di evento annullato.
- **Verifica dello stato:** I biglietti non possono essere rimborsati se già validati.

2.02.02 Protezione avanzata:

- Un biglietto può essere **validato una sola volta**.
- Se si verificano **troppi errori di minting**, il contratto si **autopausa** per prevenire attacchi.

2.03 Sistema di Pagamenti e Rimborsi

Il sistema supporta pagamenti in **ETH nativo** e gestisce i fondi in modo sicuro.

2.03.01 Gestione dei pagamenti:

- I fondi per l'acquisto di biglietti vengono **bloccati nel contratto fino alla chiusura dell'evento**.
- Il creatore dell'evento può **ricevere i fondi accumulati solo dopo la chiusura dell'evento**.

2.03.02 Gestione dei rimborsi:

- Gli utenti possono **richiedere un rimborso manuale** se l'evento viene annullato.
- Il rimborso avviene **solo se il contratto ha fondi disponibili**.
- Se il rimborso fallisce per fondi insufficienti, il sistema **attiva automaticamente la modalità di emergenza (Emergency Stop)**.

3. Pattern e Approcci Utilizzati

TickeChain utilizza diversi **design pattern e strategie** per garantire scalabilità e sicurezza:

- **Registry Pattern**: Organizza e centralizza i dati degli eventi.
- **Pull Payment Pattern**: Riduce i rischi di reentrancy nei rimborsi.
- **Circuit Breaker (Emergency Stop)**: Protegge il sistema da malfunzionamenti critici.
- **Checks-Effects-Interactions**: Evita attacchi di reentrancy.
- **State Machine**: Regola le fasi degli eventi per evitare stati incoerenti.
- **Secure Ether Transfer**: Garantisce sicurezza nei trasferimenti di ETH.
- **Guard Check**: Controllo rigoroso dei parametri prima di ogni operazione.

4. Dettaglio dei Smart Contract Implementati

TickeChain utilizza quattro contratti principali per gestire il sistema:

4.01 EventFactory.sol

Il contratto permette agli utenti di **creare, modificare e cancellare eventi**, definendo un ciclo di vita chiaro attraverso una **State Machine** (CREATED, OPEN, CLOSED, CANCELLED). Solo il creatore dell'evento può modificarne i dettagli prima che venga aperto alla vendita. Implementa un **Circuit Breaker**, che blocca temporaneamente il sistema se troppi eventi vengono annullati in poco tempo, garantendo stabilità alla piattaforma.

4.01.01 Funzioni:

- `createEvent()`: Crea un nuovo evento fornendo nome, luogo, data, prezzo del biglietto e numero di biglietti disponibili.
- `updateEvent()`: Permette al creatore di un evento di modificarne i dettagli se lo stato è CREATED.
- `deleteEvent()`: Consente al creatore di eliminare un evento se non è ancora stato aperto per la vendita.
- `changeEventState()`: Modifica lo stato dell'evento (CREATED → OPEN, OPEN → CLOSED, CANCELLED).
- `cancelEvent()`: Annulla un evento e attiva la funzione di rimborso.
- `pause() / unpause()`: Blocca e riattiva la gestione degli eventi in caso di emergenza.

4.02 EventRegistry.sol

Questo contratto memorizza e organizza tutti gli eventi creati, permettendo di **elencarli e filtrarli per organizzatore**. Gli eventi possono essere eliminati solo dal loro creatore o dall'owner del contratto.
L'uso di strutture dati ottimizzate migliora l'efficienza della ricerca, riducendo i costi di gas e garantendo una gestione fluida delle informazioni.

4.02.01 Funzioni:

- `listEvents()`: Restituisce l'elenco di tutti gli eventi registrati sulla piattaforma.
- `findEventsByCreator()`: Filtra gli eventi creati da un determinato utente.
- `deleteEvent()`: Permette al creatore o all'owner del contratto di rimuovere un evento dal registro.

4.03 PaymentManager.sol

I fondi degli utenti vengono **bloccati nel contratto** fino alla chiusura dell'evento. Se un evento viene **annullato**, gli utenti possono **richiedere un rimborso manuale**.
Per evitare problemi di liquidità, il contratto include un **Emergency Stop**, che si attiva automaticamente se i fondi disponibili non sono sufficienti per coprire i rimborsi.

4.03.01 Funzioni:

- `depositFunds()`: Permette agli utenti di depositare ETH nel contratto per acquistare biglietti.
- `processRefund()`: Esegue un rimborso a un utente specifico se l'evento è stato annullato.
- `releaseFundsToCreator()`: Rilascia i fondi accumulati al creatore dell'evento dopo la chiusura delle vendite.
- `pause() / unpause()`: Permette di bloccare temporaneamente le operazioni in caso di emergenza finanziaria.

4.04 TicketManager.sol

Il contratto si occupa della **generazione, validazione e rimborso** dei biglietti NFT. Ogni biglietto è un token unico legato a un evento specifico.
I biglietti possono essere **validati all'ingresso** e, in caso di annullamento dell'evento, rimborsati e **bruciati** per impedirne il riutilizzo.

4.04.01 Funzioni:

- `mintTicket()`: Genera un biglietto NFT associato a un evento e lo assegna all'acquirente.
- `markTicketAsVerified()`: Convalida il biglietto al momento dell'ingresso all'evento.
- `refundTicket()`: Consente all'utente di richiedere il rimborso, bruciando il biglietto NFT.
- `pause() / unpause()`: Blocca o riattiva la gestione dei biglietti in caso di emergenza.

5. Architettura del Frontend e Integrazione con gli Smart Contract

- **Framework:** Il frontend è costruito con **React e Vite** per garantire una UI moderna e reattiva.
- **Gestione dello stato:** Utilizzo di React State per tracciare lo stato della connessione dell'utente e delle transazioni.
- **Integrazione con MetaMask:**
  - Connessione con il wallet dell'utente.
  - Gestione delle transazioni tramite **Ethers.js**.
  - Interfaccia per mostrare eventi disponibili, biglietti acquistati e fondi disponibili.
- **Esempio di chiamata smart contract:**

```
1. const contract = new ethers.Contract(contractAddress, contractABI, signer);
2. await contract.mintTicket(eventId);
```

6. Verifica e Test Automatizzati

Per garantire la sicurezza e il corretto funzionamento degli smart contract di **TickeChain**, sono stati implementati test automatizzati con **Hardhat**, **Chai** e **Mocha**. Questi test verificano il comportamento dei singoli contratti e le loro funzionalità principali.

6.01 Test dei Contratti Singoli

6.01.01 EventFactoryTest.js

- Testa la **creazione di eventi**, assicurando che tutti i dati forniti (nome, luogo, data, prezzo, biglietti disponibili) vengano registrati correttamente.
- Verifica la possibilità di **modificare un evento**, assicurandosi che solo il creatore possa farlo.
- Controlla la **cancellazione di un evento**, eliminandolo dal sistema e attivando eventuali rimborsi.
- Simula la **transizione degli stati** (CREATED → OPEN → CLOSED → CANCELLED), verificando che le modifiche siano applicate correttamente.
- Testa l'**Emergency Stop**, che si attiva automaticamente dopo tre cancellazioni consecutive di eventi.

6.01.02 EventRegistryTest.js

- Simula la **registrazione degli eventi**, verificando che nome, luogo, data e creatore siano memorizzati correttamente.
- Controlla che il contratto possa **elencare tutti gli eventi registrati**.
- Testa il **filtro per creatore**, assicurandosi che gli eventi siano assegnati all'utente corretto.
- Simula l'**eliminazione di un evento**, garantendo che venga rimosso dal registro.

6.01.03 PaymentManagerTest.js

- Verifica il **deposito di fondi**, assicurandosi che l'importo venga registrato correttamente per ogni utente.
- Testa il **rimborso**, trasferendo i fondi indietro all'utente dopo un'operazione simulata.
- Simula il **rilascio dei fondi all'organizzatore dell'evento** dopo la chiusura della vendita.
- Controlla il funzionamento dell'**Emergency Stop**, verificando che possa bloccare e riprendere le operazioni.

6.01.04 TicketManagerTest.js

- Simula la **creazione e assegnazione di biglietti NFT**, verificando che l'indirizzo del proprietario e il metadata URI siano corretti.
- Testa il **rimborso con bruciatura del biglietto**, assicurandosi che il biglietto venga eliminato dalla blockchain dopo il rimborso.
- Controlla il sistema di **validazione dei biglietti**, impedendo che lo stesso biglietto possa essere riutilizzato.
- Verifica lo stato dei biglietti attivi, assicurandosi che un biglietto sia **valido fino alla sua validazione o al rimborso**.
- Tutti i test sono stati eseguiti su una rete locale con **Ganache**, coprendo scenari realistici e edge cases per garantire la robustezza del sistema.

7. Software e Tecnologie Utilizzate

- **Solidity v0.8.28:** Linguaggio di programmazione per gli smart contract.
- **OpenZeppelin Contracts:** Librerie di sicurezza per Solidity.
- **Hardhat:** Strumento per lo sviluppo, test e deploy degli smart contract.
- **Ethereum (Ganache per testing):** Blockchain utilizzata per la gestione degli eventi.
- **React (Frontend, UI/UX):** Framework per l'interfaccia utente.
- **Ethers.js:** Libreria per interagire con gli smart contract da frontend.
- **Bootstrap:** Framework per il design dell'interfaccia utente.

8. Conclusione

TickeChain rappresenta una **soluzione innovativa e completamente decentralizzata** per la gestione di eventi e biglietti NFT. Grazie a un'architettura robusta e a **meccanismi avanzati di sicurezza**, il sistema garantisce **trasparenza, efficienza e protezione dagli attacchi**. Le future ottimizzazioni permetteranno di migliorare ulteriormente l'usabilità e la scalabilità della piattaforma.