Prediction Market – Smart Contract Fundamentals (BNB Chain Testnet)

Overview

Questo documento riassume in modo strutturato tutto il flusso di progettazione, test e interazione tra un website basato su Supabase/Vercel e gli smart contract del prediction market su BNB Chain testnet.

1. Setup dell'Ambiente con Hardhat

Hardhat è il framework principale per lo sviluppo, test e deploy di smart contract. Serve solo in fase di sviluppo locale per compilare, testare e pubblicare i contratti. Su Vercel (frontend) non viene eseguito: il sito usa solo l'indirizzo e l'ABI del contratto già deployato.

2. Collegamento con la Testnet BNB

Il wallet (Rabby/MetaMask) si collega alla rete testnet tramite RPC pubblico. Le transazioni sono firmate dal wallet e confermate in pochi secondi. Serve una piccola quantità di tBNB ottenuta dal faucet per coprire le gas fee di test.

3. Ruolo di RainbowKit e Ethers.js

RainbowKit gestisce la connessione dei wallet (Connect Wallet), mentre ethers.js è la libreria che dialoga con la blockchain. RainbowKit fornisce il signer (utente connesso) ed ethers.js lo usa per leggere o scrivere dati nel contratto.

4. Firma e Attesa della Conferma On-Chain

Quando l'utente firma una transazione, ethers.js invia la tx e rimane in attesa con `await tx.wait()`. Questo comando monitora la rete RPC fino a quando la transazione non viene inclusa in un blocco. Solo dopo la conferma i dati vengono registrati su Supabase.

5. Gestione dei Dati con Supabase

Supabase registra le azioni utente (es. voto Sì/No) solo dopo la conferma on-chain. I dati in Supabase servono per aggiornamenti e statistiche live, ma la verità assoluta è sempre on-chain.

6. Ruolo della Chiave Privata

La chiave privata serve solo localmente (Hardhat) per fare il deploy del contratto. Non viene mai caricata su GitHub o Vercel. Durante l'uso del sito, le transazioni vengono firmate dal wallet dell'utente o dell'admin.

7. Fondi e Controllo del Contratto

I fondi inviati dagli utenti (es. 0.1 BNB) vengono depositati nel contratto stesso. Il contratto ha un saldo visibile su BscScan, ma non può essere gestito manualmente via wallet. Solo le funzioni del contratto, eseguibili da chi è autorizzato, possono spostarli.

8. Impostazione dell'Esito ('Sì' o 'No')

Il contratto non può sapere da solo chi ha vinto. L'owner imposta manualmente l'esito con una funzione `setWinner()`. Da quel momento, i vincitori possono fare 'claim' e ricevere automaticamente la loro parte.

9. Distribuzione delle Vincite

Il contratto conosce per ogni utente quanto ha puntato e su quale lato. Dopo l’impostazione dell’esito, calcola le quote in base alle puntate totali e distribuisce automaticamente ai vincitori proporzionalmente alla loro puntata.

10. Gas Fee

Le gas fee non sono mai pagate dal contratto ma da chi invia la transazione. L’admin paga le fee per impostare l’esito, gli utenti pagano per fare claim. Il saldo del contratto serve solo per i fondi delle scommesse.

11. Storage On-Chain

Ogni contratto possiede uno spazio di archiviazione permanente nella blockchain. Tutti i dati (puntate, utenti, importi) vengono salvati nello storage del contratto e replicati su tutti i nodi della rete. Scrivere in questo spazio costa gas, quindi si memorizza solo l’essenziale.

12. Integrità dei Dati e Sicurezza

Il contratto legge sempre i valori reali delle transazioni (`msg.value`) e registra solo le operazioni confermate nella blockchain. Non è possibile simulare o falsificare depositi: se la tx non è confermata, non esiste alcuna registrazione.

Conclusione

In un prediction market decentralizzato, il sito serve solo come interfaccia. Tutta la logica di fiducia, i fondi e i risultati vivono dentro lo smart contract. Supabase e Vercel gestiscono solo l’esperienza utente e i dati off-chain complementari.