Trust App

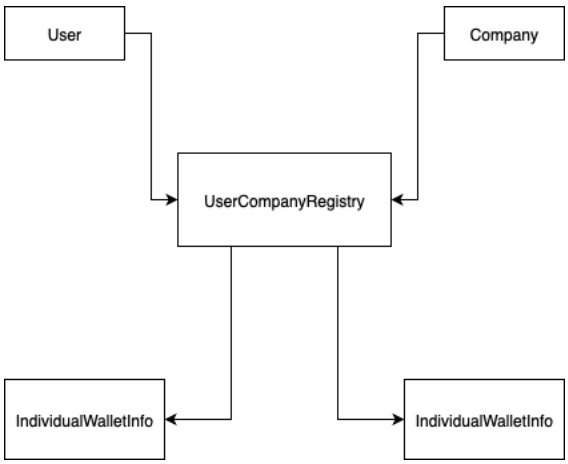
Trust App è un'applicazione decentralizzata (dApp), sviluppata per operare sulla blockchain di Ethereum (attualmente sulla testnet Sepolia).
Lo scopo è quello di gestire la creazione, la gestione e la visualizzazione di Smart Insurances: contratti assicurativi auto-eseguibili e immutabili. Questi contratti, sottoscritti direttamente tra una compagnia assicurativa e un utente, operano a fronte del pagamento di un premio e garantiscono l'emissione automatica di un rimborso (il "payout") al verificarsi di determinate clausole prestabilite.

Le Smart Insurances gestite da Trust App si basano su dati ambientali reali, recuperati da sensori specifici – in particolare la temperatura atmosferica (`saref:Temperature`) e l'umidità dell'aria (`s4agri:AmbientHumidity`). Per ogni polizza, viene stabilito un `target_value` che rappresenta la soglia di attivazione del rimborso, e vengono definite coordinate spaziali precise con un raggio circolare per identificare la zona geografica assicurata.

Un pilastro fondamentale dell'affidabilità e della sicurezza di Trust App è la sua integrazione profonda con **Zonia Oracle System**. Zonia è un sistema Oracle decentralizzato per l'acquisizione sicura e decentralizzata di dati off-chain (dal mondo reale) e la loro successiva registrazione on-chain. Questo meccanismo garantisce che le decisioni di payout siano basate su dati verificati e non alterabili.

1. Architettura dei Contratti Smart Fondamentali

Lo scheletro di Trust App è un ecosistema di contratti smart interconnessi, ognuno con un ruolo ben definito che contribuisce alla logica complessiva della piattaforma. Questi contratti sono il cuore immutabile e decentralizzato dell'applicazione.



1.1. UserCompanyRegistry

Il contratto `UserCompanyRegistry` è il gateway principale per l'applicazione, agendo come un registro centralizzato per tutte le entità che interagiscono con l'applicazione: sia gli utenti privati (assicurati) che le compagnie assicurative.

- Funzioni e Caratteristiche**:**
    - **Punto di Ingresso per Registrazione e Autenticazione:** È il contratto dal quale passa l'intero flusso di onboarding. Quando un nuovo wallet si connette a Trust App, `UserCompanyRegistry` è il primo contratto interrogato per verificarne lo stato di registrazione.
    - **Mapping Wallet-Info:** Mantiene un mapping pubblico che associa ogni indirizzo di wallet registrato, a un proprio, unico contratto `IndividualWalletInfo`. Fondamentale per recuperare rapidamente le informazioni e il ruolo (utente o compagnia) associati a un dato wallet.
    - **Creazione di IndividualWalletInfo:** Per i wallet che si registrano per la prima volta, `UserCompanyRegistry` si occupa del deploy di una nuova istanza del contratto `IndividualWalletInfo`. Questa operazione crea un "profilo on-chain" per il wallet appena connesso.
    - **Recupero Indirizzi:** In fase di accesso per utenti già registrati, il contratto restituisce semplicemente l'indirizzo del contratto `IndividualWalletInfo` già esistente, permettendo all'applicazione di recuperare autorizzazioni e insurances.

1.2. IndividualWalletInfo

Ogni utente o compagnia registrata su Trust App possiede una propria istanza del contratto `IndividualWalletInfo`. Questo contratto rappresenta un profilo digitale, contenente le informazioni essenziali e l'associazione specifica del wallet proprietario.

- **Funzioni e Caratteristiche:**
    - **Informazioni di Base del Wallet:** Memorizza dati fondamentali come il ruolo assegnato al wallet (`User` / `Company`). Questo ruolo determina l'interfaccia utente e le funzionalità disponibili nell'applicazione frontend.
    - **Gestione delle Polizze:** Contiene un elenco degli indirizzi di tutte le Smart Insurances che sono state sottoscritte da o emesse per quel wallet.

1.3. SmartInsurance

Il contratto `SmartInsurance` è la struttura di una singola polizza assicurativa. Ogni volta che una nuova assicurazione viene creata tramite Trust App, viene effettuato il deploy di una nuova istanza di questo contratto.

- **Funzioni e Caratteristiche:**
    - **Parametrizzazione Dettagliata:** Ogni istanza di `SmartInsurance` è meticolosamente configurata con parametri chiave che definiscono l'intera polizza:

        - `userWallet`: L'indirizzo del wallet dell'assicurato.
        - `companyWallet`: L'indirizzo del wallet della compagnia assicurativa emittente.
        - `premiumAmount`: L'importo del premio che l'assicurato deve pagare, espresso in `TulToken`.
        - `payoutAmount`: L'importo del rimborso che sarà erogato in caso di attivazione, anch'esso in `TulToken`.
        - `sensor`: Il tipo di dato del sensore che innescherà la polizza.
        - `target_value`: Il valore soglia numerico che, se raggiunto o superato dal dato del sensore, attiverà il rimborso.
        - `geoloc`: Le coordinate geografiche (latitudine e longitudine) e il raggio di pertinenza della polizza.
        - `query`: La stringa di query specifica che il contratto `Gate` userà per richiedere i dati all'Oracle.
        - `tokenAddress`: L'indirizzo del contratto del `TulToken` utilizzato per i pagamenti, garantendo che la polizza interagisca con il token corretto.
    - **Gestione del Ciclo di Vita (Stati):** Il contratto implementa una macchina a stati per tracciare il progresso della polizza:
        - `0: Pending`: La polizza è stata creata ma il premio non è ancora stato pagato.
        - `1: Active`: Il premio è stato pagato e la polizza è ora attiva, in attesa che si verifichino le condizioni di rimborso.
        - `2: Claimed`: Il rimborso è stato emesso con successo.
        - `3: Cancelled`: La polizza è stata annullata.
    - **Funzionalità di Pagamento Premio:** Contiene una funzione esposta (`payPremium`) che consente all'assicurato di inviare il `premiumAmount` di `TulToken` al contratto, facendolo passare allo stato `Active`.
    - **Erogazione Rimborso:** Contiene una funzione (`requestPayout`) che, una volta che le condizioni tramite l'Oracle sono state verificate come soddisfatte, trasferisce il `payoutAmount` dalla compagnia all'assicurato.

1.4. TulToken (ERC-20 Token)

Per facilitare tutte le transazioni finanziarie all'interno dell'ecosistema di Trust App, viene utilizzato un token standard ERC-20. Questo token, qui denominato `TulToken`, è la valuta operativa della piattaforma.

- **Funzioni e Caratteristiche:**
    - **Mezzo di Scambio:** Funge da valuta primaria per tutti gli scambi monetari all'interno dell'applicazione.
    - **Conformità ERC-20:** Essendo un token che aderisce allo standard ERC-20, beneficia di tutte le funzionalità e l'interoperabilità di questo standard ampiamente adottato.

2. Architettura Generale dell'Applicazione

Trust App è sviluppata tramite il framework React Native, il che consente di offrire un'esperienza utente su iOS e Android. Questa architettura è cruciale per la sua accessibilità e funzionalità.

2.1. Componenti del Frontend

- **React Native / Expo:** Il frontend è costruito con React Native, che permette lo sviluppo cross-platform da una singola codebase JavaScript/TypeScript. Expo aggiunge un layer di astrazione e strumenti che semplificano notevolmente il ciclo di sviluppo, fornendo API pre-configurate (es. navigazione, clipboard).
- **Connettività Wallet (WalletConnect v2):** L'applicazione si integra con `WalletConnect` per permettere agli utenti di connettere in modo sicuro i propri wallet mobili esistenti (es. MetaMask) e firmare le transazioni.
- **Interazione Blockchain (Ethers.js):** Tutte le chiamate ai contratti smart (lettura di dati, invio di transazioni) vengono effettuate tramite la libreria `Ethers.js`. Questa libreria astrae la complessità dell'interazione con i nodi Ethereum.
- **Stilizzazione (NativeWind):** L'UI è stilizzata usando `NativeWind`, che porta i principi di Tailwind CSS in React Native.

2.2. Gestione dello Stato Globale (React Context API)

Un componente fondamentale dell'architettura frontend è l'uso del `React Context API` per la gestione dello stato globale. In particolare, il `AuthContext` (definito in `AuthContext.tsx`) centralizza la maggior parte dello stato e delle logiche relative all'interazione blockchain:

- **Stato del Wallet:** Gestisce lo stato di connessione del wallet (`walletConnected`), l'indirizzo del wallet (`address`), e le funzioni per connettere/disconnettere (`connectWallet`, `disconnectWallet`).
- **Provider Ethereum:** Contiene l'istanza di `ethers.JsonRpcProvider` che permette all'applicazione di leggere dati dalla blockchain.
- **Signer (per transazioni):** Quando un wallet è connesso, il `AuthContext` fornisce un `Signer` (derivato dal provider e dal wallet connesso) che è essenziale per firmare e inviare transazioni che modificano lo stato on-chain.
- **Funzioni di Interazione Contratto:** Espone funzioni wrapper per le interazioni comuni con i contratti smart (es. `registerWalletOnChain`, `createSmartInsurance`, `paySmartInsurancePremium`, `submitZoniaRequest`, `paySmartInsurancePayout`).

3. Funzionamento Dettagliato dell'Applicazione e Flusso Utente

Andiamo più a fondo nel percorso che un utente o una compagnia segue nell'utilizzo di Trust App.

3.1. Onboarding: Connessione, Registrazione e Selezione Ruolo

1. **Welcome Screen:** L'applicazione si apre con una schermata iniziale che invita l'utente a connettere il proprio wallet Ethereum (Sepolia).
2. **Connessione Wallet (`connectWallet`):** L'utente clicca su "Connect Wallet", attivando il modal di `WalletConnect`. L'utente viene reindirizzato a un'applicazione wallet esterna per approvare la connessione.
3. **Verifica On-Chain (`getWalletTypeOnChain`):** Una volta connesso, l'applicazione invoca la funzione `getWalletTypeOnChain` dal `AuthContext`. Questa funzione interroga il `UserCompanyRegistry` per determinare se l'indirizzo del wallet è già associato a un `IndividualWalletInfo` esistente.
4. **Registrazione (Nuovi Utenti):**
    - Se il wallet non è registrato, l'utente viene reindirizzato a una schermata di registrazione dove può scegliere il suo ruolo (`User` o `Company`).
    - Una transazione viene inviata tramite `registerWalletOnChain` per creare un nuovo contratto `IndividualWalletInfo` e registrarlo nel `UserCompanyRegistry`.
5. **Accesso (Utenti Esistenti):**
    - Se il wallet è già registrato, l'applicazione recupera automaticamente il ruolo e l'indirizzo del `IndividualWalletInfo` associato.
    - L'utente viene quindi reindirizzato alla schermata principale di `BrowseScreen` con le funzionalità appropriate al suo ruolo.

3.2. Ruoli Distinti: Utente vs. Compagnia

L'interfaccia e le funzionalità di Trust App si adattano dinamicamente in base al `selectedAppRole` del wallet connesso:

- **Ruolo "User":**
    - Visualizza le proprie Smart Insurances sottoscritte (sia in attesa di pagamento che attive).
    - Può pagare il premio per le polizze `Pending`.
    - Può richiedere il payout per le polizze `Active` se le condizioni sono soddisfatte (tramite `Check data` e successivo `Request Payout`).
    - Non può creare nuove polizze.
- **Ruolo "Company":**
    - Visualizza tutte le Smart Insurances che ha emesso.
    - Ha la possibilità di creare nuove Smart Insurances tramite una sezione dedicata.

- Non può pagare premi o richiedere payout direttamente sulle polizze come assicurato.

3.3. Creazione di Smart Insurance (Compagnie)

Le compagnie utilizzano una schermata specifica per definire i termini di una nuova polizza:

1. **Input:** La compagnia inserisce manualmente tutti i parametri della `SmartInsurance`: indirizzo del wallet dell'assicurato, importi di premio e payout, tipo di sensore, valore target, coordinate geografiche e raggio.
2. **Validazione Frontend:** Vengono effettuate delle validazioni lato client per assicurare che gli input siano nel formato corretto.
3. **Deploy Contratto (`createSmartInsurance`):** Una volta confermati i dettagli, la compagnia invia una transazione. Questa transazione (firmata dal wallet della compagnia) avvia il deploy di una nuova istanza del contratto `SmartInsurance` sulla blockchain. Il contratto viene automaticamente collegato ai rispettivi `IndividualWalletInfo` di utente assicurato e compagnia. Durante la fase di deploy, la compagnia approva subito che possa essere prelevata dal suo wallet una quantità di `TulToken`, pari al valore di payout della Smart Insurance.

3.4. Visualizzazione e Gestione delle Polizze

La `BrowseScreen` è il dashboard centrale per tutti i tipi di utenti.

1. **Recupero Iniziale:** All'apertura della schermata, viene interrogato il `IndividualWalletInfo` del wallet collegato, per recuperare tutte le insurances associate ad esso.
2. **Recupero Dettagli:** Per ciascun indirizzo, quando si clicca su una polizza specifica, l'applicazione chiama una funzione per recuperare i parametri completi e lo stato corrente della `SmartInsurance` dalla blockchain.
3. **Filtraggio e Visualizzazione:** Le polizze vengono visualizzate in liste, raggruppate per stato (`Pending`, `Active`, `Closed`). Gli utenti possono facilmente navigare tra queste categorie.
4. **Dettagli Modale:** Cliccando su una polizza, una modale si apre, mostrando tutti i dettagli strutturati.

3.5. Ciclo di Vita della Polizza: Dal Pagamento al Payout

3.5.1. Pagamento del Premio (Utente):

1. **Pulsante "Pay Premium":** Se una polizza è `Pending` e l'utente connesso è il `userWallet` della polizza, nella modale dei dettagli appare un pulsante "Pay Premium".
2. **Approvazione Token (se necessaria):** Prima di pagare, l'utente deve approvare che il contratto possa spendere un certo ammontare di `TulToken` dal suo wallet, pari al valore premium della `SmartInsurance`.

3. **Transazione di Pagamento:** Cliccando, l'applicazione invia una transazione on-chain, trasferendo il `premiumAmount` di `TulToken` dall'utente al wallet della compagnia.
4. **Aggiornamento Stato:** Il contratto `SmartInsurance` cambia il suo `currentStatus` da `Pending` ad `Active`.

3.5.2. Richiesta e Controllo Dati (Zonia Oracle System):

1. **Pulsante "Check data":** Quando una polizza è `Active`, nella modale dei dettagli appare il pulsante "Check data".
2. **Inizio Richiesta Zonia:** Cliccando, l'applicazione invia una transazione. Questa funzione interagirà con il contratto `SmartInsurance` per avviare il processo di richiesta dati tramite Zonia Oracle System.
3. **Monitoraggio Progresso:** Viene aperta una modale per la trasparenza:
    - Mostra lo stato corrente della richiesta (gestito nel `AuthContext`), che transita attraverso le fasi: `pending`, `submitted`, `seeded`, `ready`, `completed`, `failed`.
    - Viene aggiornato in tempo reale tramite un listener on-chain sullo stato della richiesta Zonia.
4. **Risultato Zonia:** Una volta che la richiesta Zonia raggiunge lo stato `completed` (o `failed`), la modale mostra il risultato della richiesta (il dato effettivo o il messaggio di errore).

3.5.3. Erogazione del Rimborso (Payout):

1. **Pulsante "Request Payout":** Se la polizza è `Active`, il dato Zonia ha verificato che le condizioni sono soddisfatte, apparirà il pulsante "Request Payout".
2. **Transazione di Payout:** L'utente clicca, inviando una transazione on-chain. Questa transazione innesca il trasferimento del `payoutAmount` in `TulToken` dal wallet della compagnia al wallet dell'utente, come definito nel contratto `SmartInsurance`.
3. **Stato Finale:** Il `currentStatus` della polizza viene aggiornato a `Claimed` sulla blockchain, chiudendo il ciclo della Smart Insurance.

4. Librerie e Tecnologie Utilizzate

La selezione delle librerie e dei framework è stata strategica per bilanciare velocità di sviluppo, sicurezza e facilità di interazione con la blockchain.

4.1. Sviluppo Frontend e Core

- **React Native:** Il fondamento per la creazione di interfacce utente mobili native. Permette l'utilizzo di JavaScript/TypeScript per sviluppare app per iOS e Android da un'unica codebase.
- **Expo:** Un set di strumenti e servizi che estendono React Native. Expo semplifica drasticamente il processo di build, testing e deployment delle app React Native.
- **TypeScript:** Un superset di JavaScript che aggiunge tipizzazione statica. Cruciale per definire interfacce chiare per i dati dei contratti smart (es. `SmartInsuranceDetails`).

4.2. Connettività Blockchain

- **Ethers.js:** La libreria JavaScript per eccellenza per l'interazione con la blockchain Ethereum. Fornisce funzionalità complete per:
    - Creare e gestire provider (connessioni ai nodi Ethereum, es. Infura, Alchemy).
    - Interagire con i contratti smart (creare istanze `Contract` con ABI e indirizzi, chiamare funzioni `read-only` o inviare transazioni).
    - Gestire le firme delle transazioni.
    - Gestire le utility per i tipi di dati Ethereum (BigNumber, conversioni ether/wei).
- **`@walletconnect/modal-react-native`:** Integra il protocollo WalletConnect per una connessione sicura e facile tra l'app mobile e i wallet blockchain esterni. Offre un modal UI pre-configurato per l'esperienza di connessione.

4.3. Gestione dello Stato e Persistenza Dati

- **`useState` / `useEffect` / `useContext` / `useRef` (React Hooks):** Fondamentali per la gestione dello stato locale dei componenti, degli effetti collaterali (fetch dati, side effects asincroni) e dell'accesso ai contesti. `useRef` è utile per mantenere riferimenti mutabili a oggetti (come l'istanza del provider Ethers.js) che non devono causare re-render quando cambiano.
- **`@react-native-async-storage/async-storage`:** Utilizzato per la persistenza di dati leggeri e non sensibili a livello locale sul dispositivo (es. il ruolo utente selezionato). Questo permette all'applicazione di ricordare alcune impostazioni tra le sessioni.

4.4. UI e Stilizzazione

- **NativeWind:** La soluzione per integrare Tailwind CSS in React Native. Consente agli sviluppatori di applicare stili in modo dichiarativo direttamente nel JSX usando classi utility, accelerando lo sviluppo e mantenendo una forte coerenza visiva.

4.5. Navigazione

- **`expo-router`:** Un sistema di routing basato su file per React Native e Expo, ispirato al routing di Next.js. Semplifica la navigazione tra le schermate, la gestione di route dinamiche e l'organizzazione della struttura dell'applicazione in modo intuitivo e scalabile.

5. Dettagli Implementativi Chiave

5.1. Interazione Robusta con i Contratti Smart

- **ABI e Indirizzi Decentralizzati:** Gli ABI (Application Binary Interface) di tutti i contratti smart (`UserCompanyRegistry`, `IndividualWalletInfo`, `SmartInsurance`, `TulToken`, `Gate`) sono precaricati nell'applicazione (da `/constants/abis`). Gli indirizzi dei contratti

deployati sulla Sepolia testnet sono gestiti in una configurazione centralizzata (`chainIdToContractAddresses`).
- **Istanze di Contratto Dinamiche:** Nel `AuthContext`, le funzioni come `getUserCompanyRegistryContract()` o `getSmartInsuranceContract(address)` creano istanze di `ethers.Contract`. A seconda dell'operazione, queste istanze vengono create con un `provider` (per chiamate di sola lettura, che non modificano lo stato) o con un `signer` (per le transazioni che modificano lo stato, che richiedono la firma del wallet).
- **Gestione BigNumber e Unità:** Le quantità di token (premi, payout) vengono spesso gestite come `BigNumber` da Ethers.js per prevenire problemi di precisione con JavaScript numeri. Le conversioni tra unità umane (es. 100 `TulToken`) e unità native del token (wei, gwei, etc.) sono gestite internamente utilizzando `ethers.utils.parseUnits` e `formatUnits`.

5.2. Gestione degli Errori

La gestione robusta degli errori è fondamentale in una dApp per guidare l'utente attraverso potenziali problemi:

- **Blocchi Try-Catch:** Tutte le chiamate ai contratti smart e le operazioni asincrone critiche sono avvolte in blocchi `try-catch`. Questo cattura errori di rete, errori di transazione (es. `revert` dal contratto), gas insufficiente o rifiuti di firma da parte dell'utente.
- **Messaggi di Errore Significativi:** In caso di errore, l'applicazione tenta di estrarre messaggi d'errore leggibili e li presenta all'utente.

5.3. Sincronizzazione Stato On-chain/Off-chain

Mantenere l'interfaccia utente sincronizzata con lo stato immutabile della blockchain è una sfida costante.

- **Ascolto Eventi:** Un'ottimizzazione cruciale riguarda l'implementazione dell'ascolto di eventi on-chain. I contratti smart possono emettere "eventi" quando determinate azioni avvengono. L'applicazione si sottoscrive a questi eventi e, una volta rilevati, aggiorna l'UI in tempo reale, eliminando la necessità di polling costante per molte operazioni e rendendo l'esperienza utente più reattiva.

5.4. Gestione di Ruoli e Permessi D'accesso

- **Logica Frontend:** Il `selectedAppRole` (ottenuto durante la registrazione/accesso) viene utilizzato estensivamente per il rendering condizionale dell'UI. Ad esempio, il pulsante "Create Smart Insurance" è visibile solo se `selectedAppRole` è "company".
- **Controlli On-Chain:** È fondamentale che i contratti smart stessi contengano controlli di accesso (`require(msg.sender == owner)`, `require(userWallet == _user)`, etc.) per garantire che solo i wallet autorizzati possano eseguire determinate funzioni, indipendentemente da eventuali modifiche o bypass nel frontend.

5.5. Dettagli dell'Integrazione Zonia Oracle System

L'integrazione Zonia è il cuore dell'automazione delle polizze condizionate da dati esterni.

- **Input della Query:** Le `SmartInsurance` memorizzano una `query` stringa specifica e i dettagli (`sensor`, `geoloc`, `target_value`) che vengono passati al `Gate` contract quando una richiesta di dati viene avviata.
- **Tracciamento Granulare:** Il `zoniaStates` array (`"pending"`, `"submitted"`, `"seeded"`, `"ready"`, `"completed"`, `"failed"`) fornisce una mappa chiara del progresso della richiesta all'Oracle.
  Il `AuthContext` aggiorna `zoniaRequestState` in base alle risposte del `Gate` contract, permettendo all'utente di visualizzare l'attuale fase del processo.
- **Result Forwarding:** Una volta che Zonia ha aggregato e validato i dati, questo risultato viene reso disponibile alla `SmartInsurance` pertinente.
- **Logica di Retry/Fallimento:** In caso di `failed` zoniaRequestState, l'applicazione informa l'utente e permette, di riprovare la richiesta.

Conclusioni

Sfruttando la robustezza e la trasparenza della blockchain Ethereum, la piattaforma offre un modello innovativo. L'integrazione con Zonia Oracle System è la chiave per colmare il divario tra il mondo digitale on-chain e i dati del mondo reale, rendendo possibili le Smart Insurances basate su condizioni concrete e verificabili.