**INTERNET OF THINGS BASED SMART SYSTEMS**

Ingegneria Informatica LM-32

A.A. 2022/2023

**ETH-TOKENFIDELITY: Creazione di un token fungibile ERC 20 associato ad un caso d’uso**

Studenti:

Piero Galatà 1000066734

Francesco Denaro 1000050594

# Introduzione

Lo scopo dell’elaborato è quello di creare uno smart contract ERC-20 su rete **Ethereum** basato su testnet con l’obiettivo di mostrare una serie di semplici transazioni registrate all’interno di una blockchain.

Una **blockchain** è una tecnologia che può essere definita come un registro digitale in cui le informazioni vengono salvate in modo sicuro attraverso blocchi collegati in ordine cronologico. Per questo la blockchain viene definita come una “catena di blocchi”, ognuno dei quali contiene informazioni, come transazioni, ed è collegato agli altri in una struttura che impedisce la manipolazione delle informazioni una volta che sono state aggiunte alla catena. La blockchain si fonda su aspetti fondamentali quali, decentralizzazione, permission-less, immutabilità, servizi aperti e pubblici e trustless.

**Ethereum** è una piattaforma blockchain decentralizzata, progettata per eseguire smart contract: applicazioni che funzionano esattamente come programmate, senza possibilità di frodi, censura o interferenze di terze parti. In Ethereum, il token nativo, **Ether (ETH)**, non è solo una valuta, ma anche "carburante" per eseguire il codice su questa piattaforma.

Le merci e le risorse su una Blockchain sono identificate tramite indirizzi (addresses): stringhe di caratteri associate a una quantità di criptovaluta (token).

Tramite l’uso degli **smart contracts** si ha la possibilità di aggiungere della logica alla blockchain. Uno smart contract, come qualsiasi altro contratto, regola i termini e le condizioni tra le parti. A differenza di quanto avviene per un contratto tradizionale, i termini di uno smart contract vengono eseguiti sulla base di un codice programmato su una blockchain come Ethereum. Gli smart contract consentono agli sviluppatori di creare app che sfruttano la sicurezza, l'affidabilità e l'accessibilità della blockchain, offrendo sofisticate funzionalità peer-to-peer che comprendono i servizi di richiesta e concessione prestiti e assicurativi, la logistica e i giochi. Le figure che interagiscono in uno smart contract sono:

* **Owner/landlord**: crea il contratto e lo replica nei vari nodi. Può anche controllare il balance e se lo stato del contratto è stato fetchato da un nodo;

Immagine che contiene diagramma, testo, linea, Carattere

Descrizione generata automaticamente

* **Tenant**: deposita il contratto e ne modifica lo stato nei vari nodi

Immagine che contiene diagramma, testo, Carattere, schermata

Descrizione generata automaticamente

# Sviluppo dello smart contract

Il contratto presente nell’elaborato è progettato per consentire la creazione di nuovi coin (**mint**) e l'invio di tali coin a indirizzi appartenenti alla stessa blockchain (**send**). Inoltre, permette anche di mostrare il saldo di un determinato account (**showBalances**). Per creazione di nuovi coin si intende la valuta che verrà utilizzata all’interno del contratto stesso, non ETH.

| pragma solidity >= 0.7.1 < 0.9.0; contract FidelityCoin {  address public minter;  mapping (address => uint) public balances;   event Sent(address from, address to, uint amount);   constructor() {     minter = msg.sender;  }    function mint(address receiver, uint amount) public returns (uint){     require(msg.sender == minter);     require(amount < 1e60);     balances[receiver] += amount;     return balances[receiver];  }   function send(address receiver, uint amount) public {     require(amount <= balances[msg.sender], "Insufficient balance detected.");     balances[msg.sender] -= amount;     balances[receiver] += amount;     emit Sent(msg.sender, receiver, amount);  }   function showBalances(address account) external view returns (uint){     return balances[account];  } } |
| --- |

Il contratto crea un token **FidelityCoin** inizializzando un miner (con il costruttore pubblico), ovvero un indirizzo che può creare nuovi coin. Solo il creatore del contratto può coniare nuovi coin.

Questo contratto modella anche il mapping di ogni indirizzo con il suo saldo, e la funzione send controlla semplicemente la fattibilità della transazione. Se il mittente ha meno saldo di quello che desidera inviare, il contratto restituirà un messaggio di saldo insufficiente e la transazione non verrà completata. Se la transazione è invece fattibile, la funzione emetterà un evento di invio. I client Ethereum (wallets o app decentralizzate sul web) possono ascoltare questi eventi emessi sulla blockchain. L’ascolto dei contratti e delle transazioni all’interno di una blockchain è gratuito, mentre si paga la creazione delle transazioni e le modifiche dei contratti già esistenti all’interno della chain. Non appena l'evento viene emesso, l'ascoltatore riceve gli argomenti from, to e amount, il che rende possibile tenere traccia delle transazioni.

Per testare il corretto funzionamento del contratto, si è fatto uso di Remix IDE, un tool open source per scrivere, compilare e testare contratti in Solidity. Dopo aver scritto il contratto, sono stati effettuati diversi deploy utilizzando la tab “Deploy & run transactions”:

* Per il primo deploy è stato impostato come **Environment** “Remiv VM (London)”, impostato un valore di **Gas limit** a 6721975 e **Value** a 0, poiché la funzione chiamata deve essere payable per inserire un value diverso da quello indicato. Il gas si riferisce alla fee richiesta per condurre con successo una transazione o eseguire un contratto sulla piattaforma blockchain di Ethereum, poiché ogni transazione consuma risorse di calcolo;
* Per il secondo deploy è stato effettuato il collegamento a Metamask, dopo aver creato un account e ottenuto i fondi necessari da un faucet nella testnet Sepolia. In particolare, i fondi di test sono stati richiesti dal faucet “<https://sepolia-faucet.pk910.de/>”

Immagine che contiene testo, schermata

Descrizione generata automaticamente

# Creazione dell’interfaccia web

Al fine di simulare le funzioni contenute nel contratto, è stata creata un’interfaccia web facendo uso di HTML e jQuery. Il risultato ottenuto è il seguente:

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

Tale pagina consente di:

* Richiamare la funzione di **mint** premendo sul bottone “Mintare Token”, inserendo l’indirizzo del ricevitore e l’amount negli appositi input fields;
* Richiamare la funzione di **showBalances** premendo sul secondo bottone “Ottieni Saldo”, inserendo l’indirizzo da controllare nell’input field corrispondente.
* Richiamare la funzione di **send** premendo sul bottone “Trasferisci Token”, inserendo l’indirizzo a cui voler inviare i fondi e l’importo da voler inviare

Al click sui bottoni, verranno eseguite delle chiamate ajax alle API esposte dal server scritto in Express.

# Server Express (Node.js)

Il server è stato implementato utilizzando **Express**, un framework per **Node.js**, seguendo il principio di **Singola Responsabilità**. Questo principio, tipicamente utilizzato nella programmazione orientata agli oggetti, può essere visto anche come un principio architetturale, analogo alla **separazione dei concetti**. In particolare, il principio afferma che ogni entità (come una classe o un modulo) dovrebbe avere **una sola responsabilità** e quindi **un solo motivo per cambiare**. L'unico contesto in cui un'entità può subire modifiche è quando vi è la necessità di aggiornare il modo in cui adempie alla sua specifica responsabilità.

L'adozione di questo principio promuove la modularità del sistema, permettendo l'aggiunta di nuove funzionalità attraverso l'estensione di nuovi moduli o classi, anziché aumentare la complessità di quelli già esistenti. Questo approccio facilita inoltre l'aggiornamento e la gestione di nuove versioni delle API esposte

### Struttura del Codice e Servizi

Il codice più rilevante è contenuto all'interno del file **services/blockchain.js**, dove sono definite alcune delle funzioni principali che interagiscono con lo smart contract sulla blockchain.

**Funzione mintTokens**

La funzione asincrona mintTokens ha il compito di generare nuovi token sul contratto smart. Una volta effettuato il **deploy** del contratto, viene invocata la funzione **mint**, e successivamente la funzione **send** che consente di modificare lo stato del contratto. Poiché ogni modifica del contratto richiede il pagamento di una **commissione in gas**, è necessario specificare anche il valore del gas nelle transazioni.

**Funzione getBalance**

La funzione asincrona getBalance consente di recuperare il bilancio associato a un indirizzo specificato nel corpo della richiesta HTTP. In questo caso, viene utilizzata la funzione **showBalances** e successivamente la funzione **call** sul contratto, che permette di interrogare lo stato del contratto senza alterarlo. Ciò significa che non vi è alcuna modifica allo smart contract e, pertanto, nessuna fee in gas viene richiesta.

**Funzione transferTokens**

La funzione asincrona transferTokens è utilizzata per eseguire una transazione di token dal mittente a un destinatario sulla blockchain. In questo caso viene richiamata la funzione **transfer** sul contratto, per trasferire i token e successivamente viene utilizzato il metodo **encodeABI** per codificare il codice in un formato eseguibile alla Virtual Machine Ethereum. Inoltre, la transazione viene firmata localmente utilizzando la chiave privata del mittente. Questo passaggio è necessario per autenticare la transazione prima di inviarla alla blockchain. Il metodo signTransaction firma la transazione con la chiave privata e genera una transazione firmata (signedTx). Infine, la transazione firmata viene inviata alla rete tramite **sendSignedTransaction**. Questo invio viene gestito in modo asincrono, e restituisce la ricevuta della transazione una volta confermata sulla blockchain.

È importante sottolineare che durante questo processo non avviene alcun trasferimento di criptovalute. Il valore restituito rappresenta l'informazione relativa al bilancio, modificabile esclusivamente dall'**owner** del contratto.

Il server include un sistema di **debug** che permette di monitorare e leggere informazioni aggiuntive relative all'elaborazione delle richieste tramite il terminale.

### Simulazione Locale

Per eseguire il progetto in ambiente di sviluppo locale, è necessaria l'installazione di **Ganache**, uno strumento che fornisce una blockchain locale di test con una serie di indirizzi preconfigurati. Ganache consente di simulare interazioni con il contratto, senza dover interagire con una blockchain pubblica.

# Descrizione caso d’uso

Il progetto prevede la creazione di una piattaforma, dal nome “Eth-TokenFidelity”, che consente al proprietario di un e-commerce di utilizzare un token ERC-20 per incentivare i clienti attraverso un programma di fidelizzazione. I clienti ricevono i token, ossia i “FidelityCoin”, come ricompensa per ogni acquisto effettuato, che potranno utilizzare per ottenere sconti o premi speciali.

L’obiettivo è fornire al proprietario di un e-commerce uno strumento per gestire facilmente i FidelityCoin tramite una dashboard, la quale consente di generare nuovi coin al proprietario e trasferirli agli account dei clienti in base agli ordini effettuati. Ad esempio, per ogni acquisto completato, in base all’importo speso, una quantità predefinita di FidelityCoin verrà inviata all’account del cliente, incrementando il suo saldo. I coin rappresentano punti fedeltà che possono essere usati successivamente per ottenere sconti o premi.  
Inoltre, la dashboard offre una vista aggiornata del saldo di fidelityCoin per ciascun account. Il proprietario può monitorare il proprio saldo e quello di tutti gli account clienti, mentre i clienti possono verificare quanti token hanno accumulato, aumentando la trasparenza e l’engagement.

Ovviamente è possibile creare diverse logiche per automatizzare la creazione di nuovi token sull’account proprietario, tramite job schedulati, evitando l’intervento manuale.

Tramite l’utilizzo di questa dashboard, l’e-commerce trae vantaggi come:

* **Fidelizzazione del cliente**: offrendo FidelityCoin come ricompensa, l’e-commerce aumenta la fidelizzazione dei clienti, incentivando questi ultimi a fare acquisti per guadagnare ulteriori vantaggi.
* **Automazione**: la possibilità di mintare token automaticamente riduce il carico gestionale del proprietario, che può concentrarsi su altre aree strategiche dell’attività.
* **Sicurezza e trasparenza**: grazie alla tecnologia blockchain e agli smart contract, tutte le transazioni sono immutabili e verificabili, garantendo così massima fiducia da parte dei clienti.

# Sviluppo e branch

Per lo sviluppo è stato scelto di creare due branch diversi, in modo tale da avere due soluzioni diverse: una esemplificativa (branch **test-develop**) e una finale, utilizzabile dal proprietario dell’e-commerce (branch **master**).

Il branch “**test-develop**” è stato creato per scopi di testing, in quanto fornisce allo sviluppatore la flessibilità necessaria per verificare il corretto funzionamento delle funzionalità del contratto. In particolare, in questo branch, l’interfaccia utente permette l’inserimento manuale dell’indirizzo e della private key dell’account mittente, nella funzione utile al trasferimento dei token, consentendo di eseguire test approfonditi e verificare che la funzione **transfer** fallisca nel caso in cui non ci sia corrispondenza tra chiave privata e indirizzo account e invece abbia successo se c’è corrispondenza.

Il branch “**master**” rappresenta la versione finale del progetto, pronta per essere utilizzata dal proprietario dell’e-commerce. In questa versione, l’interfaccia utente non consente l’inserimento manuale dell’indirizzo e della chiave private del mittente, poiché saranno esclusivamente quelli del proprietario dell’e-commerce e verranno recuperati in modo sicuro da un file di configurazione protetto. Questo garantisce che solo il proprietario abbia il controllo sui fondi e che la sicurezza dell’account non sia compromessa. Inoltre, tutto ciò evita errori umani e aumenta la sicurezza, poiché l’inserimento di informazioni sensibili è gestito in modo sicuro e automatico, riducendo il rischio di esporre dati critici come chiavi protette.