Milk Tracker Project Report

Michele Stelluti – 1038862 – michele.stelluti@studio.unibo.it
Marco Benito Tomasone – 1038815 – marcobenito.tomasone@studio.unibo.it

Corso di Blockchain and Cryptocurrencies - Dipartimento di Informatica
Università di Bologna

Abstract. Questo documento presenta un progetto per la realizzazione di una Dapp basata su tecnologia blockchain per il tracciamento completo dei prodotti caseari, tra cui mozzarelle, formaggi e altro. La Dapp sfrutterà gli smart contract implementati in Solidity e basati su Ethereum, integrati con Truffle, Ganache, React.js, Web3.js e Metamask.
Il nostro obiettivo è creare una piattaforma che consenta a qualsiasi caseificio di gestire il tracciamento dei propri prodotti caseari in modo trasparente e affidabile. I caseifici avranno la possibilità di registrarsi sulla Dapp e inserire informazioni riguardanti le mucche, i lotti di latte e i prodotti caseari prodotti. Inoltre, la piattaforma permetterà ai caseifici di effettuare operazioni, come l'eliminazione di mucche o il trasferimento di mucche e lotti di latte ad altri caseifici.
Un aspetto fondamentale della nostra Dapp è la sicurezza e la tracciabilità dei prodotti per gli utenti finali. Ogni utente potrà verificare l'origine e la qualità dei prodotti caseari acquistati e potranno segnalare eventuali problemi o avarie riscontrate nei prodotti, contribuendo così a migliorare la qualità dell'intera catena produttiva.

1 Introduzione

L'acronimo DApp sta per Decentralized Applications, ovvero applicazioni decentralizzate. Si tratta di un tipo specifico di applicazioni che per funzionare non hanno bisogno di server centrali, ma di una rete decentralizzata.
Tale rete prevede che i suoi utenti ne abbiano il pieno controllo e consente di accedere a vari servizi in maniera sicura. Per le app tradizionali i dati vengono archiviati su server centralizzati e di fatto si permette alle società proprietarie di poter operare azioni di controllo, censura e manipolazione.
Tutto questo, invece, non può accadere con le applicazioni decentralizzate perché girano su network peer-to-peer oppure su blockchain.

La nostra applicazione utilizza Smart Contracts implementati in Solidity e basati su Ethereum.
Gli smart contract, o "contratti intelligenti", sono programmi informatici autoeseguibili che operano su una blockchain. Essi permettono di automatizzare, verificare e far rispettare l'esecuzione di accordi e transazioni tra parti senza bisogno di intermediari. Grazie alla loro natura decentralizzata e alla sicurezza

garantita dalla tecnologia blockchain, gli smart contract offrono trasparenza, affidabilità e immutabilità delle operazioni, rappresentando un pilastro fondamentale per l'implementazione di diverse applicazioni decentralizzate (Dapp) in svariati settori.

L'applicazione fa uso di Truffle, Ganache, React.js, Web3.js e Metamask che verranno spiegati nei capitoli successivi.

1.1 Funzionalità del progetto

Il presente progetto implementa diverse funzionalità che si suddividono tra il lato del caseificio e il lato dell'utente.

Funzionalità per il caseificio:

1. Registrazione di un nuovo caseificio;
2. Registrazione di una nuova mucca;
3. Registrazione di un nuovo lotto di latte;
4. Registrazione di un nuovo prodotto caseario;
5. Eliminazione di una mucca;
6. Trasferimento di una mucca a un altro caseificio;
7. Trasferimento di un lotto di latte a un altro caseificio.

Funzionalità per il cliente:

1. Ricerca di un prodotto caseario tramite il suo codice identificativo;
2. Ricezione delle informazioni riguardanti il prodotto caseario, incluso il lotto di latte di origine e le informazioni sulla mucca produttrice;
3. Possibilità di segnalare un prodotto avariato o difettoso;
4. Ricezione di avvisi riguardo a prodotti provenienti da lotti di latte segnalati come avariati.

La realizzazione delle precedenti funzionalità verranno descritte nel dettaglio nel capitolo 3.

2 Tecnologie utilizzate

In questo capitolo vengono riportate le tecnologie necessarie per il corretto funzionamento della Dapp.

Solidity

Solidity è un linguaggio di programmazione ad alto livello utilizzato per sviluppare smart contract sulla piattaforma Ethereum. Creato per essere specificamente utilizzato in ambienti blockchain, Solidity è progettato per consentire la scrittura di contratti intelligenti che automatizzano e gestiscono transazioni e accordi tra diverse parti in modo sicuro e affidabile.
Questo linguaggio di programmazione orientato agli oggetti è ispirato a C++, JavaScript e Python e fornisce una sintassi chiara e intuitiva. Gli smart contract scritti in Solidity vengono eseguiti all'interno della macchina virtuale di Ethereum (EVM), garantendo la sicurezza e l'immutabilità delle operazioni grazie alla tecnologia blockchain.

Ethereum

Ethereum è una piattaforma open source di blockchain che consente agli sviluppatori di creare e distribuire applicazioni decentralizzate (Dapp) e smart contract.
Ciò che rende Ethereum unico è il suo supporto per i contratti intelligenti (smart contract), che sono programmi informatici autoeseguibili. Gli smart contract consentono agli sviluppatori di creare e gestire accordi digitali in modo sicuro e automatico, senza bisogno di intermediari.
Invece di focalizzarsi solo sulla moneta digitale (ETH), Ethereum mira a fornire un ecosistema completo per applicazioni decentralizzate. Ciò è reso possibile grazie alla sua macchina virtuale denominata Ethereum Virtual Machine (EVM), che esegue gli smart contract in modo sicuro e affidabile su tutti i nodi della rete.

Web3.js

Web3.js è una libreria JavaScript che consente agli sviluppatori di interagire con la blockchain di Ethereum e le applicazioni decentralizzate (Dapp). Con Web3.js, è possibile creare Dapp, gestire transazioni e interagire con i contratti intelligenti sulla rete Ethereum direttamente dal browser web. Essa è essenziale per lo sviluppo di applicazioni decentralizzate complesse e ha contribuito all'espansione dell'ecosistema di Ethereum.

React.js

React.js è una libreria JavaScript open-source utilizzata per la creazione di interfacce utente interattive e reattive.
La caratteristica distintiva di React.js è il suo approccio basato su componenti, dove ogni parte dell'interfaccia utente è rappresentata come un componente autonomo e isolato. Questi componenti possono essere facilmente combinati per formare interfacce complesse e reattive.

Metamask

MetaMask è un'estensione per browser e un portafoglio digitale che consente agli utenti di interagire con la blockchain di Ethereum e le applicazioni decentralizzate (Dapp) direttamente dal proprio browser web. Funziona come un ponte tra il browser e la blockchain, consentendo agli utenti di gestire i loro account Ethereum, effettuare transazioni e interagire con i contratti intelligenti senza dover lasciare l'interfaccia web.

Truffle

Truffle è un framework di sviluppo per Ethereum che offre un ambiente integrato per scrivere, testare e distribuire contratti intelligenti. Truffle semplifica il processo di sviluppo, permettendo di scrivere e compilare contratti intelligenti, gestire le migrazioni dei contratti sulla blockchain e creare test automatizzati per verificare il corretto funzionamento dei contratti.

Ganache

Ganache è un ambiente di sviluppo locale e personale per Ethereum, progettato per semplificare il processo di sviluppo e test di applicazioni decentralizzate (Dapp) basate sulla blockchain di Ethereum.
Con Ganache, gli sviluppatori possono eseguire una blockchain Ethereum privata e isolata direttamente sul proprio computer. Questo ambiente locale offre vantaggi come la velocità di esecuzione e la possibilità di simulare transazioni e interazioni con i contratti intelligenti senza connettersi alla rete Ethereum principale

Table 1: Lista delle tecnologie utilizzate

Nome	Versione
Web3.js	1.10.0
React.js	18.2.0
Metamask	10.33.1
Ganache	7.7.7
Truffle	5.8.2
Solidity	0.8.17

3 Implementazione

Nello sviluppo del progetto, si è fatto riferimento allo schema presente in Fig.1.

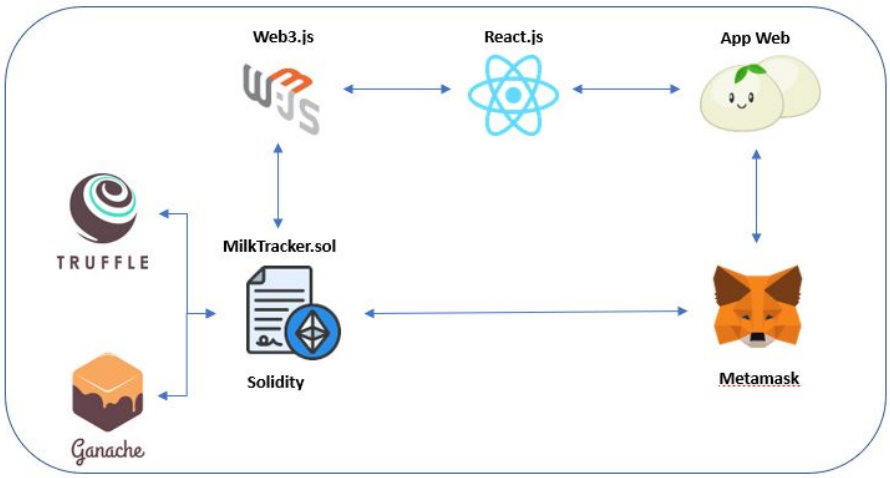

Fig. 1: Schema di comunicazione delle componenti principali del progetto

In questo capitolo verrà analizzato più nel dettaglio lo Smart Contract, gli elementi e le funzioni che lo compongono e le connessioni con i vari componenti principali del progetto come in Fig.1.

3.1 Smart Contract MilkTracker.sol

Lo Smart-Contract rappresenta nel progetto l'elemento più importante dal momento che permette la generazione e la gestione delle informazioni relativi ad ogni mucca, lotto di latte e prodotto caseario.

Per memorizzare tutte le informazioni si sono create le seguenti `struct` all'interno del contratto.

```
struct Dairy {
        address dairyAddress;
        string dairyName;
        string dairyPlace;
}
```

```
struct Cow {
        uint256 id;
        uint16 weight;
        string breed;
        string birthDate;
        string residence;
}
```

```
struct Milk {
        uint256 id;
        uint256 cowId;
        string dateOfProduction;
}
```

```
struct Product {
        uint256 id;
        uint256 milkId;
        string dateOfProduction;
        string productType;
        string expiryDate;
}
```

Vengono usate le seguenti `liste` per memorizzare ogni singolo elemento.

```
Dairy[] public dairyList;
Cow[] public cowList;
Milk[] public milkList;
Product[] public productList;
```

Altri parametri utili al corretto funzionamento dello Smart Contract sono i seguenti `mapping`.

```
//Dato l'ID di una mucca restituisce l'indirizzo del proprietario
mapping(uint256 => address) public cowToOwner;
//Dato l'ID del latte restituisce l'indirizzo del proprietario
mapping(uint256 => address) public milkToOwner;
//Dato l'indirizzo di un proprietario restituisce la lista delle mucche
mapping(address => Cow[]) public ownerToCowList;
//Dato l'indirizzo di un proprietario restituisce la lista dei latti
mapping(address => Milk[]) public ownerToMilk;
//Dato l'ID di un latte restituisce se il latte è avariato
mapping(uint256 => bool) public spoiledMilks;
//Dato l'ID di una mucca restituisce se la mucca è morta
mapping(uint256 => bool) public deathCows;
```

Funzioni Smart Contract

Di seguito sono riportate le funzioni del contratto “MilkTracker.sol”.

Funzione per memorizzare un caseificio nella blockchain a partire dall’indirizzo dell’utente.

```
function addDairy(string memory _dairyName,
                  string memory _dairyPlace) public {...}
```

Funzione per controllare se un caseificio è già stato registrato, ovvero è già presente all'interno della Blockchain.

```
function checkDairyIsRegistered() public view returns (bool) {...}
```

Queste due funzioni vengono utilizzate in fase di apertura della sezione `Dairy` nella pagina `Home`. Se il caseificio è già stato registrato, allora potrà accedere nella sezione di registrazione dei vari assets (mucche, lotti di latti e prodotti caseari), altrimenti verrà riportato nella sezione di registrazione per il caseificio, utilizzando la funzione `addDairy` precedente.

Funzione per restituire la lista di tutti i caseifici registrati all'interno della Blockchain escluso se stesso. In questo modo il caseificio potrà ottenere la lista di tutti i caseifici per poter trasferire uno o più assets.

```
function getAllDairyExceptSelf() public view
                                returns (Dairy[] memory) {...}
```

Funzione per trasferire una mucca da un caseificio ad un altro dato l'indirizzo di destinazione e l'ID della mucca.

```
function transferCow(uint256 _cowId,
                     address _newOwner)
                     public onlyOwnerOf(_cowId)
{...}
```

Funzione per trasferire un lotto di latte da un caseificio ad un altro dato l'indirizzo di destinazione e l'ID del latte.

```
function transferMilk(uint256 _milkId, address _newOwner) public
onlyOwnerOfMilk(_milkId) {...}
```

Funzione per segnalare la more di una mucca dato il suo ID.

```
function killCow(uint256 _cowId) public onlyOwnerOf(_cowId) {...}
```

Queste funzioni vengono utilizzate nella sezione `Handle Assets` presente nella `Navbar`.

Le seguenti tre funzioni vengono utilizzate nella sezione `Dairy`, presente nella `Homepage`, solo dopo che il caseificio sia stato registrato. Le funzioni permetto di inserire le informazioni riguardanti le mucche, i lotti di latte e i prodotti caseari.

```
function addCow(uint16 _weight,
                string memory _breed,
                string memory _birthDate,
                string memory _residence) public {...}

function addMilk(uint256 _cowId,
                string memory _dateOfProduction)
                public onlyOwnerOf(_cowId) {...}

function addProduct(uint256 _milkId,
                    string memory _dateOfProduction,
                    string memory _productType,
                    string memory _expiryDate)
                    public  onlyOwnerOf(milkToCow[_milkId]) {...}
```

Funzioni per restituire tutte le mucche, lotti di latti o prodotti registrati nella Blockchain.

```
function getAllCows() public view returns(Cow[] memory) {...}
function getAllMilks() public view returns(Milk[] memory) {...}
function getAllProducts() public view returns(Product[] memory) {...}
```

Funzioni per restituire tutte le mucche o lotti di latte di un proprietario.

```
function getCowsOfOwner() public view returns(Cow[] memory) {...}
function getMilksOfOwner() public view returns (Milk[] memory) {...}
```

Funzione per restituire solo le mucche ancora vive di un proprietario.

```
 function getAliveCowsOfOwner() public view returns(Cow[] memory)
 {...}
```

Funzioni per restituire una mucca, un lotto di latte o un prodotto dato il loro ID.

```
function getProduct(uint256 _productId) public view returns(Product memory) {...}
function getMilk(uint256 _milkId) public view returns(Milk memory) {...}
function getCow(uint256 _cowId) public view returns(Cow memory) {...}
```

Funzione per controllare se un lotto di latte è avariato dato il suo ID.

```
1 function isMilkSpoiled(uint256 _milkId) public view returns(bool)
2 {...}
```

Funzione per segnalare un prodotto avariato dato il suo ID. La funzione imposta come avariato il latte di origine di quel prodotto.

```
1 function reportSpoiledProduct(uint256 _productId) public {...}
```

Le ultime due funzioni sono utilizzate nella sezione `View` della `Navbar`.

3.2 App Web

Web3.js

```
1  const web3 = new Web3( Web3.givenProvider || "http://127.0.0.1:7545");
2
3  function getWeb3Context() {
4      return web3;
5  }
6
7  async function getAccounts() {
8      return await getWeb3Context().eth.requestAccounts();
9  }
10
11 async function addDairyToContract(account, data) {
12     const MilkTracker = new web3.eth.Contract(getContractABI(),
13         getContractAddress());
14     await MilkTracker.methods.addDairy(data.dairyName, data.dairyPlace)
15             .send({ from: account, gas:3000000 });
16 }
```

Viene mostrato solo un estratto delle funzioni utilizzate per l'interazione con il contratto tramite la libreria `Web3.js`. Nella `riga 1`, è presente la definizione del contesto `Web3`, il quale è utilizzato sia per ricevere tutti gli account collegati che per chiamare le funzioni all'interno del contratto. Per ogni funzione legata al contratto, nella `riga 12` del nostro esempio, è presente la definizione di una variabile rappresentante il contratto, che fa uso dell'Abi e dell'indirizzo del contratto.

```
useEffect(() => {
        getAccounts().then((accounts) => {
            setMyAccount(accounts[0]);
        });
    }, []); // empty dependency array to run only once

    useEffect(() => {
    // In base al componente, codice diverso
    }, [myAccount]);

    window.ethereum.on('accountsChanged', function (accounts) {
        setMyAccount(accounts[0]);
    });
```

Il seguente codice viene utilizzato per gestire gli account all'interno dei vari componenti dell'applicazione blockchain. Esso permette di riconoscere i cambiamenti degli account e di aggiornare dinamicamente il comportamento dell'applicazione in risposta a tali cambiamenti. Ad esempio, nel componente `InsertMilk`, ogni volta che avviene un cambiamento nell'account, la lista di mucche possedute da quell'account viene automaticamente aggiornata per riflettere la modifica.

Questo approccio consente di mantenere un'interfaccia utente reattiva e coerente, adattandosi in tempo reale alle azioni dell'utente e alle modifiche negli account. Inoltre, garantisce che le informazioni visualizzate siano sempre aggiornate e corrette, migliorando l'esperienza complessiva dell'utente nell'applicazione blockchain.

React.js

In questa sezione verranno mostrati tutti i componenti `React` utilizzati per il corretto funzionamento della DApp.

AddressSendDialog.js Componente Dialog React per selezionare dalla lista dei casefici registrati nella blockchain, il caseficio a cui trasferire una mucca o un lotto di latte.

FormDialog.js Componente Dialog React per inserire il codice identificativo di un prodotto caseario acquistato da un utente e ricevere le informazioni sul prodotto, lotto di latte di provenienza e mucca produttrice.

LoginView.js Componente React che controlla se un caseificio è già stato registrato nella blockchain. Se il caseificio non è presente, viene chiamato il componente React `InsertDairy` altrimenti viene chiamato il componente `InsertView`.

InsertView.js Componente React che rappresenta l'homepage per i caseifici registrati nella blockchain. Da qui è possibile far riferimento ai componenti React `InsertMilk`, `InsertCow` e `InsertProduct`.

InsertDairy.js Componente React per registrare un nuovo caseificio, nella blockchain.

InsertCow.js Componente React per registrare una mucca, nella blockchain, da parte di un caseificio.

InsertMilk.js Componente React per registrare un lotto di latte, nella blockchain, da parte di un caseificio.

InsertProduct.js Componente React per registrare un prodotto caseario, nella blockchain, da parte di un caseificio.

HandleAssets Componente React che permette ad un caseificio registrato di segnalare una mucca come morta o di trasferire un proprio lotto di latte o una mucca ad un altro caseificio registrato nella blockchain.

Navbar.js Componente React che rappresenta una `Navbar` che contiene il titolo e il riferimento ai componenti React: `InsertView`, `ViewView` e `HandleAssets`.

SplashScreen.js Componente React che contiene il logo della DApp, utilizzato all'apertura dell'app web.

ViewView.js Componente React che permette all'utente di leggere tutte le informazioni sul prodotto caseario cercato, solo se presente nella Blockchain.

4 Analisi dei costi

Nella rete Ethereum, il concetto di "Gas" è fondamentale per comprendere il funzionamento delle transazioni. Il termine "Gas" si riferisce all'unità di misura utilizzata per quantificare l'effort computazionale necessario per eseguire operazioni specifiche sulla rete.

In altre parole, ogni volta che viene effettuata una transazione sulla rete Ethereum, è richiesta una quantità di Gas per completarla. Questo avviene perché le transazioni vengono elaborate dai nodi della rete, i quali necessitano di risorse computazionali per eseguire i calcoli necessari.

Ogni tipo di operazione all'interno della rete Ethereum ha un costo di Gas associato. Ad esempio, l'invio di Ether da un indirizzo a un altro, l'esecuzione di un contratto intelligente o l'aggiornamento di dati sulla blockchain richiedono ciascuna una quantità specifica di Gas.

Il Gas ha anche un costo in Ether (la criptovaluta di Ethereum). Quindi, per eseguire una transazione, è necessario pagare una certa quantità di Ether proporzionale alla complessità dell'operazione e alla quantità di Gas richiesta.

Questo meccanismo di "Gas" serve a proteggere la rete Ethereum da abusi e attacchi di tipo DoS (Denial of Service), in quanto rende oneroso l'uso eccessivo delle risorse computazionali. Inoltre, incoraggia gli sviluppatori a scrivere codice efficiente e ottimizzato per ridurre i costi di esecuzione delle transazioni.

Nel contratto sviluppato, vi è la presenza di diverse funzioni che variano sugli effetti. Alcune funzioni necessitano della generazione di una transazione che deve essere salvata all'interno di un blocco (come ad esempio la registrazione di una mucca, lotto di latte o prodotto caseario), mentre altre portano ad operazioni che non pesano sul portafoglio dell'utente (come ad esempio la ricerca di un prodotto tramite ID).

Di seguito viene mostrata una tabella che mostra il costo massimo, minimo e medio (in ETH) di ogni funzione presente nel contratto. In genere, un costo elevato è dovuto dalla lunghezza delle stringhe inserite dall'utente e utilizzate in quella specifica funzione.

Funzioni	Costo min	Costo max	Costo avg
Add Dairy	0,0002413	0,000672	0,00045665
Add Cow	0,000608	0,001424	0,001016
Add Milk	/	/	0,000508
Add Product	0,000302	0,00048	0,000391
Report	/	/	0,000131
Kill Cow	/	/	0,0001205
Transfer Cow	/	/	0,0006555
Transfer Milk	/	/	0,0003625

Fig. 2: Tabella dei costi per ogni transazione

Dalla tabella precedente si può notare come le funzioni `Add Dairy`, `Add Cow` e `Add Milk` richiedono una particolare cura in quanto la presenza di campi stringa rischia di influire sui costi a prescindere dalla quantità di informazioni nella catena. Per questo motivo, per calcolare il costo minimo sono state utilizzate delle stringhe ad un solo carattere mentre per calcolare il costo massimo sono state utilizzate delle stringhe a 50 caratteri.

Transazioni	Add Dairy	Add Cow	Add Milk	Add Product	Report	Kill Cow	Transfer Cow	Transfer Milk
1	0,0002418	0,0007084	0,0003865	0,0003173	0,0001549	0,0001154	0,001179	0,000953
2	0,0002418	0,0007084	0,0003865	0,0003173	0,0001615	0,0001154	0,001143	0,0009546
3	0,0002418	0,0007084	0,0003865	0,0003173	0,0001635	0,0001154	0,001146	0,0009677
4	0,0002418	0,0007084	0,0003865	0,0003173	0,0001653	0,0001154	0,00115	0,0009679
5	0,0002418	0,0007084	0,0003865	0,0003173	0,0001658	0,0001154	0,001152	0,0009681
6	0,0002418	0,0007084	0,0003865	0,0003173	0,0001666	0,0001154	0,001155	0,0009683
7	0,0002418	0,0007084	0,0003865	0,0003173	0,0001669	0,0001154	0,001159	0,0009684
8	0,0002418	0,0007084	0,0003865	0,0003173	0,000167	0,0001154	0,001161	0,0009688
9	0,0002418	0,0007084	0,0003865	0,0003173	0,0001672	0,0001154	0,001163	0,0009689
10	0,0002418	0,0007084	0,0003865	0,0003173	0,000168	0,0001154	0,001167	0,0009694
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
28	0,0002418	0,0007084	0,0003865	0,0003173	0,0001716	0,0001154	0,001254	0,0009765
29	0,0002418	0,0007084	0,0003865	0,0003173	0,0001745(*)	0,0001154	0,001373(*)	0,0012(*)
30	0,0002418	0,0007084	0,0003865	0,0003173	0,0002786(**)	0,0001154	0,001707(**)	0,001526(**)

Fig. 3: Tabella dei costi per ogni transazione
(*) Elementi in posizione centrale
(**) Elementi in posizione ultima

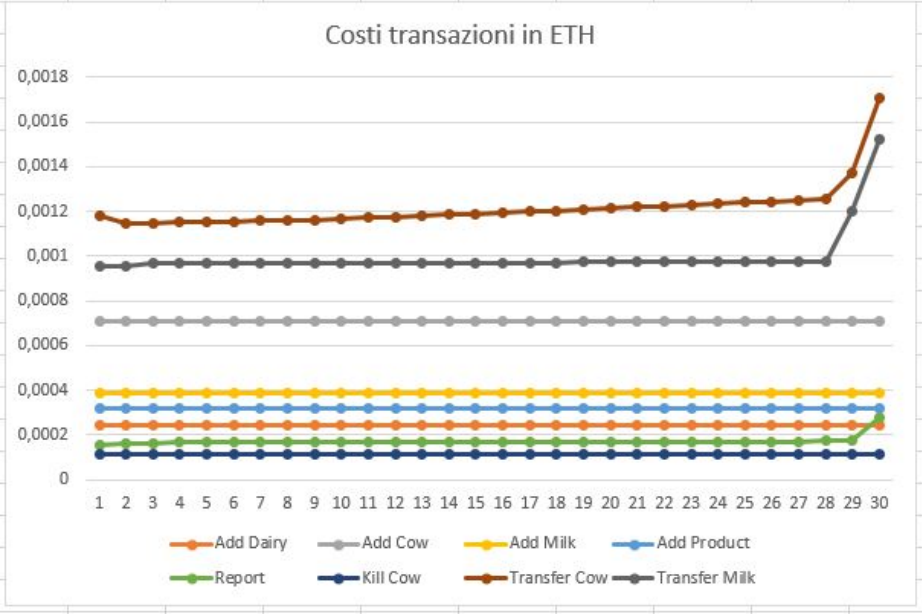

Fig. 4: Grafico dei costi delle transazioni

Come è possibile notare nella Fig. 3 e nella Fig. 4, i costi delle transazioni per la maggior parte delle funzioni rimangono stabili nel tempo. Ciò è dovuto al fatto che nessuna operazione di ricerca viene effettuata e la lunghezza delle catene nella blockchain rimane costante.

Tuttavia, osservando il grafico, notiamo che per le operazioni `TransferCow`, `TransferMilk` e `Report`, il prezzo aumenta leggermente nel tempo. Questo fenomeno è causato dall'uso di funzioni di ricerca nel codice per individuare la mucca, il latte o il prodotto sulla base del loro ID. Poiché per memorizzare tutti questi elementi vengono utilizzate delle liste, gli elementi che si trovano in posizioni più lontane dalla testa della lista richiederanno un consumo di gas leggermente maggiore rispetto agli altri elementi che li precedono.

5 Conclusioni

Nel presente lavoro, è stata proposta una Dapp per il tracciamento dei prodotti caseari. Questa Dapp consente a qualsiasi caseificio di registrarsi e, successivamente, di registrare mucche, lotti di latte e prodotti caseari, ciascuno identificato da un ID univoco.

Il caseificio avrà la possibilità di eliminare una mucca dalla blockchain o di trasferire una mucca o un lotto di latte ad un altro caseificio registrato nella Blockchain.

Inoltre, la Dapp fornirà agli utenti la possibilità di ottenere tutte le informazioni riguardanti un prodotto acquistato, utilizzando il suo ID, e di segnalare eventuali difetti o avarie del prodotto. Grazie a questa funzionalità, tutti gli altri prodotti realizzati con lo stesso lotto di latte di un prodotto segnalato come difettoso potranno essere identificati e trattati di conseguenza.

5.1 Considerazioni

Dopo aver realizzato la Dapp ed eseguito l'analisi dei costi, siamo giunti alla seguenti considerazioni o critiche:

1. Nell'attuale Dapp, gli utenti sono tenuti a pagare una certa quantità di ETH per segnalare un prodotto avariato. Questo requisito è fondamentale per garantire la massima decentralizzazione dell'applicazione web. Tuttavia, potrebbe rappresentare un potenziale ostacolo all'incitamento degli utenti a segnalare prodotti difettosi, poiché comporta un costo a loro carico.
2. Nell'attuale Dapp, qualsiasi competitor potrebbe segnalare un prodotto come un falso difettoso o avariato al solo fine di svalutare l'azienda produttrice poiché questa operazione non richiede alcun controllo di veridicità;
3. Nell'attuale Dapp, come evidenziato dall'analisi dei costi, i prodotti che si trovano in posizioni più lontane dalla testa della lista, dove sono memorizzati, presentano un costo maggiore. Questo aspetto potrebbe causare fluttuazioni nei prezzi, e un utente potrebbe trovarsi a pagare un prezzo più alto rispetto a un altro utente, semplicemente perché il suo prodotto è stato realizzato in un momento successivo. Dal grafico in Fig. 4 si può dedurre che il prezzo aumenterà sempre nel tempo.

5.2 Lavori futuri

Come possibili lavori futuri e sviluppo per la nostra Dapp, oltre a trovare delle possibili soluzioni sui problemi emersi e descritti del capito precedente, suggeriamo l'implementazione dello **standard ERC-721**.

Lo standard ERC-721 è un protocollo di token non fungibili (NFT) basato sulla blockchain Ethereum. Gli NFT sono token unici, ciascuno con una sua identità e valore distinti. Questo standard è stato proposto per consentire la creazione, il trasferimento e l'interoperabilità di tali token su diverse applicazioni decentralizzate (DApp) all'interno dell'ecosistema Ethereum.
Caratteristiche principali dello standard ERC-721:

- Unicità;
- Ownership;
- Trasferibilità;
- Metadata;
- Interoperabilità.

Queste caratteristiche hanno reso gli NFT basati su ERC-721 molto popolari nell'ambito delle collezioni digitali, giochi blockchain, arte digitale, tokenizzazione di asset unici e in altre applicazioni che richiedono l'attribuzione di un valore e un'identità distinti a specifici oggetti digitali.

Un altro possibile sviluppo futuro potrebbe essere l'impiego di ispettori per valutare con precisione le segnalazioni di prodotti difettosi o avariati. Ciò consentirebbe di effettuare controlli su ciascun prodotto segnalato come danneggiato, aumentando la veridicità delle informazioni presenti nella blockchain e riducendo il rischio di segnalazioni false. Tuttavia, bisogna tenere presente che questa soluzione comporterebbe una minore decentralizzazione, poiché verrebbe riposta fiducia in figure centrali ed esterne alla blockchain.