Università degli Studi di Parma

Ingegneria Informatica Elettronica e delle Telecomunicazioni
Ingegneria dei Sistemi Informativi

Tecnologie Internet

Cloudest
Cloud storage decentralizzato basato sulla blockchain Ethereum e IPFS

Filippo Scaramuzza
Luca Pastori

A.A. 2020/2021

Contents

1 Introduzione

1.1 Il problema

Secondo un articolo di SeedScientific [1], ogni giorno vengono prodotti 2,5 quintilioni di byte di dati. Va notato che sul totale dei dati nel mondo, oltre il 90% dei dati è stato prodotto negli ultimi 4 anni. La maggior parte dei dati attualmente disponibile attraverso Internet è però "centralizzata", ovvero archiviata da un numero molto ristretto di aziende tecnologiche che hanno l'esperienza e il capitale per costruire enormi data center in grado di gestire queste enormi quantità di dati. Il problema fondamentale di questo approccio è proprio la centralità dei dati. Questi ultimi infatti sono alla mercé di malintenzionati che possono accedere, modificare o addirittura cancellare i dati caricati nei data center. Quando si parla di aziende molto grandi questo è un rischio molto basso, ma un altro grande problema è dietro l'angolo. Ciò che è più importante per un consumatore è la sua privacy, molto spesso messa a repentaglio da agenzie terze che utilizzano i dati caricati per fini di marketing e pubblicità, spesso all'insaputa dei consumatori. Inoltre, il costo sostenuto per l'archiviazione dei dati in server centralizzati è molto alto e di grande impatto sull'ambiente e molte volte gli utenti finali devono pagare per un piano di archivazione anche se utilizzano solo una minima parte dello spazio che hanno effettivamente comprato. Un ulteriore problema è la scalabilità del sistema: è infatti difficile scalare un sistema di archiviazione centralizzato per soddisfare la crescente domanda.

1.2 La possibile soluzione

La piattaforma che questa relazione vuole descrivere, Cloudest, tenta di risolvere i numerosi problemi tecnici, etici ed economici dell'attuale sistema di archiviazione centralizzato.

La blockchain è di fatto un sistema di cloud storage decentralizzato che garantisce la sicurezza dei dati. Qualsiasi nodo di elaborazione connesso a Internet può unirsi e formare una rete di peer, massimizzando così l'utilizzo delle risorse. Ogni nodo della rete memorizza una copia della blockchain rendendola così immutabile. In Cloudest l'utente può caricare un file sulla piattaforma che a sua volta invia la risorsa alla rete P2P IPFS (*InterPlanetary File System*). Come si vedrà nel seguito, IPFS utilizza un sistema di *hash* per recuperare i file distribuiti nella rete, Cloudest si occupa di salvare questi hash e renderli disponibili salvandoli con tutte le informazioni necessarie sulla blockchain.

2 L'architettura di Sistema

2.1 Struttura e schema generale

Cloudest, raggiungibile al link https://cloudest.cloud è un'*applicazione decentralizzata*, struttura anche conosciuta come *DApp*, appunto *Decentralized Application*. La struttura può essere rappresentata come mostrato in Figura 1.

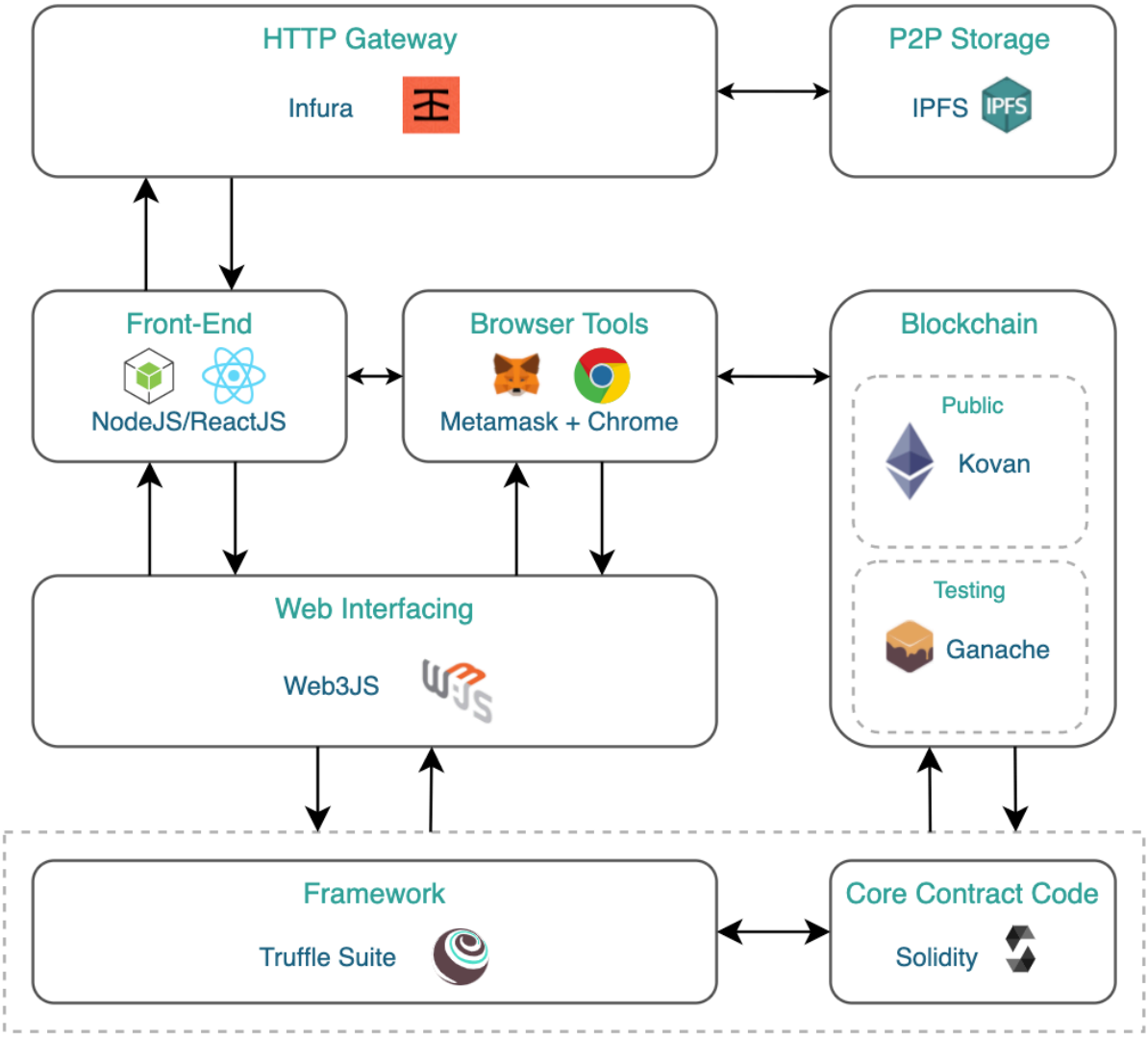

Figure 1: Architettura di Cloudest

Di seguito viene fatta una breve descrizione delle componenti principali.

2.2 Framework: Truffle Suite

Truffle [2] è un framework di test e una pipeline di risorse per la blockchain che utilizza la *Macchina Virtuale Ethereum* (EVM), con l'obiettivo di *semplificare* la vita agli sviluppatori. Fornisce infatti una serie di strumenti volti alla migrazione di contratti sulle reti private e pubbliche ed una serie di strumenti per l'integrazione con le Web App, in particolare interfacciandosi con la libreria Web3JS.

2.3 Core Contract Code: Solidity

Solidity [3] è un linguaggio ad alto livello orientato agli oggetti utilizzato per l'implementazione di *smart contracts*. Nello specifico caso di Cloudest, viene utilizzato un solo contratto (si veda nel seguito) che si occupa di organizzare tramite particolari strutture i dettagli dei file associandoli ai singoli utenti.

2.4 Web Interfacing: Web3JS

Web3JS [4] è una libreria scritta interamente in Javascript che permette l'interazione, tramite opportuni strumenti, ad esempio Metamask (si veda nel seguito), con la blockchain, nel caso di Cloudest tramite HTTP. Infatti una volta che viene eseguito il comando `truffle compile` (comando specifico per la compilazione di contratti della suite di Truffle), viene creato un file `json` relativo al contratto che Web3JS interpreta, richiamando ad esempio specifiche funzioni del contratto dal codice Javascript.

2.5 Front-End e Browser Tools

2.5.1 NodeJS/ReactJS

Cloudest utilizza come *client* un'applicazione Javascript che utilizza NodeJS [5], in particolare ReactJS [6] per gestire la parte grafica. Lo scopo di React è quello di visualizzare i dati recuperati dalla blockchain e da IPFS mostrandoli all'utente e fornisce gli strumenti (bottoni, textbox, ecc...) per l'interazione con l'applicazione e IPFS nonché con la blockchain.

2.5.2 Metamask + Chrome

Lo sviluppo di Cloudest è stato orientato ad un suo utilizzo su Chrome, scelta fatta considerando soprattutto l'utilizzo dell'estensione *Metamask* [7], ben integrata con Chrome. Metamask è di fatto un wallet per Ethereum, che si integra alla perfezione con le applicazioni sviluppate ad esempio in javascript, e soprattutto con Web3JS. Infatti fornisce un modo semplice per effettuare le transazioni necessarie per l'utilizzo dei contratti e permette di effettuare uno *switch* rapido tra reti Ethereum differenti.

2.6 Blockchain

2.6.1 Testing: Ganache

Ganache [8] è una blockchain personale per lo sviluppo rapido di applicazioni distribuite Ethereum e Corda. È possibile utilizzare Ganache in tutto il ciclo di sviluppo, consentendo di sviluppare, distribuire e testare le DApps in un ambiente sicuro e deterministico.

2.6.2 Public: Kovan

Kovan [9] è una blockchain *Proof of Authority* (PoA) accessibile al pubblico di Ethereum;Questa nuova testnet utilizzerà Parity (un client Ethereum sviluppato da Parity Technologies, FKA Ethcore) per fornire un ambiente testnet stabile e sicuro per gli sviluppatori Ethereum. Si è deciso di utilizzare Kovan come testnet e non reti più conosciute come ropsten data la sua stabilità e velocità nell'elaborare nuove transazioni.

2.7 HTTP Gateway: Infura

Infura [10] è un framework che fornisce diversi servizi sia per l'interazione con la blockchain di Ethereum (ad esempio per il *deploy* dei contratti su reti pubbliche, fungendo da *HTTP Provider*) o per l'interazione con IPFS (fungendo da *Gateway*). Nel caso di cloudest Infura è stato utilizzato sia per interagire con IPFS che per effettuare il deploy del contratto sulla rete *Kovan*.

2.8 P2P Storage: IPFS

IPFS [11], *Interplanetary File System* è un protocollo hypermedia distribuito peer-to-peer open source che mira a funzionare come un file system per tutti i dispositivi di elaborazione connessi. Il suo funzionamento consiste nel suddividere i file, ai quali viene assegnato un identificatore univoco un hash crittografico, in blocchi e distribuiti ai nodi della rete. I duplicati vengono rimossi attraverso la rete e la cronologia delle versioni viene tracciata per ogni file. Ciò porta a contenuti permanentemente disponibili, inoltre, l'autenticità del contenuto è garantita attraverso questo meccanismo e, quando un dispositivo cerca i file, si chiede essenzialmente alla rete di trovare nodi che memorizzano il contenuto dietro l'hash di identificazione univoco associato a quel contenuto.

3 Lo sviluppo di Cloudest

In questa sezione verranno mostrati e citati gli strumenti e le tecnologie utilizzate per lo sviluppo e per la gestione del progetto Cloudest.

3.1 Organizzazione e Time Management

Lo sviluppo di Cloudest è stato suddiviso in 3 grandi *milestone*.

3.1.1 Milestone 1: Faucet

Nella prima lo sviluppo è stato incentrato su una semplice applicazione che fungeva da *Faucet* (una sorta di "banca" per ethereum. Un utente può contattare il Faucet così da richiedere una somma in Ether. E' uno strumento comune, naturalmente, solo nelle testnet). Da questa applicazione in particolare si può anche contribuire a fornire fondi al Faucet. Questa prima milestone è servita per prendere dimestichezza con l'interazione con Metamask e con i contratti sulla blockchain. In Figura 2 uno screenshot dell'applicazione.

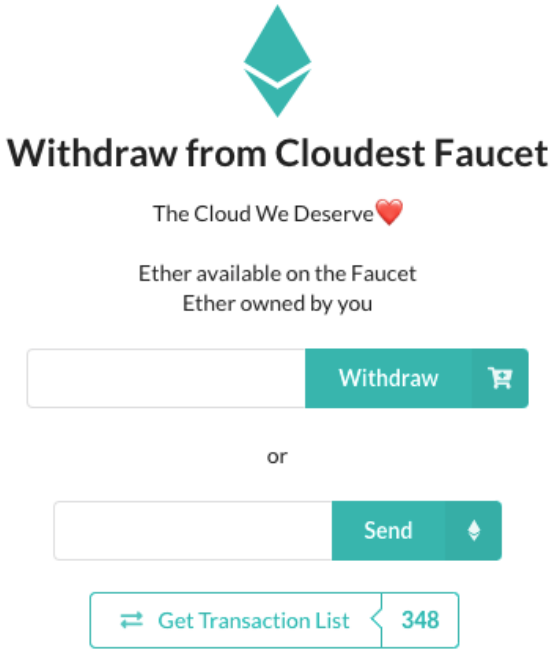

Figure 2: Cloudest Faucet User Interface

3.1.2 Milestone 2: prima integrazione con IPFS

La seconda milestone è servita a prendere dimestichezza con il sistema *Infura + IPFS*. In questa semplice applicazione è stato testato l'upload e il download di un singolo file dalla rete IPFS. Uno screenshot dell'applicazione è in Figura 3.

3.1.3 Milestone 3: Integrazione e sviluppo della Piattaforma

La terza e ultima milestone, nonché la più corposa, è stata volta allo sviluppo della piattaforma vera e propria, come integrazione delle due milestone precedenti e delle varie funzionalità e scrittura del contratto.

Figure 3: Test di upload/download IPFS

3.2 Time/Tasks Management e Sviluppo

3.2.1 Time/Tasks Management

Per la gestione dei tasks, del tempo, delle scadenze e in generale per l'organizzazione di un progetto di per se piuttosto complesso è stata utilizzata la piattaforma *Microsoft Planner* [12], di cui è riportato uno screenshot in Figura 4.

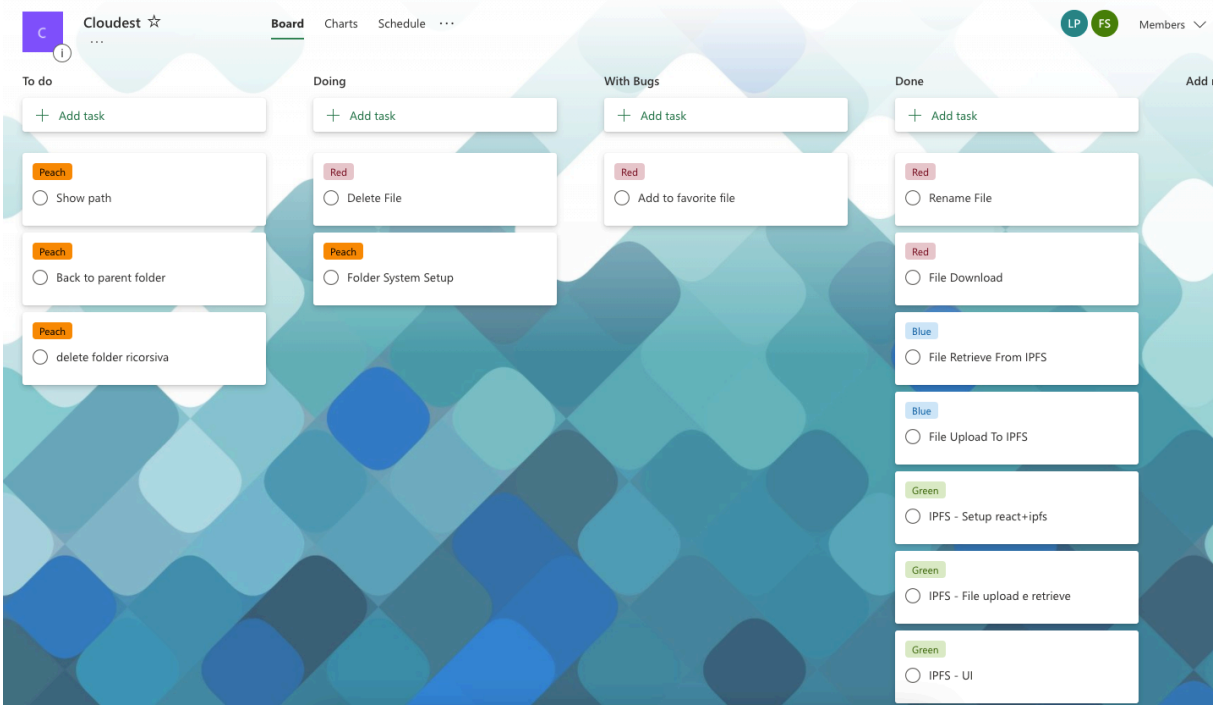

Figure 4: Microsoft Planner

3.2.2 Sviluppo

Per conservare in modalità sicura il codice scritto si è utilizzato Github un'applicazione utile per store e versioning del codice. Per lo sviluppo della piattaforma è stato utilizzato interamente Visual Studio Code [13], scelto per la sua semplicità e comodità d'uso, nonché per le sue funzionalità. Infatti per renderlo idoneo all'utilizzo sono state aggiunte alcune estensioni come ad esempio *Live Shere* utile per lo sviluppo simultaneo di codice in tempo reale. Con questa estensione è stato anche possibile condividere il server di sviluppo (`localhost`) su tutti i dispositivi connessi alla sessione di Live Share, così da poter operare e visualizzare contemporaneamente il codice e il suo effettivo funzionamento, come se fosse un solo computer.

Grazie a questi strumenti è stato possibile sviluppare l'intera piattaforma interamente a distanza.

4 Il funzionamento e l'utilizzo di Cloudest

Per ogni sezione viene mostrato il funzionamento della piattaforma dal lato dell'utente finale (Foreground) e dal lato della piattaforma, con i dettagli tecnici sul funzionamento (Background).

4.1 Avvio e "login"

4.1.1 Foreground

Quando l'utente si reca su https://cloudest.cloud questa è la schermata che gli si presenta (Figura 5). Se

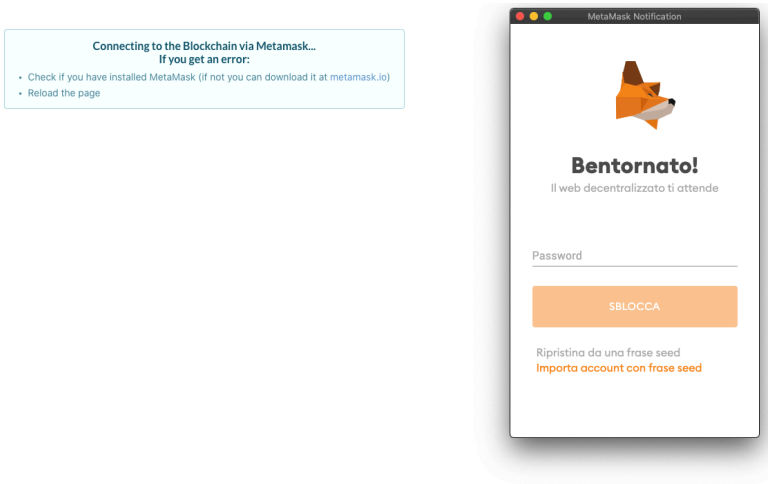

Figure 5: Avvio e login

tutto funziona correttamente, Metamask propone all'utente di fare il login e sbloccare il portafoglio, così da permettere all'applicazione di accedere a quest'ultimo. Se di nuovo tutto funziona correttamente all'utente viene mostrata la pagina dell'applicazione vera e propria, di cui uno screenshot è mostrato in Figura 6.

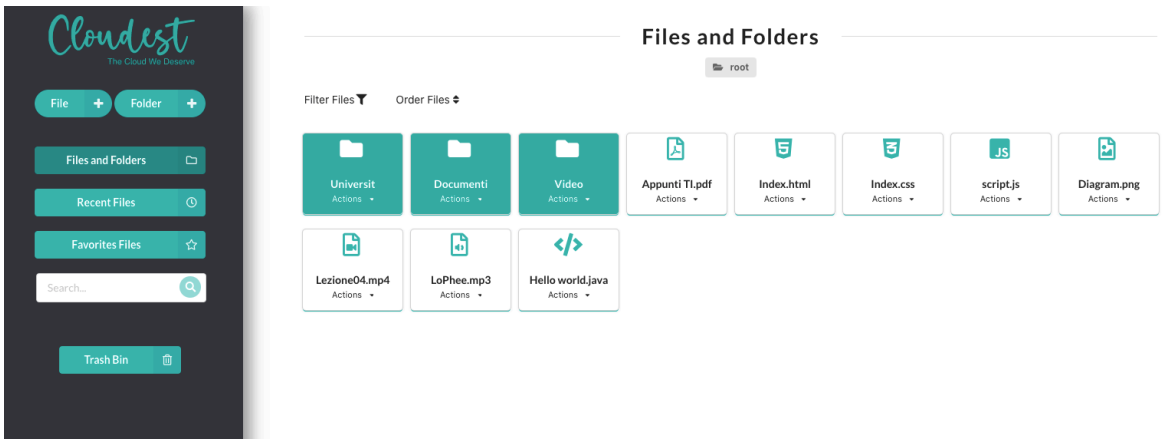

Figure 6: Home di Cloudest

4.1.2 Background

Ciò che accade "dietro le quinte" è ciò che consente a Metamask e all'applicazione React di comunicare. Tramite una particolare funzione (`getWeb3()`), viene recuperato l'oggetto Javascript `web3` dall'oggetto `window`. Questo oggetto è fornito direttamente da Metamask e permette di comunicare con la libreria Web3JS.

4.2 Upload di un file/cartella

4.2.1 Foreground

L'utente cliccando sul pulsante "File +" può caricare un file dal popup che gli viene mostrato. Una volta confermato il caricamento su IPFS viene mostrato all'utente il popup di Metamask che chiede la conferma della transazione. In Figura 7 si può vedere uno screenshot della funzionalità di upload.

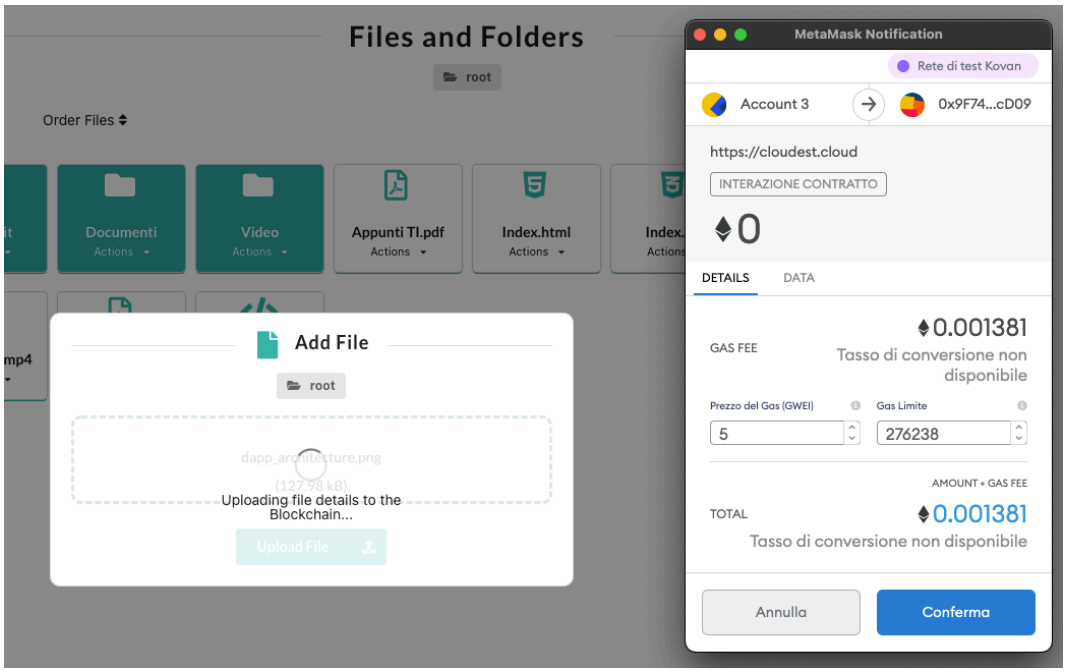

Figure 7: Home di Cloudest

La procedura per la creazione di una cartella è pressoché identica a quella dell'upload di un file.

4.2.2 Background

Per quanto riguarda l'upload dei File, vengono richiamate principalmente due funzioni. Con la prima il file viene caricato sulla rete IPFS. Questa funzione restituisce poi l'hash identificativo del file, che viene caricato insieme a tutte le informazioni sulla blockchain dalla seconda funzione richiamata.

Le cartelle vengono trattate dallo smart contract esattamente come un file, il cui tipo è però `folder`. Il sistema di cartelle e sottocartelle è stato implementato tramite l'aggiunta di un attributo `parentFolder` ad ogni file e ad ogni cartella, in modo tale che sia possibile ricostruire *on demand* la struttura ad albero del filesystem, permettendo anche la cancellazione *ricorsiva* dei file e delle sottocartelle che li contengono.

Il file, o meglio, il suo buffer, viene cifrato prima di essere caricato sulla rete IPFS tramite l'algoritmo *AES-256* utilizzando come chiave di cifratura l'hash di 32 byte dell'indirizzo del portafoglio dell'utente ottenuto tramite l'algoritmo di hashing *keccak256*.

4.3 Download, Rename File e Add To Favorite

4.3.1 Foreground

L'utente premendo in "action", link visibile in ogni file e cartella, ha accesso alle funzioni quali download dell'elemento, rinominarlo o l'aggiunta dei favoriti. In figura 8 si può vedere uno screenshot delle funzionalità di download, rename e add to file.

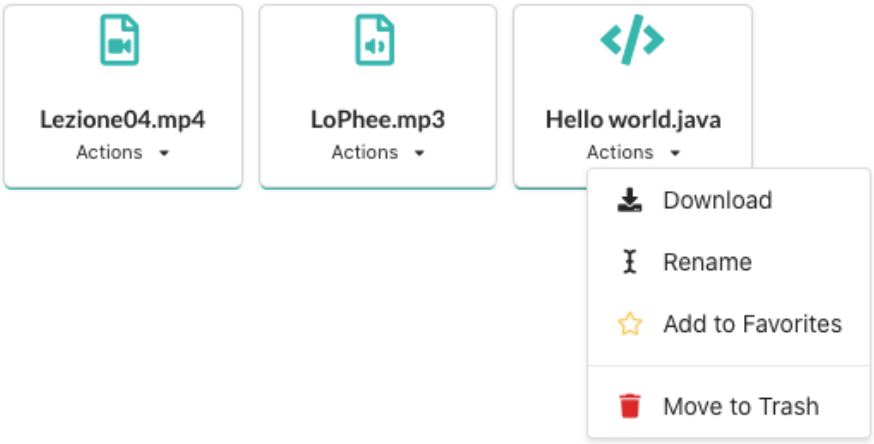

Figure 8: Download, Rename File e Add To Favorite

4.3.2 Background

Quando l'utente preme su "Download" viene recuperato il buffer del file dalla rete IPFS tramite l'hash del suddetto file per essere decifrato. A questo punto il file viene scaricato tramite la funzione `fileDownload()` con tutte le informazioni relative (quali nome ed estensione) recuperate dal contratto. Sia nel caso di "Rename" che nel caso di "Add/Remove to Favorites", c'è il richiamo ad una specifica funzione del contratto che si occupa di modificare gli attributi del file secondo l'evenienza. Di seguito verrà approfondito il contratto e la sua struttura.

4.4 Move to Trash e Delete Definitively

4.4.1 Foreground

Quando si desidera eliminare un file per prima cosa l'utente dovrà spostarlo nel cestino. All'interno di esso si potrà scegliere se eliminare definitamente il file/cartella oppure recuperarlo e riportarlo nella posizione originaria. In Figura 9 si può vedere uno screenshot delle funzionalità di delete.

4.4.2 Background

Come accennato sopra il sistema di foldering è basato sul concetto di *parent folder*, così da poter effettuare, se necessaria una cancellazione ricorsiva. C'è infatti un particolare metodo nel contratto che verrà approfondito in seguito, in cui in modo ricorsivo vengono spostati i file nel cestino oppure eliminati definitivamente. Va notato che se si sposta una cartella nel cestino ogni file e cartella al suo interno verranno spostati nel cestino, per essere eventualmente spostati nella cartella root se recuperati dal cestino. Questo implica quindi che la struttura di cartelle e sottocartelle viene mantenuta solo se ad essere spostata nel cestino è una intera cartella.

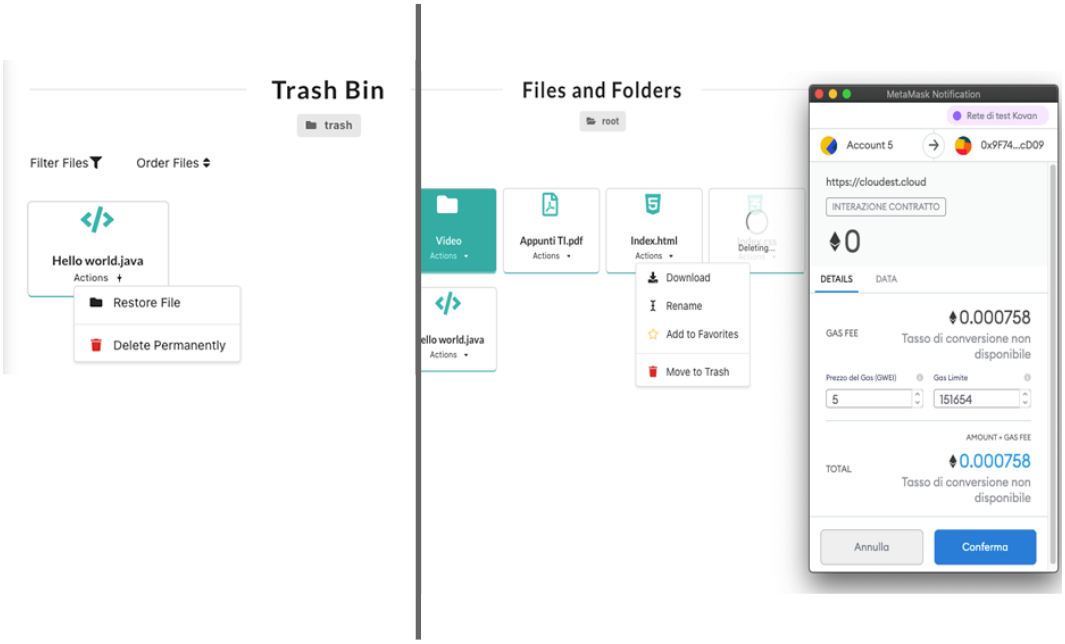

Figure 9: Move to Trash e Delete Definitively

4.5 File Preferiti, File Recenti, Ricerca e Filtri

4.5.1 Foreground

L'utente può, tramite i pulsanti nella barra di navigazione laterale, spostarsi tra la pagina che contiene tutte le cartelle, sottocartelle e file cliccando su "Files and Folders", oppure spostarsi nelle pagine "Favorites Files", che mostra i file marcati come *favorite*, e "Recent Files". In questa pagina è possibile visionare tramite opportuni filtri i file caricati nell'ultima ora, nella giornata corrente, nell'ultima settimana o nell'ultimo mese.

Vari filtri sono stati implementati anche nella pagina "Files and Folders", dando così all'utente la possibilità di visualizzare solo alcuni tipi di file oppure ordinarli secondo data o nome (crescenti o decrescenti).

E' stata inoltre implementata la funzione di ricerca (anch'essa accessibile dalla barra laterale), che permette di ricercare qualsiasi tipo di file dal suo nome, in tutte le cartelle e sottocartelle.

4.5.2 Background

Tutte le funzionalità sopra esposte sono state realizzate tramite le funzioni di React. In particolar modo è stato utilizzato durante tutto lo sviluppo il concetto di *state*, caratteristico delle compoenenti di React, che permette di memorizzare dati (e.g. la cartella corrente in cui si trova l'utente), aggiornarli e modificarli, innescando così determinate azioni. L'utilizzo di questi meccanismi ha fatto sì che sia stato possibile realizzare una cosiddetta *single-page application*, in *real-time* (i.e. senza la necessità di aggiornare la pagina per ristabilire la connessione con Node, mantenendo infatti una connessione costante).

5 Il contratto FileDetailsManager.sol

FileDetailsManager è il nucleo di tutta la piattaforma. Come accennato nelle sezioni precedenti, *FileDetailsManager* è uno *smart contract* il cui scopo fondamentale è quello di gestire, memorizzandoli e modificandoli, i dati dei singoli utilizzatori di Cloudest. Qesti *dati*, non sono i dati personali (si ricorda che lo scopo della piattaforma è anche quello di mantenere la password), bensì i dettagli dei file che gli utenti caricano nel "cloud" tramite Cloudest.

5.1 Strutture dati

5.1.1 Struttura "FileDetails"

I dettagli di ogni singolo file sono salvati in una particolare struttura dati (una `struct`), così formata:

```
struct FileDetails {
    string uniqueId; //unique id of the folder or the file's IPFS fileHash
    string name; // folder's name or file's name
    string transactionDate; // The date in which file's info was stored
    string fileType; // "folder" if folder otherwise the extension of the file
    string parentFolderId; // parent folder id (both for files and folders)
    bool isFavorite; // true if favorite, false otherwise
    bool isTrash; // true if currently in the trash, false otherwise
}
```

Come si può notare ad ogni "file" è associato il suo hash relativo ad IPFS, mentre ad ogni cartella (che si ipotizza possano avere nomi uguali), è associato un indirizzo univoco. Questo viene calcolato tramite un contatore che viene incrementato ogni volta che una cartella viene creata.

5.1.2 Mapping wallet → fileDetails

Come accennato sopra, ad ogni utente, o per meglio dire ad ogni *wallet* che interagisce con il contratto aggiungendo file, è assegnato un array contenente tutti i dettagli di tutti i file aggiunti. Per fare questo è stata utilizzata una struttura dati chiamata `mapping`, come segue:

```
mapping(address => FileDetails[]) filesList;
```

5.2 Metodi

5.2.1 AddFile

```
function addFile(string memory uniqueId,
                 string memory name,
                 string memory fileType,
                 string memory date,
                 string memory parentFolderId)
```

Questo metodo serve ad aggiungere un file all'array dei file di uno specifico *wallet*, con i dati specificati nei parametri.

5.2.2 AddFolder

```
function addFolder(string memory name,
                   string memory date,
                   string memory parentFolderId)
```

Questo metodo serve ad aggiungere una cartella all'array di uno specifico *wallet*, con i dati specificati nei parametri.

5.2.3 deleteFile

```
function deleteFile(string memory uniqueId,
                    string memory name)
```

Questo metodo rimuove uno specifico file dall'array di file di uno specifico *wallet*.

5.2.4 deleteFolder

```
function deleteFolder(string memory uniqueId)
```

Questo metodo permette la cancellazione di una cartella ed, in modo ricorsivo, la cancellazione di tutti i file e di tutte le cartelle al suo interno.

5.2.5 getFiles

```
function getFiles() returns (FileDetails[])
```

Questo metodo restituisce l'array contente file e cartelle di uno specifico *wallet*

5.2.6 renameFileName

```
function renameFileName(string memory uniqueId,
                        string memory name,
                        string memory newName)
```

Questo metodo permette di rinominare uno specifico file di uno specifico *wallet*.

5.2.7 setFavorite

```
function setFavorite(string memory uniqueId,
                     string memory name,
                     bool isFavorite)
```

Questo metodo permette di impostare un nuovo valore per il campo `isFavorite` di uno spcifico file di uno specifico *wallet*.

5.2.8 setTrashFile

```
function setTrashFile(string memory uniqueId,
                      string memory name,
                      bool isTrash,
                      bool isSingleDelete)
```

Questo metodo permette di impostare un nuovo valore per il campo `isTrash` di uno specifico file o wallet

5.2.9 setTrashFolder

```
function setTrashFolder(string memory uniqueId,
                        bool isTrash,
                        bool isSingleDelete)
```

Questo metodo permette di impostare un nuovo valore per il campo `isTrash` di una cartella e in modo ricorsivo per tutte le sottocartelle ed i file in esse contentuti.

6 Firebase

Cloudest è raggiungibile dal link https://cloudest.cloud. Per renderlo pubblico è stata utilizzata la piattaforma *Firebase* [14], una piattaforma per la creazione e la gestione di applicazioni per dispositivi mobili e web sviluppata da Google.

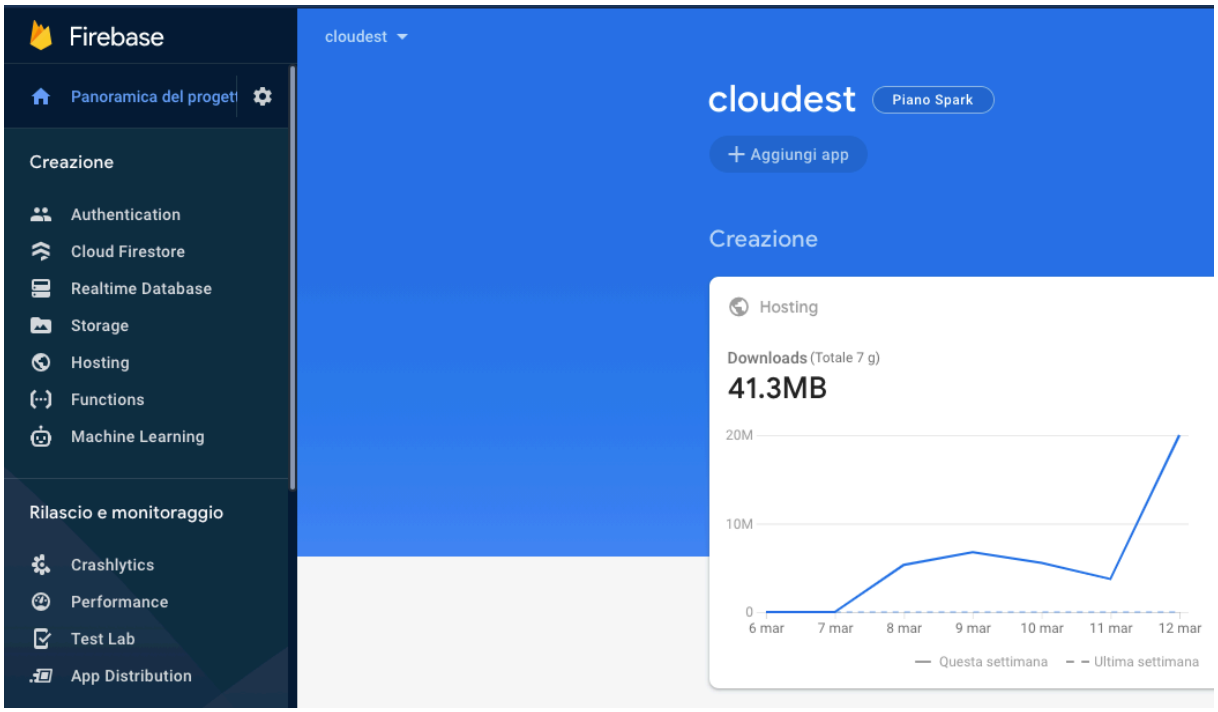

Figure 10: Firebase console

References

[1] SeedScientific Article, https://seedscientific.com/how-much-data-is-created-every-day, *How Much Data Is Created Every Day?*, Branka Vuleta, 28 Gen - 2021.

[2] Truffle Suite, https://www.trufflesuite.com/, ConsenSys Software Inc.

[3] Solidity Language, https://soliditylang.org/, Solidity Team

[4] Web3JS, https://github.com/ChainSafe/web3.js, *Ethereum Javascript API*, ChainSafe

[5] NodeJS, https://nodejs.org/, OpenJS Foundation

[6] ReactJS, https://reactjs.org/, Facebook Inc.

[7] Metamask, https://metamask.io/, Metamask, a ConsenSys Formation

[8] Ganache, https://www.trufflesuite.com/ganache, ConsenSys Software Inc.

[9] Kovan Testnet, https://kovan-testnet.github.io/website/, Parity Technologies

[10] Infura, https://infura.io/, Infura Inc.

[11] IPFS, https://ipfs.io/, Protocol Labs

[12] Microsoft Planner, https://www.microsoft.com/, Microsoft

[13] Visual Studio Code, https://code.visualstudio.com/, Microsoft

[14] Firebase, https://firebase.google.com/, Google Developers, Google