



UNIVERSITÀ DI TORINO

Dipartimento di informatica
Corso di Laurea in Informatica
Anno Accademico 2021/2022

**Un sistema di gestione degli accessi basato su smart contract per la
memorizzazione di file su una rete IPFS per il progetto Andromeda**

Relatore:
Prof. Claudio Schifanella

Candidato:
Andrea Birritteri

Sommario

Abstract	4
Introduzione	5
Descrizione Azienda	5
Andromeda.....	5
Motivazione Tesi.....	6
Requisiti Progetto	6
1 Blockchain.....	8
1.1 Algoritmi di consenso.....	8
1.1.1 Proof of Work (PoW).....	9
1.1.2 Proof of Stake (PoS)	9
1.1.3 Proof of Authority (PoA)	10
2 Tecnologie Blockchain utilizzate	11
2.1 Ethereum.....	11
2.1.1 Gas.....	13
2.1.2 Smart Contracts.....	13
2.1.3 Dapp	13
2.2 IPFS	15
2.2.1 IPFS vs HTTP	15
2.2.2 Swarm Key.....	16
2.2.3 Kubo IPFS.....	16
2.3 Metamask	17
2.4 Remix IDE	18
2.5 Protocollo JSON-RPC	18
2.5.1 JSON-RPC in Ethereum.....	20
2.6 EAHP - Ethereum Access HTTP Proxy	21
2.6.1 Protocolli Ethereum di rete.....	21
2.6.2 Soluzione aziendale.....	21
3 Tecnologie di supporto	23
3.1 REST API	23
3.2 RPC API.....	23
3.3 Java Spring.....	23
3.4 Vue.js.....	24
4 Progetto.....	25

4.1	Architettura	25
4.1.1	Upload file	27
4.1.2	Retrieve file	28
4.1.3	Lista caricamenti dei file	29
4.2	Sicurezza.....	29
4.3	Rete IPFS Privata.....	30
4.4	Smart Contracts	31
4.4.1	IpfsMetadata.sol	31
4.4.2	MyAccessControl.sol	33
4.5	Server-Side	33
4.5.1	Upload File	34
4.5.2	Retrieve File.....	34
4.5.3	Payload Validator	34
4.6	Client-Side	35
4.6.1	Payload Generator	35
4.6.2	Upload File	36
4.6.3	Retrieve File.....	37
4.6.4	Lista degli ultimi eventi dello Smart Contract	37
5	Conclusioni.....	39
	Ringraziamenti	40
	Bibliografia	41
	Indice delle figure	43

DICHIARAZIONE DI ORIGINALITÀ

"Dichiaro di essere responsabile del contenuto dell'elaborato che presento al fine del conseguimento del titolo, di non avere plagiato in tutto o in parte il lavoro prodotto da altri e di aver citato le fonti originali in modo congruente alle normative vigenti in materia di plagio e di diritto d'autore. Sono inoltre consapevole che nel caso la mia dichiarazione risultasse mendace, potrei incorrere nelle sanzioni previste dalla legge e la mia ammissione alla prova finale potrebbe essere negata."

Abstract

Questa tesi esamina lo sviluppo di una piattaforma decentralizzata per la condivisione e gestione di documenti, implementata sulla base di Andromeda, una blockchain basata su Ethereum progettata per facilitare l'integrazione della tecnologia blockchain in soluzioni aziendali. Andromeda mira a promuovere l'adozione di soluzioni enterprise basate su blockchain permissionless Ethereum.

Durante il progetto, ho contribuito attivamente all'analisi, alla progettazione architeturale, allo sviluppo e ai test della piattaforma, con un focus particolare sull'implementazione di una soluzione decentralizzata per la condivisione e gestione di file e documenti. L'implementazione comprende lo sviluppo di componenti back-end e front-end, garantendo un'esperienza utente efficace e sicura.

Al termine del tirocinio ho acquisito competenze avanzate in analisi e progettazione, conoscenze tecniche ed architetture in ambito blockchain.

Inoltre, si è approfondito lo studio dei temi legati alla compliance e alla sicurezza delle informazioni, fornendo un quadro completo delle sfide e delle opportunità che emergono dall'integrazione della tecnologia blockchain nelle soluzioni aziendali.

Introduzione

Descrizione Azienda

Il Gruppo ALTEN, leader europeo nella consulenza per le tecnologie avanzate in campo ingegneristico e ICT, è quotato alla Borsa di Parigi e vanta più di 45.000 collaboratori in oltre 30 Paesi nel mondo.

In Italia ALTEN è presente su tutto il territorio nazionale con oltre 4,000 collaboratori e uffici collocati in diverse città italiane, tra cui Milano, Torino, Gallarate, Brescia, Genova, Padova, Modena, Bologna, Firenze, Roma, Napoli, e Bari.

ALTEN si propone al mercato con servizi legati al mondo dell'ingegneria, dell'information technology, delle telecomunicazioni e del life sciences, con numerosi centri di eccellenza tra cui: digital, business intelligence, testing, formazione (ALTEN Academy), ITSM, IoT, hw, sw embedded, quality e CSV. La forza di ALTEN si fonda sulla competenza e professionalità dei profili tecnici e manageriali, che vengono selezionati e formati con percorsi strutturati e replicati anche all'interno dell'ALTEN Academy, che eroga corsi di formazione e certificazione anche verso l'esterno. [1]

Andromeda

L'azienda ha richiesto una piattaforma decentralizzata di condivisione e gestione di file e documenti, aggiungendolo come modulo alla loro già esistente piattaforma Andromeda. La piattaforma Andromeda di Alten estende le funzionalità della blockchain Ethereum, permettendo l'integrazione della tecnologia in soluzioni aziendali, facilitando l'installazione, la gestione e il monitoraggio.

Andromeda risolve le principali criticità derivanti dall'utilizzo della Blockchain, ossia:

- Governance: gli strumenti di gestione e monitoraggio non sono inclusi nel software Geth
- Infrastruttura: i nodi di rete decentralizzati e distribuiti sono complessi e costosi da installare.
- Paradigma: i principi della blockchain devono essere applicati correttamente per utilizzare a pieno i suoi benefici e potenzialità

Inoltre, ha come obiettivo:

- fornire strumenti, librerie e linee guida utili all'integrazione della tecnologia blockchain.
- trattare temi legati alla compliance e alla notarizzazione dei documenti.

Andromeda Development Kit - ADK

Rappresenta una funzionalità che permette di applicare le metodologie di ingegneria del software aziendale tradizionale (riuso, ottimizzazione dei costi, alto disaccoppiamento) agli

Smart Contract. Questa funzionalità è costituita da un insieme di strumenti, spiegati di seguito.

- Platform Contract: Smart Contract che offrono le funzionalità base standardizzate, ossia storage, access control e contract registry;
- Service Contract: Smart Contract di utilità che offrono funzionalità avanzate, ossia checkpointing e DTT;
- Librerie Java e Javascript: per l'invocazione degli Smart Contract dalle componenti off-chain.

[2]

Motivazione Tesi

La piattaforma Andromeda di Alten estende le funzionalità della blockchain Ethereum, permettendo l'integrazione della tecnologia in soluzioni aziendali, facilitando l'installazione, la gestione e il monitoraggio.

Realizzata dopo tre anni di ricerca e sviluppo su progetti blockchain, gli obiettivi di Andromeda sono:

- accelerare l'implementazione di soluzioni enterprise basate su blockchain Ethereum, minimizzando le criticità di paradigma, governance e infrastruttura;
- fornire strumenti, librerie e linee guida utili all'integrazione della tecnologia blockchain.
- trattare temi legati alla compliance e alla notarizzazione dei documenti.

Attualmente, grazie all'utilizzo della blockchain Ethereum, Andromeda mette a disposizione una piattaforma di calcolo decentralizzato; al fine di soddisfare nuovi requisiti funzionali, è necessario evolvere la soluzione, aggiungendo una piattaforma decentralizzata di condivisione e gestione di file e documenti.

Sono stato coinvolto nelle fasi di analisi, progettazione architettuale, sviluppo e test.

Le attività di implementazione prevedono sviluppi back-end e front-end.

Al termine del periodo di tirocinio ho acquisito conoscenze di analisi e progettazione, competenze tecniche ed architetture su blockchain, competenze tecniche e metodologiche su DevOps e sicurezza. Inoltre, sono stati trattati i temi legati a compliance e sicurezza delle informazioni.

Requisiti Progetto

L'azienda ha richiesto una piattaforma decentralizzata di condivisione e gestione di file e documenti, aggiungendolo come modulo alla loro già esistente piattaforma Andromeda. La piattaforma Andromeda di Alten estende le funzionalità della blockchain Ethereum, permettendo l'integrazione della tecnologia in soluzioni aziendali, facilitando l'installazione, la gestione e il monitoraggio.

Andromeda risolve le principali criticità derivanti dall'utilizzo della Blockchain, ossia:

- Governance: gli strumenti di gestione e monitoraggio non sono inclusi nel software Geth
- Infrastruttura: i nodi di rete decentralizzati e distribuiti sono complessi e costosi da installare.
- Paradigma: i principi della blockchain devono essere applicati correttamente per utilizzare a pieno i suoi benefici e potenzialità

Inoltre, ha come obbiettivo:

- fornire strumenti, librerie e linee guida utili all'integrazione della tecnologia blockchain.
- trattare temi legati alla compliance e alla notarizzazione dei documenti.

Per la costruzione di questo modulo mi è stato richiesto di integrarmi con il preesistente sistema di gestione degli accessi, già utilizzato in Andromeda.

È stato richiesto di realizzare un'interfaccia utente che si allineasse a quella già presente nella piattaforma.

Per il salvataggio dei documenti è stato richiesto di realizzare una piattaforma di storage decentralizzato privato che esponesse i servizi tramite un server. Questo storage dovrà essere in comune e ogni utente dalla propria interfaccia potrà vedere i documenti caricati di tutti gli utenti autorizzati.

È stato richiesto che l'intero modulo permettesse all'utente di autenticarsi su Andromeda e successivamente caricare, scaricare un file e vedere gli ultimi file caricati.

In corso d'opera mi è stato richiesto di salvare, oltre al file, anche i suoi metadati e visualizzarli.

1 Blockchain

Una blockchain è un registro digitale decentralizzato di transazioni che viene utilizzato per registrare e verificare le transazioni in modo sicuro e trasparente. Si tratta di una catena di blocchi, ciascuno contenente un gruppo di transazioni, collegati tra loro mediante crittografia. Una volta aggiunto un blocco alla blockchain, le informazioni in esso contenute non possono essere alterate o cancellate.

Una delle caratteristiche principali di una blockchain è la sua natura decentralizzata, ovvero non è controllata da un'unica entità. La rete è invece gestita da una rete di computer, chiamati nodi, che collaborano per convalidare le transazioni e aggiungerle alla blockchain. Questa decentralizzazione rende le blockchain altamente resistenti a manomissioni, frodi e censure.

La prima e più nota applicazione della tecnologia blockchain è Bitcoin, che utilizza una blockchain per registrare e verificare le transazioni della sua criptovaluta. Tuttavia, i potenziali utilizzi della tecnologia blockchain vanno ben oltre le sole valute digitali e possono essere applicati in vari settori come la finanza, la gestione della catena di approvvigionamento, i sistemi di voto e la gestione delle identità digitali.

Le blockchain possono essere pubbliche o private. Le blockchain pubbliche sono aperte a tutti e sono gestite da una rete decentralizzata di nodi. Le blockchain private, invece, sono tipicamente utilizzate dalle organizzazioni e sono aperte solo ai partecipanti autorizzati. [3]

1.1 Algoritmi di consenso

L'algoritmo di consenso è un componente fondamentale della tecnologia blockchain e serve come meccanismo per raggiungere un accordo tra i nodi sullo stato. In pratica permette alla rete di concordare su un'unica versione della blockchain, nonostante l'assenza di un'autorità centrale.

L'algoritmo si occupa di assicurare che solo le transazioni valide, che seguono le regole del protocollo, siano incluse nella blockchain.

Inoltre, si occupa di distribuire il controllo e l'autorità decisionale tra i nodi della rete. Questo contribuisce alla robustezza della blockchain, poiché non esiste un singolo punto di guasto che possa compromettere l'intero sistema.

L'algoritmo si occupa anche di fornire un meccanismo per la creazione di nuovi blocchi e della loro aggiunta. Questo processo di solito prevede la selezione di un nodo che propone il nuovo blocco. Viene poi gestita la convalida e l'approvazione del suo contenuto da parte degli altri nodi della rete.

Infine, protegge la blockchain da vari tipi di attacchi, come gli attacchi Sybil, gli attacchi al 51% o il selfish mining, rendendo difficile o economicamente non conveniente per gli attori malintenzionati ottenere il controllo della rete.

Alcuni popolari algoritmi di consenso utilizzati nelle reti blockchain sono Proof of Work, Proof of Stake e Proof of Authority. [4]

1.1.1 Proof of Work (PoW)

Il Proof of Work (PoW) è un algoritmo di consenso utilizzato nelle reti blockchain per convalidare le transazioni e aggiungere nuovi blocchi alla catena. In PoW, i nodi minatori competono per risolvere complessi problemi matematici e il primo minatore che risolve il problema aggiunge un nuovo blocco alla catena e riceve una ricompensa sotto forma di criptovaluta. La difficoltà del problema matematico viene regolata per mantenere un tempo di blocco costante ed evitare che la rete venga sommersa da nuovi blocchi. PoW è considerato un algoritmo di consenso sicuro e affidabile perché richiede ai minatori di spendere risorse computazionali per convalidare le transazioni, rendendo difficile per gli attaccanti manipolare la rete. Tuttavia, è anche criticato per il suo elevato consumo energetico e per i problemi di scalabilità. [4]

1.1.2 Proof of Stake (PoS)

Il Proof of Stake (PoS) è un algoritmo di consenso utilizzato nelle reti blockchain per convalidare le transazioni e aggiungere nuovi blocchi alla catena. In PoS, i validatori sono scelti per creare nuovi blocchi in base alla quantità di criptovaluta che possiedono e che sono disposti a "puntare" come garanzia. I validatori sono incentivati a mantenere l'integrità della rete perché rischiano di perdere la criptovaluta che hanno puntato se agiscono in modo malevolo. La selezione dei validatori è tipicamente casuale o basata su un punteggio di reputazione, a seconda dell'implementazione PoS specifica. Il PoS è considerato più efficiente dal punto di vista energetico e più scalabile del PoW perché non richiede ai minatori di spendere risorse computazionali per convalidare le transazioni.

Tuttavia, è anche criticato per il fatto che potenzialmente accentra il potere tra un piccolo numero di convalidatori con quantità significative di criptovalute puntate. [4]

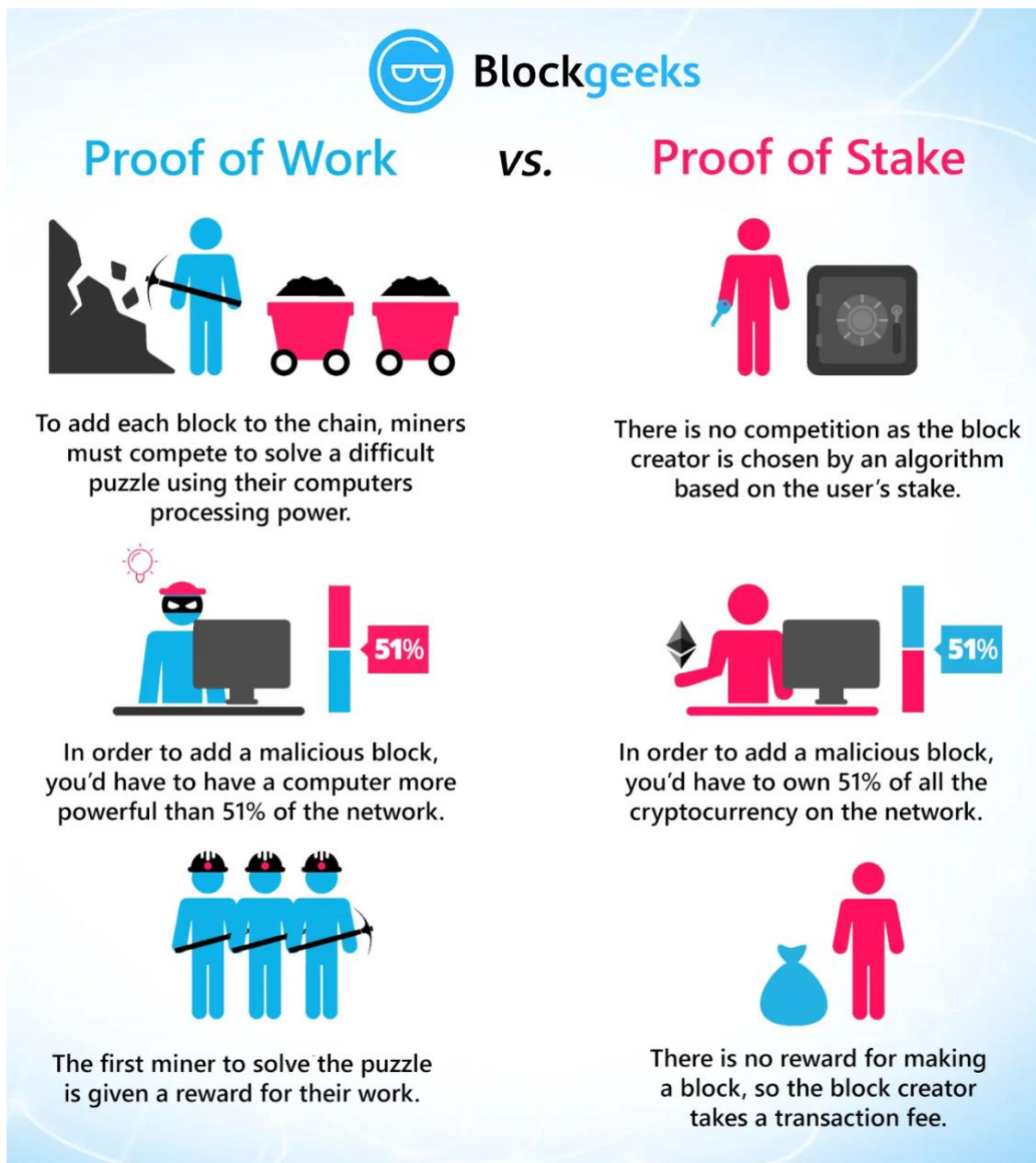


Figura 1 PoW vs PoS [5]

1.1.3 Proof of Authority (PoA)

Proof of Authority (PoA) è un algoritmo di consenso utilizzato nelle reti blockchain per convalidare le transazioni e aggiungere nuovi blocchi alla catena. In PoA, i validatori sono scelti in base alla loro reputazione e ricevono l'autorità di creare nuovi blocchi. I validatori sono in genere selezionati da un'autorità centrale o da un consorzio e la loro identità è nota a tutti i partecipanti alla rete. I validatori sono incentivati a mantenere l'integrità della rete perché è in gioco la loro reputazione. PoA è considerato più efficiente dal punto di vista energetico e più veloce di PoW e PoS, perché non richiede risorse computazionali per convalidare le transazioni. Tuttavia, è anche criticato per essere meno decentralizzato di altri algoritmi di consenso, perché si affida a un piccolo gruppo di validatori con identità note. [6]

2 Tecnologie Blockchain utilizzate

2.1 Ethereum

Ethereum è una piattaforma blockchain decentralizzata e open-source che consente la creazione di smart contract e applicazioni decentralizzate (dApp). È stata annunciata nel 2013 da Vitalik Buterin e lanciata ufficialmente nel 2015.

La blockchain di Ethereum è simile a quella di Bitcoin, in quanto è un libro mastro pubblico che registra tutte le transazioni sulla rete. Tuttavia, Ethereum ha aggiunto il concetto di Smart Contract, ciò consente la creazione di applicazioni decentralizzate, che possono essere costruite sulla blockchain di Ethereum.

Gli Smart Contract consentono di creare un'ampia gamma di applicazioni, dagli scambi decentralizzati alle piattaforme di gioco. Queste applicazioni possono funzionare senza alcuna possibilità di censura, tempi morti, frodi o interferenze di terzi.

Ethereum funziona grazie alla propria criptovaluta, chiamata Ether (ETH), che viene utilizzata per pagare le commissioni di transazione e i servizi di calcolo sulla rete. Queste commissioni sono note come Gas e sono necessarie per eseguire gli smart contract e svolgere altri compiti sulla blockchain di Ethereum.

Un altro aspetto importante di Ethereum è il suo meccanismo di consenso, chiamato Proof-of-Stake (PoS), che consente ai titolari di Ethereum di convalidare le transazioni e guadagnare ricompense, anziché ai minatori. Questo dovrebbe rendere la rete più efficiente dal punto di vista energetico e più sicura rispetto al precedente Proof-of-Work (PoW). [7]

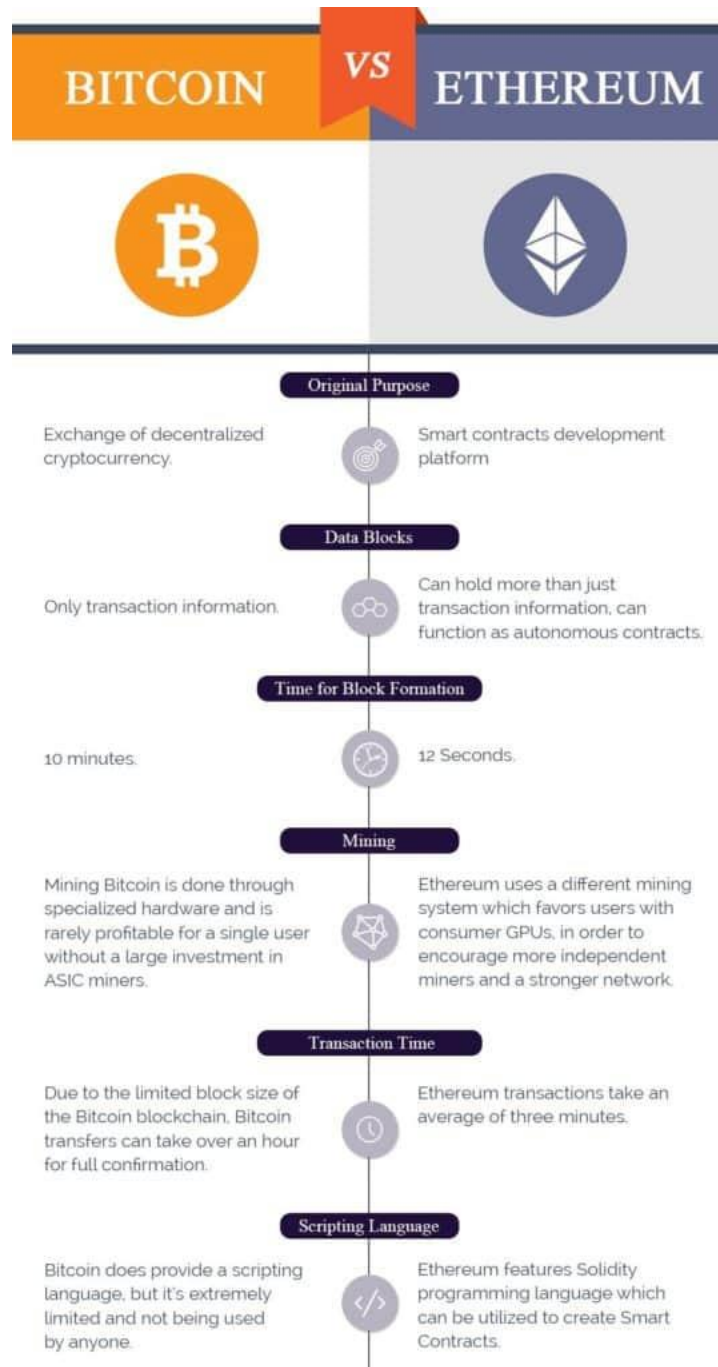


Figura 2 Bitcon vs Ethereum [8]

2.1.1 Gas

Il gas in Ethereum si riferisce al costo di esecuzione di una transazione o di uno smart contract sulla blockchain. È un'unità di misura che rappresenta la quantità di sforzo computazionale richiesto per eseguire una specifica operazione sulla rete.

Il gas è strettamente legato al bytecode di uno smart contract: quando uno smart contract viene compilato per essere caricato nella blockchain, viene generato del bytecode. Il bytecode è il codice macchina di Ethereum, dove ogni byte corrisponde ad una operazione di basso livello da compiere. Ad ogni operazione viene quindi assegnato un “peso” in gas. Così è possibile valutare il peso di ogni operazione ad alto livello sulla blockchain.

Ogni operazione sulla rete, richiede il pagamento di gas da parte del mittente, il gas viene pagato in Ether, la criptovaluta nativa della rete Ethereum. Quando un utente invia una transazione o uno smart contract, include il prezzo del gas, che determina quanto è disposto a pagare per ogni unità di gas. I minatori, che elaborano le transazioni ed eseguono gli smart contract, scelgono quali transazioni includere in un blocco in base al prezzo del gas offerto. Prezzi del gas più elevati comportano tempi di elaborazione più rapidi, in quanto i minatori sono incentivati a dare priorità alle transazioni con tariffe più elevate.

Il sistema del gas fornisce un meccanismo per limitare l'utilizzo delle risorse sulla rete e per evitare che persone malintenzionate sovraccaricano la rete con un numero eccessivo di operazioni ad alto consumo di risorse. L'uso del gas garantisce inoltre che tutti i partecipanti alla rete siano interessati al suo funzionamento e alla sua sicurezza, in quanto devono pagare per le risorse che consumano. [9]

2.1.2 Smart Contracts

Uno smart contract è un programma informatico che esegue automaticamente i termini di un contratto quando vengono soddisfatte determinate condizioni. Viene memorizzato ed eseguito su una rete blockchain. Questo rende gli smart contract trasparenti, sicuri e a prova di manomissione, poiché il codice e tutte le transazioni eseguite sono pubblicamente accessibili e registrate su più nodi della rete. L'uso degli smart contract elimina la necessità di intermediari e consente di stabilire la fiducia tra le parti attraverso il codice, anziché affidarsi a un'autorità centrale. L'esecuzione di uno smart contract è innescata da un evento specifico, come la ricezione di un pagamento, il risultato è determinato da regole e condizioni predefinite scritte nel codice. Inoltre, fornisce la possibilità di emettere eventi per tenere traccia delle operazioni effettuate, questi eventi possono essere ascoltati in real-time o recuperati in differita. [10]

2.1.3 Dapp

Una DApp, o applicazione decentralizzata, è un'applicazione software che gira su una rete decentralizzata e utilizza la tecnologia blockchain. A differenza delle applicazioni tradizionali,

che si affidano a server e intermediari centralizzati per elaborare le transazioni e gestire i dati, le DApp sono costruite su reti decentralizzate dove i dati e le transazioni sono registrati su un libro mastro condiviso e gestito da una rete di nodi.

Le DApp hanno tipicamente le seguenti caratteristiche:

- sono costruite su una rete decentralizzata, senza autorità centrali o intermediari che controllano i dati o le transazioni.
- il codice è open source e pubblicamente accessibile, consentendo a chiunque di contribuire allo sviluppo e al miglioramento dell'applicazione.
- possibilità di avere una propria criptovaluta o un token che viene utilizzato per incentivare gli utenti a partecipare alla rete e contribuire alla sua sicurezza.

[11]

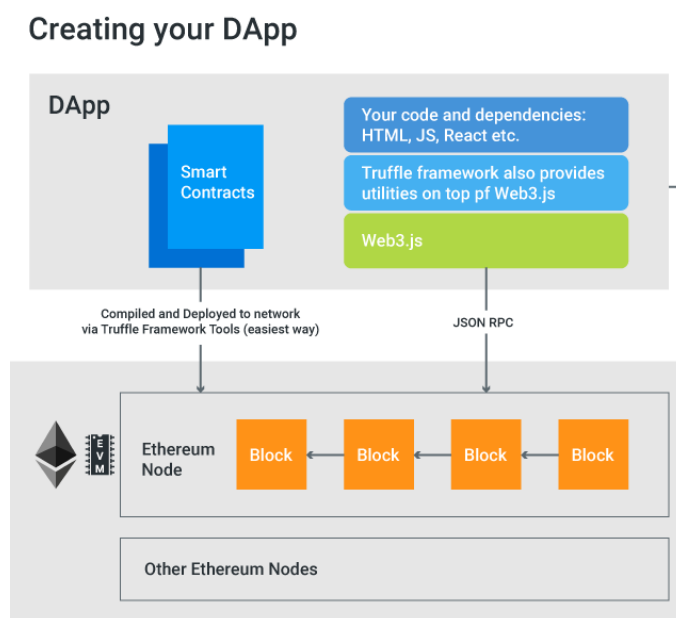


Figura 3 Sviluppo di una dApp [12]

Per lo sviluppo di una dApp è necessario scrivere Smart contracts, compilarli e farne il deploy sulla rete. Per fare ciò i software più comuni sono Truffle Framework e Remix IDE. Per interagire con la blockchain è necessario un front-end, tipicamente sviluppato con un framework di javascript che utilizzerà la libreria web3.js.

L'utente per utilizzare la dApp necessiterà di utilizzare il proprio wallet, per fare ciò si utilizza un portafoglio digitale come Metamask.

2.2 IPFS

IPFS è una rete decentralizzata e peer-to-peer utilizzata per il caricamento e la condivisione dei file. Per utilizzare IPFS è necessario creare una rete di nodi che potranno, quindi, comunicare tra di loro condividendo e propagando i file caricati.

In IPFS i file sono identificati da un hash crittografico anziché dal nome/path, noto come Content Identifier (CID). Questo significa che per recuperare un file sarà necessario conoscere il suo hash.

IPFS utilizza la struttura dati MerkleDAG (Directed Acyclic Graph), dove ogni nodo del grafo è un “content-addressed block” e i collegamenti tra i nodi sono dei CID, questa struttura permette il versioning dei dati, dove la nuova versione del file ha sempre un collegamento alla precedente.

IPFS utilizza una tabella hash distribuita (DHT) che consente ai nodi di trovarsi l'un l'altro e di localizzare i file, inoltre, include il Block Exchange Protocol (Bitswap).

Bitswap è un protocollo ispirato a BitSwap di BitTorrent, per lo scambio di blocchi di dati tra i peer. I nodi richiedono i blocchi di cui hanno bisogno e i peer che hanno i blocchi richiesti li inviano. Ciò consente di distribuire e replicare in modo efficiente i dati sulla rete. [13]

2.2.1 IPFS vs HTTP

IPFS e HTTP sono entrambi protocolli utilizzati per il trasferimento e la gestione dei file, ma si differenziano per diversi aspetti fondamentali.

Il funzionamento dei due protocolli è il seguente:

- HTTP è utilizzato per la trasmissione di dati sul Web tramite una architettura client-server. Questa architettura richiede la gestione dei server da parte di un'autorità centrale, comportando possibili problemi di scalabilità e/o la possibilità di censure.
- IPFS è un file system decentralizzato che consente l'archiviazione e il recupero dei file in una rete peer-to-peer. Non si affida quindi ad un unico nodo centrale, bensì ogni nodo della rete contribuisce alla distribuzione ed al mantenimento dei file.

HTTP, usando un protocollo basato sulla connessione ad un nodo centrale, può essere vittima di problemi come collegamenti interrotti in caso di spostamento di un file.

Questo problema viene risolto in IPFS: la gestione di file tramite hash rimuove completamente questo problema, in quanto l'hash è univoco e prescinde dalla posizione.

Anche in caso di aggiornamento/eliminazione del file, IPFS è migliore: quando si aggiorna un file, la vecchia versione non viene cancellata. Viene assegnato un nuovo HASH al nuovo file. Questo consente di avere tutte le versioni del file(versioning) e di non avere dead-link, dato che il vecchio file esiste ancora.

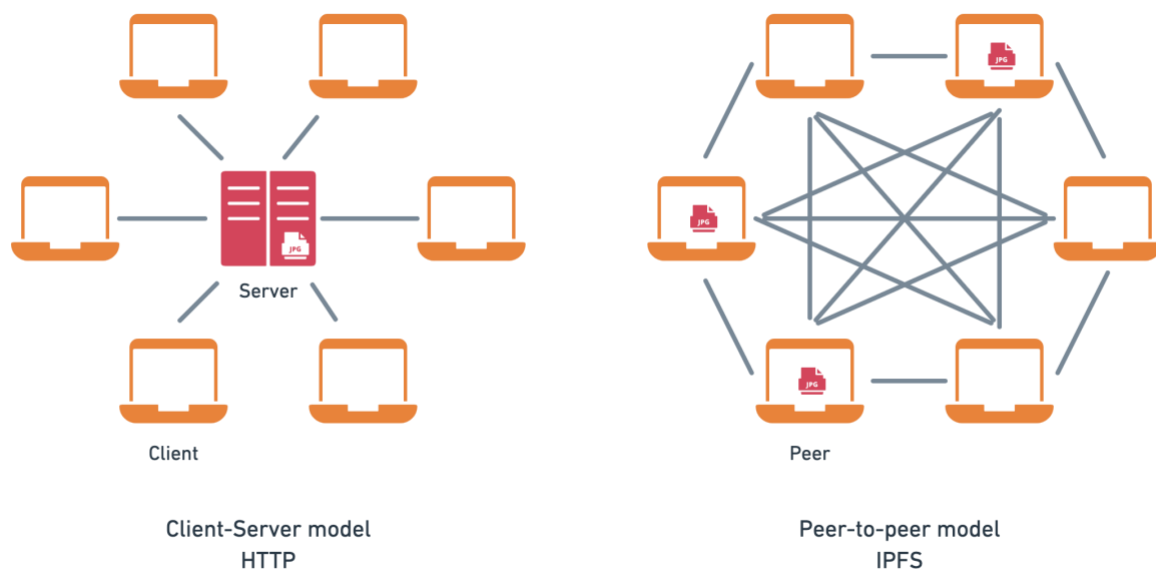


Figura 4 HTTP vs IPFS [14]

2.2.2 Swarm Key

La Swarm key è un componente di IPFS, essa è una chiave crittografica utilizzata per criptare e decriptare i file. Queste chiavi sono utilizzate per proteggere i file e garantire che solo chi ne è in possesso possa accedervi.

Un file sulla rete IPFS viene diviso in pezzi i quali hanno un unico hash identificativo. Ogni pezzo verrà crittografato usando la swarm key, cosicché possa essere letto solo da chi possiede la chiave.

Per dare l'accesso ai file della rete, quindi, sarà necessario condividere la chiave con tutti i nodi che dovranno immettersi nella rete.

2.2.3 Kubo IPFS

Kubo è stata la prima implementazione di IPFS e, ad oggi, è la più utilizzata.

Esso consente l'utilizzo dell'IPFS, standard Web3, tramite lo standard Web2 HTTP.

L'implementazione di Kubo IPFS utilizza Go come linguaggio di programmazione.

Le sue caratteristiche principali sono: permettere di eseguire un nodo IPFS come servizio di rete, una comoda interfaccia a riga di comando per gestire i nodi IPFS, il gateway HTTP locale da Web2 a Web3, le API HTTP RPC per accedere e controllare il daemon, la Webgui interna di IPFS che può essere usata per gestire i nodi Kubo.

Inoltre, include libp2p una libreria di rete modulare che fornisce funzionalità di comunicazione, rilevamento e crittografia del trasporto peer-to-peer, utilizzata per facilitare lo sviluppo di applicazioni peer-to-peer. [15]

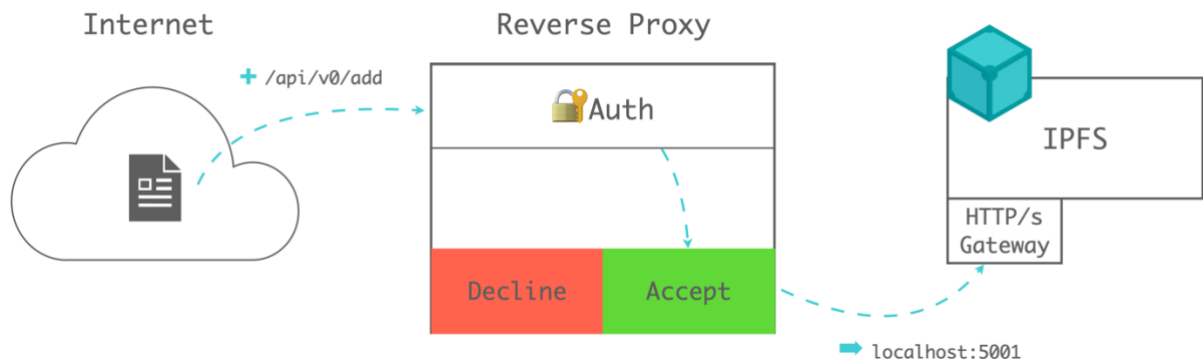


Figura 5 IPFS API [16]

2.3 Metamask

MetaMask è un popolare portafoglio di criptovalute che consente agli utenti di archiviare, gestire e scambiare in modo sicuro i propri beni digitali.

È un'estensione del browser che può essere aggiunta ai browser web più diffusi, come Chrome, Firefox e Brave. Con MetaMask, gli utenti possono connettersi facilmente alle applicazioni decentralizzate (dApp) e interagire con le reti blockchain come Ethereum. Fornisce un'interfaccia facile da usare per gestire più portafogli di criptovalute, effettuare transazioni e interagire con le dApp. MetaMask consente inoltre agli utenti di memorizzare e gestire token non fungibili (NFT), che hanno guadagnato popolarità negli ultimi anni. Fornisce funzioni di sicurezza avanzate come l'autenticazione a due fattori e il backup della frase di seed, garantendo la sicurezza dei wallet degli utenti. Nel complesso, MetaMask è diventato una scelta popolare per gli utenti che desiderano un portafoglio di criptovalute affidabile e facile da usare. [17]

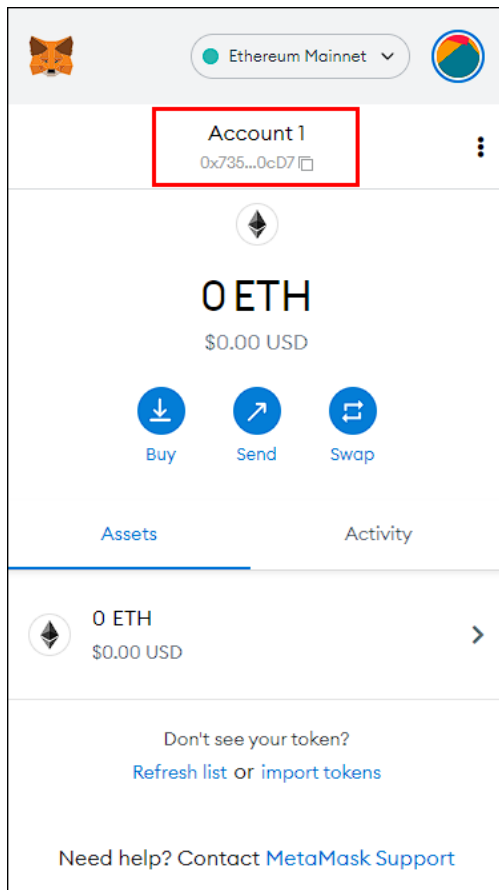


Figura 6 Metamask [18]

2.4 Remix IDE

Remix è un ambiente di sviluppo basato sul web che consente di scrivere, distribuire e testare smart contract sulla blockchain e viene usato combinato al linguaggio di programmazione Solidity.

L'editor di codice, come un comune IDE, fornisce l'evidenziazione della sintassi, il completamento del codice e altre funzioni che aiutano gli sviluppatori a scrivere il codice in modo più efficiente.

Remix si integra anche con i più diffusi strumenti e reti di sviluppo Ethereum, come Ganache, Truffle e Rinkeby, rendendo più facile per gli sviluppatori utilizzare Remix nel loro flusso di lavoro esistente. [19]

2.5 Protocollo JSON-RPC

JSON-RPC è un protocollo di chiamata di procedura remota (remote procedure call) codificato in JSON, un formato molto utilizzato sul web perché consente di codificare i dati in formato facilmente leggibile e trasferibile, per esempio, tra client e server.

JSON-RPC consente di inviare richieste a un server che implementa questo protocollo: i parametri di input possono essere passati al metodo remoto come array o oggetti, mentre il metodo stesso può restituire anche più valori di output.

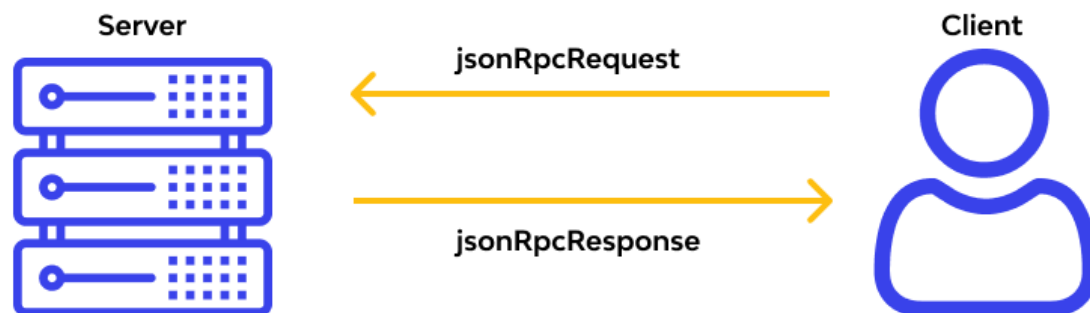
I parametri sono serializzati utilizzando JSON. Una richiesta, che è una chiamata a un metodo specifico fornito da un sistema remoto, deve contenere tre proprietà:

- id: identificativo della richiesta, costituito da un valore di qualsiasi tipo, utilizzato per abbinare la risposta alla richiesta a cui sta rispondendo;
- method: una stringa con il nome del metodo da invocare;
- params: un oggetto o una matrice di valori da passare come parametri al metodo definito.

Il destinatario della richiesta risponde a tutte le richieste ricevute: una risposta deve contenere le seguenti proprietà:

- id: l'id della richiesta a cui sta rispondendo;
- result: i dati restituiti dal metodo invocato (questo elemento è formattato come oggetto JSON-stat: se si è verificato un errore durante il richiamo del metodo, questo valore deve essere null);
- error: un codice di errore specificato se si è verificato un errore nel richiamo del metodo, altrimenti null.

Poiché ci sono situazioni in cui non è necessaria o addirittura desiderata la risposta, sono state introdotte le notifiche: una notifica è simile a una richiesta, fatta eccezione per l'id, che non è necessario perché non verrà restituita alcuna risposta; in questo caso la proprietà id dovrebbe essere omessa (versione 2.0 del protocollo) o essere nulla (versione 1.0 del protocollo).



Example:

```
request:
{"method":"my_method","params":[1,2,3],"id":"my_id"}
response:
{"result":"my_result","error":null,"id":"my_id"}
```

Figura 7 JSON-RPC Request [20]

2.5.1 JSON-RPC in Ethereum

JSON-RPC è il protocollo di comunicazione nativo per i nodi blockchain di Ethereum, che viene utilizzato per permettere il collegamento e il trasferimento dati tra i vari client utenti e i nodi della blockchain.

JSON-RPC contiene una suite di comandi di basso livello che possono essere inviati a un nodo, tramite protocolli di comunicazione, tra cui HTTPS e WebSocket. Le comunicazioni JSON-RPC sono complesse e si occupano dei payload codificati con prefisso di lunghezza ricorsiva binaria (RLP) generati secondo un'interfaccia binaria di applicazione (ABI) dei metodi su uno smart contract. Per firmare e inviare una transazione sono necessarie molte chiamate individuali, incluso il polling asincrono fino a quando le transazioni non vengono estratte in blocchi. Gli sviluppatori di applicazioni decentralizzate (Dapp) raramente programmano direttamente le stesse API JSON-RPC, ma sfruttano librerie client di alto livello specifiche del linguaggio di programmazione utilizzato.

Le dimensioni di queste librerie sono abbastanza ridotte da poter essere integrate facilmente in un browser Web o dispositivo mobile, per consentire a un wallet di essere associato con l'applicazione dell'utente finale.

La libreria ufficiale, scritta in JavaScript è web3.js; all'interno della documentazione è possibile trovare i riferimenti a librerie analoghe sviluppate in altri linguaggi.

2.6 EAHP - Ethereum Access HTTP Proxy

L'applicazione EAHP consente di esporre su Internet il protocollo JSON-RPC di un nodo Ethereum di una blockchain privata, sfruttando un token aggiuntivo di connessione per verificare l'identità del chiamante e stabilire le politiche di accesso.

Progettando di erogare servizi di Blockchain PAAS, emerge la necessità di offrire degli strumenti per la gestione sicura degli accessi alle risorse, cercando di alterare il meno possibile la natura permissionless di Ethereum.

Il processo di analisi rispetto alla tematica della gestione degli accessi ha portato alla proposta qui descritta.

2.6.1 Protocolli Ethereum di rete

La blockchain Ethereum utilizza tre protocolli differenti:

- Kademlia, protocollo p2p per il discovery dei nodi: permette lo scambio di informazioni in chiaro, in quanto non espone dati interni della blockchain ma soltanto dati sulla disponibilità dei nodi;
- RLPx, protocollo p2p per lo scambio tra i nodi dei dati di blocchi e transazioni: cifrato e gestito in autonomia dai nodi;
- JSON-RPC, protocollo applicativo client/server per l'interrogazione e l'invocazione dei servizi esposti da un nodo della blockchain: tale protocollo delega al livello di trasporto le politiche riguardanti la sicurezza.

Nel caso di fruizione di una blockchain PAAS, l'unico protocollo esposto esternamente è quello JSON-RPC, mentre gli altri vengono utilizzati all'interno della VPC della blockchain.

Attualmente, per rendere sicura la comunicazione JSON-RPC su HTTP, risultano applicate le seguenti strategie:

- filtro IP per il controllo degli accessi;
- autenticazione tramite HTTP Basic Auth;
- reverse proxy con esposizione HTTPS.

Applicando le strategie sopra descritte rimangono aperte le seguenti criticità:

- il filtro IP risulta essere una strategia poco flessibile e troppo generica;
- la gestione di autenticazioni di tipo HTTP Basic Auth implica ulteriori politiche di accesso e l'utilizzo di strumenti aggiuntivi per la loro gestione.

2.6.2 Soluzione aziendale

La soluzione che l'azienda utilizza per l'autenticazione è la seguente:

- utilizzare un reverse proxy con esposizione HTTPS al fine di stabilire un canale di comunicazione cifrato;
- aggiungere all'indirizzo HTTPS di esposizione della blockchain un contesto autorizzativo, detto *signed payload*, ovvero una stringa contenente le credenziali, opportunamente codificate, per accedere alla blockchain;
- costruire le credenziali di accesso tramite una firma, realizzata con le tecnologie proprie della blockchain, utilizzando il wallet del chiamante;
- realizzare un reverse-proxy che intercetta la richiesta, verifica la firma, estrae l'indirizzo del chiamante e verifica, tramite strategie predefinite, se il chiamante è abilitato all'accesso oppure no. [21]

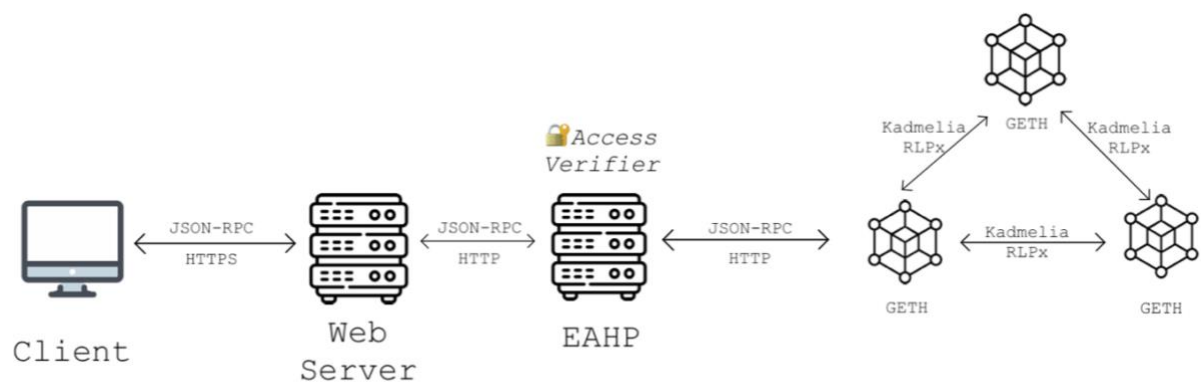


Figura 8 Esempio di utilizzo reale

3 Tecnologie di supporto

3.1 REST API

REST (Representational State Transfer) API è uno stile architetturale del software che definisce un insieme di vincoli da utilizzare nella creazione di servizi web.

Esse utilizzano un modello client-server, in cui il client invia una richiesta al server e il server restituisce una risposta. La risposta può essere sotto forma di dati, come un documento JSON o XML, oppure può ritornare i classici valori HTML di successo/errore (200, 403, 404, 500, ...)

Le API REST sono *stateless*: questo comporta che il server non memorizza alcuna informazione sul client. Quindi, in caso di necessità, sarà necessario creare una nostra implementazione, ad esempio tramite un token.

Le API REST utilizzano metodi HTTP, come ad esempio GET, POST, PUT e DELETE, per indicare l'azione da eseguire sulla risorsa richiesta.

3.2 RPC API

L'API Remote Procedure Call (RPC) è un protocollo di comunicazione utilizzato per effettuare richieste da remoto su una rete. L'obiettivo principale delle API RPC è quello di consentire a un client di chiamare una funzione su un server, come se fosse una funzione locale, senza doversi preoccupare dei dettagli della rete sottostante.

In un'API RPC, il client invia una richiesta al server, specificando il nome della procedura o della funzione da eseguire, insieme ai vari argomenti necessari. Il server esegue poi la procedura richiesta e invia il risultato al client.

Le API RPC sono utilizzate spesso nei sistemi distribuiti come il nostro, dove i diversi componenti si trovano su diversi dispositivi in rete.

3.3 Java Spring

Java Spring è un framework open-source per la costruzione di applicazioni web scalabili e robuste.

Gestisce il ciclo di vita degli oggetti dell'applicazione, fornendo servizi chiave come l'aggiunta delle dipendenze e la gestione delle transazioni. Questi servizi consentono di scrivere codice modulare e testabile, che può essere facilmente esteso e personalizzato durante l'evoluzione dell'applicazione.

Consente, inoltre, di utilizzare i migliori strumenti e framework per le diverse parti dell'applicazione, pur mantenendo un'architettura e un modello di sviluppo coesi.

Uno dei vantaggi principali di Java Spring è il suo vasto ecosistema di plugin ed estensioni, che facilita l'aggiunta di nuove feature e funzionalità all'applicazione: comprende tutto il necessario, dalle librerie ai framework di terze parti, agli strumenti per costruire e distribuire le applicazioni nel cloud. [22]

3.4 Vue.js

Vue.js è un framework JavaScript che fornisce agli sviluppatori un insieme di strumenti per la creazione di interfacce utente dinamiche e applicazioni a pagina singola.

È un framework leggero e versatile che consente agli sviluppatori di creare componenti riutilizzabili, facilitando la gestione di interfacce utente complesse.

Può essere utilizzato con altre librerie o progetti esistenti, ha una sintassi semplice e una documentazione chiara, che ne facilita l'apprendimento.

Inoltre, offre la possibilità di avere il routing di pagine diverse grazie a Vue Router, dall'interfaccia l'applicazione risulterà sempre single page, il che rende tutto più gradevole.
[23]

4 Progetto

4.1 Architettura

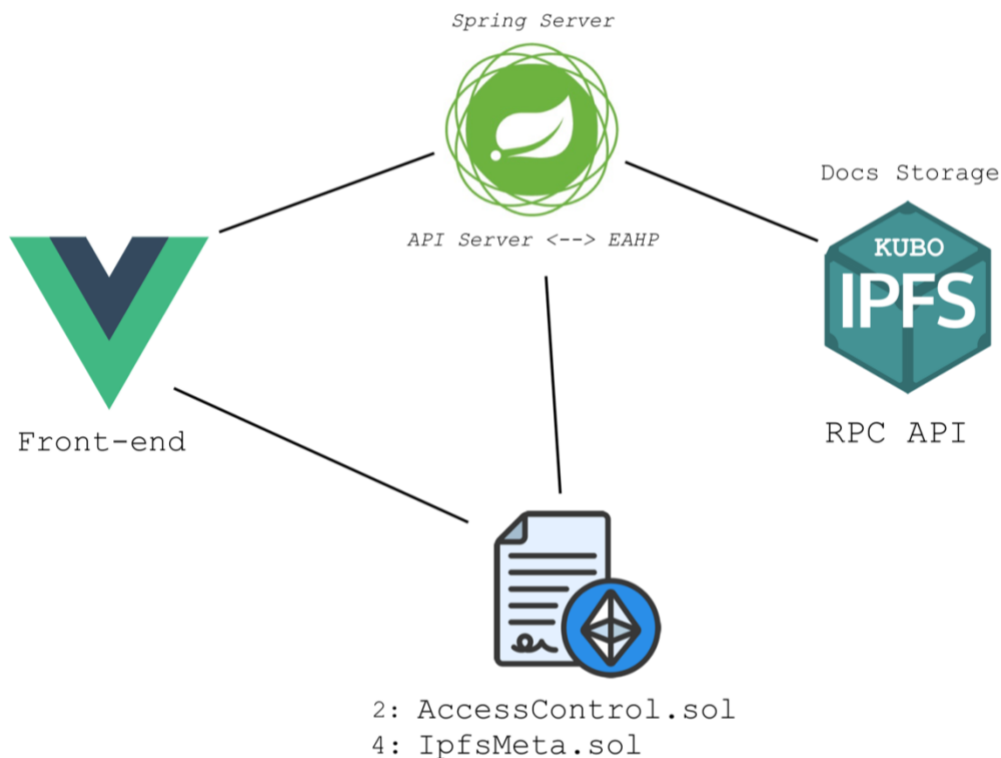


Figura 9 Workflow

Vista la necessità di avere un'interfaccia utente, è stata implementata una pagina web che permettesse di compiere agevolmente le operazioni richieste: la generazione del payload di accesso, upload, download e lo storico dei file caricati. Per lo sviluppo del front-end è stato scelto un framework di JavaScript, Vue.js, così da poter utilizzare la libreria web3.js per interfacciarsi con la blockchain.

Per gestire le richieste, utilizzeremo un server che espone delle REST API, che servirà da tramite tra il Frontend e le API dello Docs Storage, in modo da creare delle procedure ad hoc per ogni richiesta. Per svilupparlo si è scelto di utilizzare lo standard aziendale, Java Spring.

Per fare lo storing dei documenti in comune ci serviremo di una piattaforma decentralizzata, optando per la più diffusa, ovvero IPFS. La sua scelta è stata dettata dalle sue peculiarità già evidenziate, tra le quali che, rispetto a HTTP, fornisce un collegamento al file immutabile, un alto grado di persistenza e versioning del file.

Essendo in una rete aziendale, servirà rendere la rete IPFS privata: per fare questo utilizzeremo l'implementazione Kubo IPFS che, tramite le swarm key, permette l'accesso ai file solo a chi possiede quest'ultima.

Inoltre, è stato scelto Kubo sia perché offre una ampia scelta di HTTP API che verranno utilizzate per mettere di interfacciarsi con la rete IPFS, sia per l'affidabilità fornita essendo l'implementazione più diffusa.

È stato richiesto che questo modulo di Andromeda fosse accessibile solo a chi autorizzato. Per fare ciò si è seguito lo standard aziendale, in cui l'autenticazione è regolata attraverso un *payload* generato dal front-end che l'applicativo EAHP sviluppato in Java Spring, verificherà.

Inoltre, l'EAHP si occuperà di chiamare uno Smart Contract già utilizzato dall'azienda che chiameremo *AccessControl*, che permetterà di verificare che l'utente sia abilitato.

Questo sistema di autenticazione è stato appositamente studiato, come già spiegato, per esporre il protocollo JSON-RPC di un nodo di una blockchain privata, risolvere le criticità della comunicazione JSON-RPC su HTTP e gestire i permessi di un'utente direttamente "on-chain" senza doverli salvare su un database.

L'azienda, vedendo in fase di progettazione che una volta salvato il file per lo standard di IPFS veniva solo salvato l'hash del file, ha richiesto di salvare anche i metadati dei file. È stato deciso di salvare questi dati sempre on-chain per questioni di convenienza e sicurezza, anziché salvare i metadati su un database.

Per questa necessità abbiamo deciso di memorizzare i metadati sulla blockchain tramite uno Smart Contract che verrà chiamato *IPFSMeta*, quest'ultimo verrà sempre chiamato dal front-end.

Utilizzando questi due Smart Contracts rispetto a un database garantisce trasparenza e sicurezza, poiché il codice e tutte le transazioni eseguite sono accessibili e registrate su più nodi della rete.

A scopo di test l'EAHP per l'autenticazione e le Rest API verranno implementati sullo stesso server.

4.1.1 Upload file

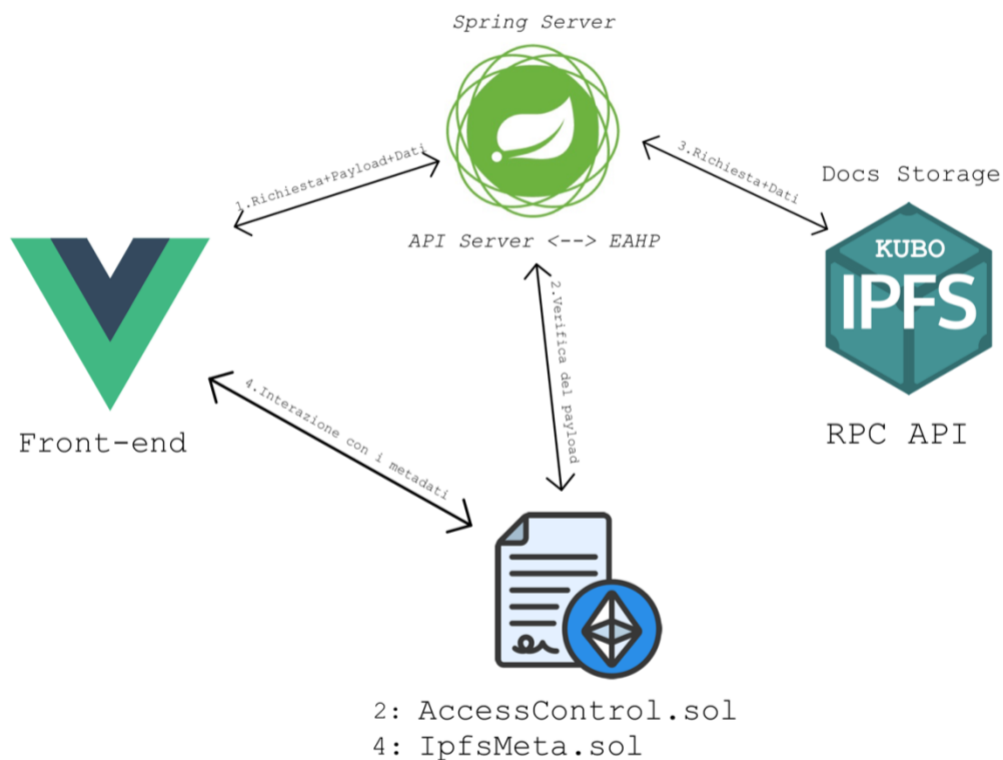


Figura 10 Upload File

Dal front-end viene fatta una richiesta contenente il payload ed il file, verso lo Spring Server.

Lo Spring Server, all'arrivo della richiesta, verifica il payload chiamando il contratto *AccessControl*. Superato questo controllo, il server esegue la richiesta di upload sulla rete IPFS. Una volta caricato il file sulla rete verrà restituito l'hash associato, che verrà rispedito al Spring Server e conseguentemente al front-end.

Il front-end utilizzerà l'hash per interagire con il contratto *IpfsMeta*, così da salvare i metadati associati al file.

Il front-end visualizzerà, infine, l'esito di tutte queste operazioni.

4.1.2 Retrieve file

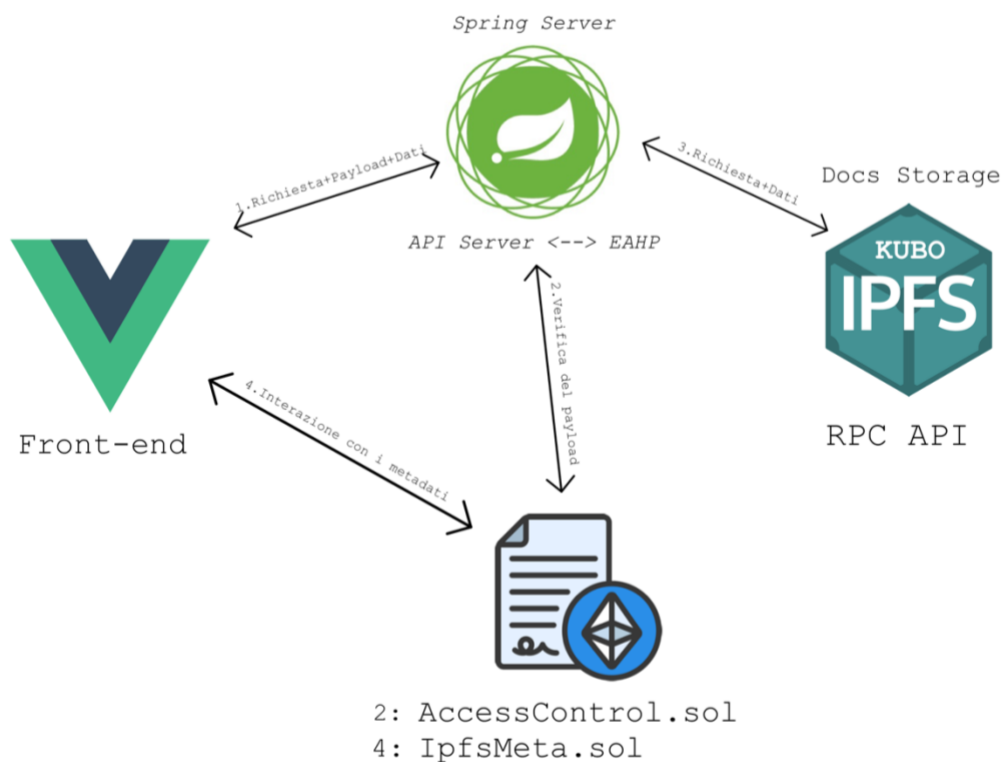


Figura 11 Retrieve File

Dal front-end viene fatta una richiesta contenente il payload e l'hash del file, verso lo Spring Server.

Lo Spring Server, all'arrivo della richiesta, verifica il payload chiamando il contratto *AccessControl*.

Superato questo controllo, lo Spring Server esegue una richiesta alla rete IPFS, contenente l'hash del file. Una volta che il file viene recuperato sulla rete IPFS, verrà rispedito allo Spring Server e conseguentemente al front-end.

Il file caricato sulla rete IPFS non contiene i metadati: per recuperarli sarà necessario eseguire una chiamata al contratto *IpfsMeta*, specificando l'hash del file.

I metadati ottenuti verranno quindi associati al file durante la costruzione del link di download.

Il front-end visualizzerà l'esito di tutte queste operazioni.

4.1.3 Lista caricamenti dei file

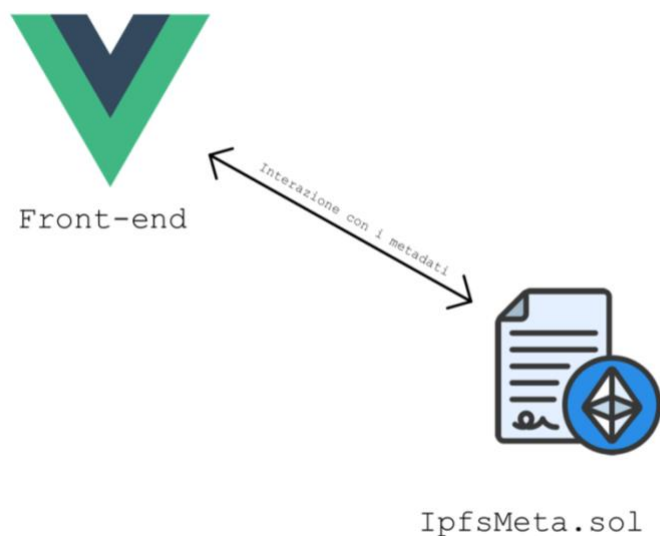


Figura 12 Event Listener

Per recuperare i file caricati, il front-end ascolta gli ultimi eventi del contratto *IpfsMeta*.

Questa chiamata restituirà una lista di hash che verrà poi utilizzata per recuperare i metadati associati ai file, tramite chiamate a *IpfsMeta*.

Il front-end, infine, comporrà una tabella dove ogni riga sarà composta dall'hash del file ed i metadati associati.

4.2 Sicurezza

Per implementare la soluzione aziendale utilizzo la seguente procedura in javascript per il calcolo del *signed payload*, utilizzando la codifica in base 36 per rendere il payload più corto:

1. selezionare il numero di versione dell'algoritmo da utilizzare a (normalmente 1) e codificarlo in base 36 ottenendo la stringa A;
2. selezionare il valore esadecimale dell'indirizzo del wallet chiamante b e codificarlo in base 36 ottenendo la stringa B;
3. selezionare una data di inizio validità c, espressa nel formato numerico Milliseconds since Unix Epoch, e codificarla in base 36 ottenendo la stringa C;
4. selezionare una data di fine validità d, espressa nel formato numerico Milliseconds since Unix Epoch, e codificarla in base 36 ottenendo la stringa D;
5. costruire la stringa *payload* utilizzando i valori precedentemente codificati, separati dal carattere "punto"; il risultato sarà A.B.C.D;
6. firmare il *payload* con la chiave privata del wallet, ottenendo la firma signature;
7. selezionare il valore esadecimale r della firma signature e codificarlo in base 36 ottenendo la stringa R;
8. selezionare il valore esadecimale s della firma signature e codificarlo in base 36 ottenendo la stringa S;

9. selezionare il valore esadecimale *v* della firma signature e codificarlo in base 36 ottenendo la stringa *V*;
10. costruire la stringa signed payload utilizzando la stringa payload più *R*, *S* e *V* separati dal carattere "punto"; il risultato sarà payload.R.S.V.

Questo *payload* sarà verificato dall'EAHP server-side.

4.3 Rete IPFS Privata

Per unire la comodità di IPFS alla necessità di lavorare su una rete aziendale, creeremo una rete IPFS Privata.

Per creare una rete IPFS privata ho utilizzato Kubo IPFS.

Kubo fornisce molte opzioni, noi utilizzeremo oltre ai comandi basi la possibilità di condividere un swarm key tra i nodi. Utilizzeremo questa feature per creare una rete privata che a scopo di test faremo di tre nodi.

Viene inizializzato ogni nodo della rete con il comando "ipfs init":

```
IPFS_PATH=~/.ipfs ipfs init
```

Viene creata la swarm key da condividere con tutti nodi nel primo nodo e successivamente viene inserita nella cartella di ogni nodo.

Con il seguente comando viene generata:

```
./go/bin/ipfs-swarm-key-gen > ~/.ipfs/swarm.key
```

Viene rimossa la connessione alla net globale di IPFS in ogni nodo con questo comando:

```
IPFS_PATH=~/.ipfs ipfs bootstrap rm --all
```

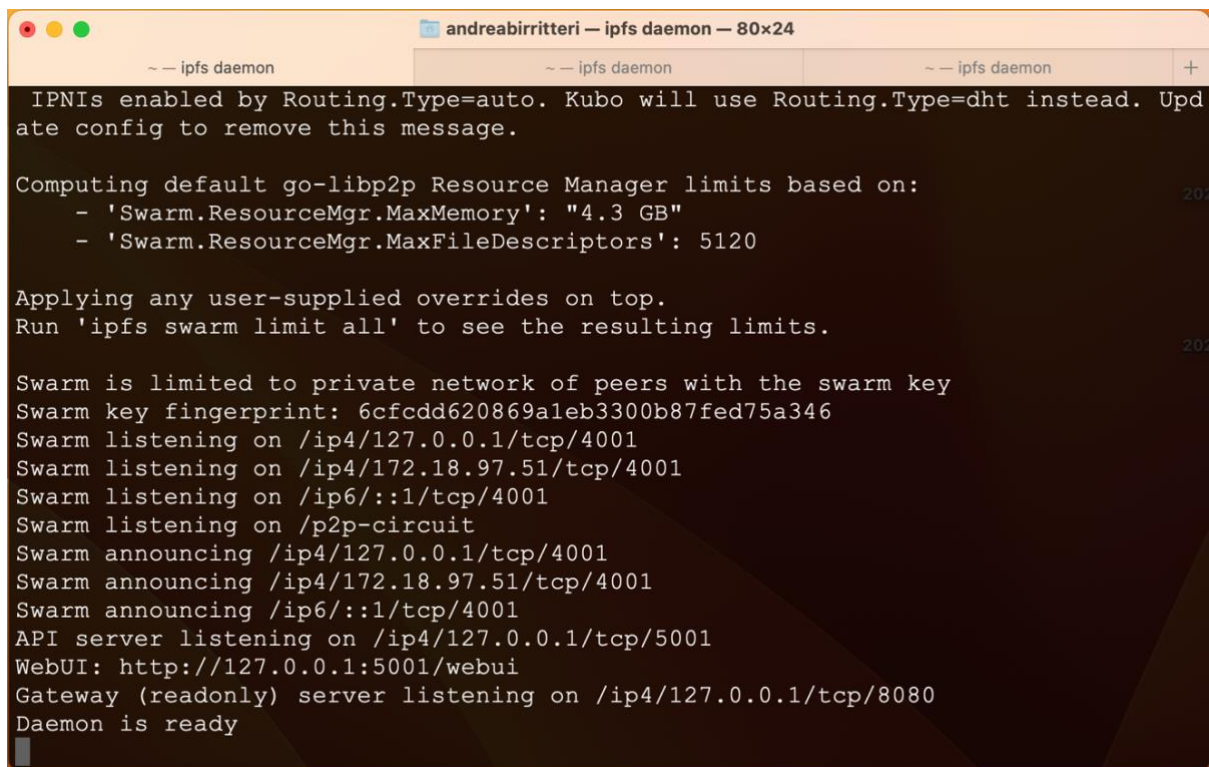
Viene aggiunto a tutti i nodi l'indirizzo del nodo di bootstrap con questo comando:

```
IPFS_PATH=~/.ipfs ipfs bootstrap add /ip4/<ip address of bootnode>/tcp/4001/ipfs/<peer identity hash of bootnode>
```

Viene avviato ogni nodo forzando la rete privata con questo comando:

```
export LIBP2P_FORCE_PNET=1 && IPFS_PATH=~/.ipfs ipfs daemon
```

cambiando la IPFS_PATH a seconda del nodo. [24]



```
~ -- ipfs daemon
IPNIs enabled by Routing.Type=auto. Kubo will use Routing.Type=dht instead. Update config to remove this message.

Computing default go-libp2p Resource Manager limits based on:
- 'Swarm.ResourceMgr.MaxMemory': "4.3 GB"
- 'Swarm.ResourceMgr.MaxFileDescriptors': 5120

Applying any user-supplied overrides on top.
Run 'ipfs swarm limit all' to see the resulting limits.

Swarm is limited to private network of peers with the swarm key
Swarm key fingerprint: 6cfcdd620869a1eb3300b87fed75a346
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/172.18.97.51/tcp/4001
Swarm listening on /ip6:::1/tcp/4001
Swarm listening on /p2p-circuit
Swarm announcing /ip4/127.0.0.1/tcp/4001
Swarm announcing /ip4/172.18.97.51/tcp/4001
Swarm announcing /ip6:::1/tcp/4001
API server listening on /ip4/127.0.0.1/tcp/5001
WebUI: http://127.0.0.1:5001/webui
Gateway (readonly) server listening on /ip4/127.0.0.1/tcp/8080
Daemon is ready
```

Figura 13 Esempio da terminale con tre nodi

4.4 Smart Contracts

Per salvare i metadati del file e controllare l'accesso alla blockchain è stato pensato di implementare i seguenti smart contracts:

- IpfsMetadata
- MyAccessControl

4.4.1 IpfsMetadata.sol

Come già detto questo Smart contract assocerà a l'hash del file i suoi corrispettivi metadati; quindi, a livello di sviluppo è stato pensato di associare ad ogni hash una struttura dati contenenti i metadati. Per cui per salvare queste coppie di dati è stato pensato di implementare un hash table.

Come prima cosa viene creata una struttura dati che contenga i metadati interessanti del file come segue:

```
struct IpfsMetadata{
    string name;
    string mimeType;
    address owner;
    uint32 size;
    uint256 timestamp;
```



```
bool value;
}
```

Un mapping che associa all' hash del file i suoi metadati:

```
mapping(bytes32 => IpfsMetadata) private metadataMap;
```

Viene anche creato un evento che verrà emesso ogni volta che vengono aggiunti dei metadati a un hash in modo da poter rintracciare le operazioni fatte:

```
event AddIpfsFileEvent(string fileHash);
```

Viene creata una funzione per l'aggiunta dei Metadata che ha come parametri l'hash dei file e i suoi metadati, inoltre è stato aggiunto un require in modo che l'operazione venga annullata del caso l'hash sia già assegnato a dei metadati, questo viene fatto controllando un boolean. Infine, viene emesso l'evento che l'operazione è andata a buon fine

```
function addMetadata(string memory fileHash, string memory name, string memory mimeType,uint32 size) public
{
    require (metadataMap[keccak256(abi.encodePacked(fileHash))].value==false,"Already exists");
    IpfsMetadata memory metadata=IpfsMetadata(name,mimeType,msg.sender,size,block.timestamp,true);
    metadataMap[keccak256(abi.encodePacked(fileHash))]=metadata;
    emit AddIpfsFileEvent(fileHash);
}
```

Per ricavare i metadati di un file viene fatta una semplice GET con parametro l'hash del file

```
function getMetadata(string memory fileHash) public view returns(IpfsMetadata memory){
    return metadataMap[keccak256(abi.encodePacked(fileHash))];
}
```

Viene anche la possibilità di disabilitare dei dati e non cancellarli, ovviamente, in quanto è una blockchain, per fare questo si manda come parametro l'hash del file e controllo prima di tutto che l'associazione esiste e che colui che vuole cancellare l'associazione sia anche il creatore, fatto questo metto a false il boolean:

```
function deleteMetadata(string memory fileHash) public {
    require (metadataMap[keccak256(abi.encodePacked(fileHash))].value,"Notexists");
    require (metadataMap[keccak256(abi.encodePacked(fileHash))].owner==msg.sender,
```

```
"Not owner");
metadataMap[keccak256(abi.encodePacked(fileHash))].value=false;
}
```

4.4.2 MyAccessControl.sol

Questo contratto è stato sviluppato dall'azienda e consente l'autenticazione in Andromeda. L'azienda ha pensato che per accedere alla rete si debba avere un ruolo settato, se non si ha un ruolo non è possibile accedere.

Il contratto permette di associare a un address un ruolo. Questo avviene estendendo il noto accesscontrol.sol di OpenZeppelin, che utilizza la coppia *address* e *role* codificato in bytes32 e fornisce dei *setters* e dei *getters*.

Open Zeppelin è una libreria che ha lo scopo di fornire un set di Smart Contracts open-source, che sono stati scritti da esperti e testati ampiamente dalla comunità. Questo aiuta a ridurre al minimo il rischio di vulnerabilità e di exploit, sono costantemente aggiornati per fronteggiare nuove problematiche.

Inoltre, fornisce un'ottima documentazione per capire il funzionamento dei contratti.

Il nostro contratto utilizza un mapping per memorizzare i dati; quindi, il funzionamento è molto simile al precedente

```
struct RoleData {
mapping(address => bool) members;
bytes32 adminRole;
}

mapping(bytes32 => RoleData) private _roles;

bytes32 public constant DEFAULT_ADMIN_ROLE = 0x00;

function hasRole(bytes32 role, address account) public view virtual override returns (bool) {
return _roles[role].members[account];
}
```

[25]

4.5 Server-Side

Ho creato in Java Spring un Rest Controller che gestisce le richieste del client e richiama i metodi necessari scritti nella classe Service.

```
@PostMapping(value = "upload/{payload}")
public ResponseEntity<String> uploadFile(@RequestBody MultipartFile file,
@PathVariable("payload") String payload) throws IOException,
SignatureException {
```

```

        if (PayloadValidator.isValidSignature(payload))
            return ipfsService.uploadFile(file);
        else
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }

    @GetMapping(value = "file/{hash}")
    public ResponseEntity<byte[]> getFile(@PathVariable("hash") String hash,
    @RequestParam("payload") String payload) throws IOException,
    SignatureException {
        if (PayloadValidator.isValidSignature(payload))
            return ipfsService.loadFilez(hash);
        else
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
}

```

4.5.1 Upload File

Per l'upload del file viene mappata un chiamata POST che riceve un `MultipartFile` ed il payload, che prima controlla la validità del payload e se il wallet chiamante è abilitato, verificato questo prende il file e chiama il metodo `add` delle RPC API di Kubo

```
"http://127.0.0.1:5001/api/v0/add"
```

che a sua volta è una richiesta POST, a cui viene allegato il file, una volta andata a buon fine restituirà l'hash del file caricato e lo si restituisce all'interno di un `RequestEntity<String>` che il front-end leggerà.

4.5.2 Retrieve File

Per il retrieve del file viene mappata un chiamata POST che riceve l'hash del file ed il payload, che prima controlla la validità del payload e se il wallet chiamante è abilitato, verificato questo chiama il metodo `cat` delle RPC API di Kubo concatenando l'hash del file

```
"http://127.0.0.1:5001/api/v0/cat?arg=" + hash;
```

che a sua volta è una richiesta POST, una volta andata a buon fine restituirà il file come array di byte e viene restituito all'interno di un `RequestEntity<byte[]>` che il front-end leggerà.

4.5.3 Payload Validator

Per validare il payload viene fatta l'operazione inversa della creazione, si decodifica la prima parte del *payload* (il messaggio), si decodificano i valori *r,s,v* e viene ricomposta la firma(`SignatureData`)

Inserendoli come parametro al metodo:

```
public static BigInteger signedMessageToKey(byte[] message, SignatureData
signatureData)
    throws SignatureException {
    return signedMessageHashToKey(Hash.sha3(message), signatureData);
}
```

che restituisce la public key(address) che andrà confrontata con quella dentro il payload, così sarò sicuro della validità del payload, infine si andrà a richiamare lo Smart Contract *MyAccessControl* per verificare che l'address sia abilitato a lavorare nella blockchain.

Per facilitare tutto questo utilizzo la libreria web3j, costruita apposta per lavorare con web3 [26]

4.6 Client-Side

L'interfaccia è stata fatta utilizzando Vue 3, Vite 3, Vuetify 3, Vuex e Vue Router.

Per la parte di collegamento alla blockchain vengono richiamate delle funzioni scritte in javascript, che importano la libreria web3 e gli smart contracts in formato json.

4.6.1 Payload Generator

Viene prevista una gestione del wallet precedente, solo a fine di test viene permesso di inserire direttamente la private key.

L'interfaccia l'utente ha una casella dove inserire la chiave privata e un bottone che genera il payload, chiamando la funzione in javascript che fa tutto il lavoro descritto a 4.2 utilizzando le librerie di web3.

Una volta creato viene utilizzato Vuex, per salvare in locale il payload generato, in modo da poterlo passare tra le varie pagine.

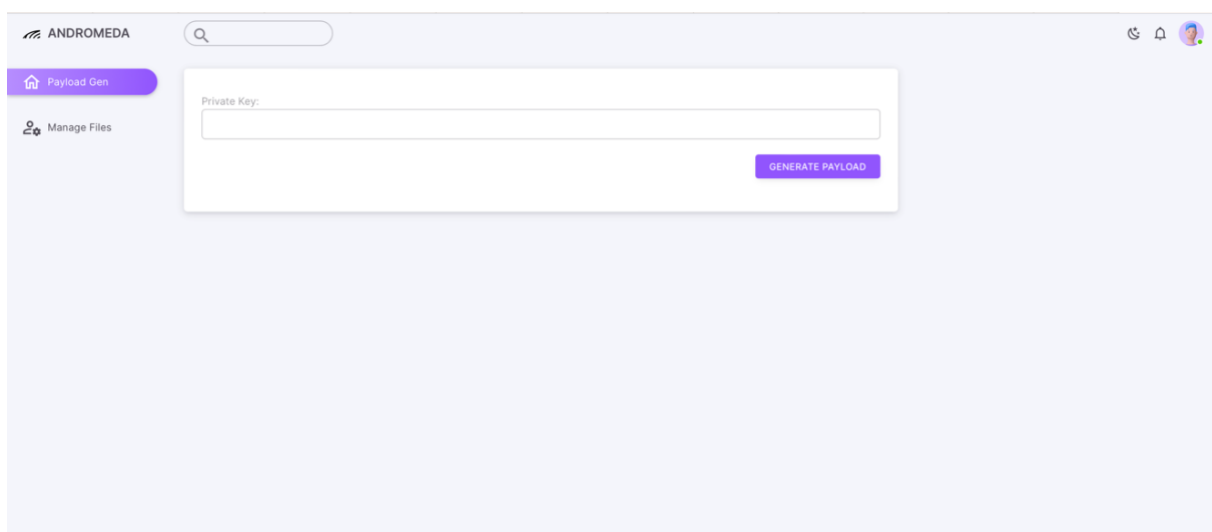


Figura 14 Generazione iniziale del payload

4.6.2 Upload File

Per fare l'upload del file l'utente allegherà il file in un form e premendo il tasto di submit verrà fatta una richiesta POST all'indirizzo delle mie REST API, con il file nel body della chiamata e il payload come parametro, una volta che il server restituisce l'hash del file viene fatta una chiamata allo Smart Contract, chiamando la funzione *addMetadata*, con parametro l'hash e i metadati del file ricavati in locale.

```
uploadFile() {
  this.file = this.$refs.fileInput.files[0]
  if (this.file) {
    this.fileName = this.file.name
    this.fileSize = this.file.size
    this.fileType = this.file.type
  } else {
    this.fileName = ''
    this.fileSize = 0
    this.fileType = ''
  }
},
submitHandler() {
  this.isLoading = true
  let input = new FormData()
  input.append('file', this.file)
  axios

    .post('http://127.0.0.1:8091/upload/' +
      store.getters.getSku,
      input)
    .then(async response => {
      console.log(response.data)
      this.hashz = response.data['Hash']
      await addMeta(response.data['Hash'], this.fileName, this.fileType,
this.fileSize)
      const meta = await getMeta(response.data['Hash'])
      this.metadata = 'Name: ' + meta[0] + ' Type: ' + meta[1] + ' Owner: '
+ meta[2] + ' Size : ' + meta[3] + ' bytes Timestamp: ' + meta[4]
      await this.refreshTable()
      this.isLoading = false
    })
    .catch(error => {
      this.isLoading = false
      console.log(error)
    })
}
```

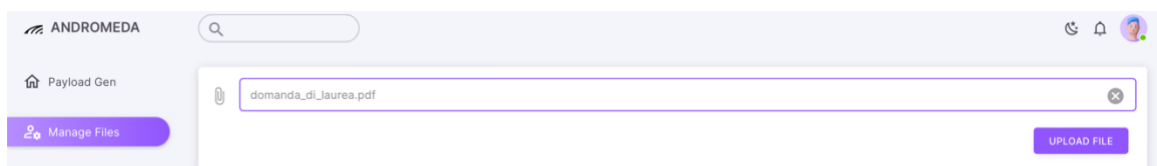


Figura 15 Upload del file

4.6.3 Retrieve File

Inserendo in una casella l'hash del file e premendo il tasto *submit* verrà fatta richiesta GET all'indirizzo delle REST API, con l'hash come parametro e il payload come parametro, dallo Spring Server verrà restituito un array di byte che utilizzeremo per creare un blob da scaricare e per scrivere il nome del file corretto viene fatta una chiamata allo Smart Contract *IpfsMetadata*, chiamando la funzione *getMetadata*, con parametro l'hash, così si avrà tutto il necessario per fare il download.

```
const meta = await getMeta(this.inputHash)

try {
  const response = await fetch('http://127.0.0.1:8091/file/' +
this.inputHash + ' ' +
  '?payload=' +
    store.getters.getSku, {
    method: 'GET',
    headers: {
      'Content-Type': meta[1],
    },
  })
  const blob = await response.blob()
  const url = window.URL.createObjectURL(new Blob([blob]))
  const link = document.createElement('a')
  link.href = url
  link.setAttribute('download', meta[0])
  document.body.appendChild(link)
  link.click()
} catch (error) {
  console.error(error)
}
```

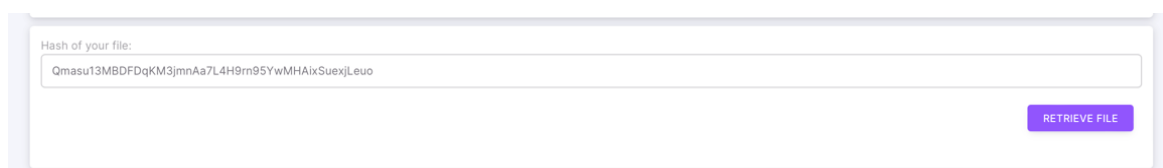


Figura 16 Download del file

4.6.4 Lista degli ultimi eventi dello Smart Contract

Per fare visualizzare gli ultimi eventi di uno Smart Contract utilizzo web3 collegandomi all'indirizzo dello Smart Contract e chiamando il metodo:

```
myContract.getPastEvents(event[, options][, callback])
```

Questo metodo restituisce una lista di hash per come è scritto lo Smart Contract, e per ognuno di loro chiamerò il metodo *getMetadata* che mi restituirà i metadati assegnati all'hash.

```
const fileHashes = res.map(obj => obj["returnValues"].fileHash);
const ris = []
for (const fileHash of fileHashes) {
  try {
    const result = await getMeta(fileHash);
    ris.push([fileHash, ...result]);
  } catch (error) {
    console.log(`Error calling method for file hash ${fileHash}:
${error}`);
  }
}
```

Infine, mi ritroverò con questo array contenente tutte le informazioni del file e mi basterà visualizzarle nella tabella nel front-end.

Name	Type	Owner	Size	Timestamp
Blockchain - Formazione - Bibliografia e materiale.pdf	application/pdf	0x2Ca324a93066cf042fE9590EaAB5A7bdbBC7DA84	222982 Bytes	1677931912
test.png	image/png	0x2Ca324a93066cf042fE9590EaAB5A7bdbBC7DA84	9060 Bytes	1677764443
Coderunner.pdf	application/pdf	0x2Ca324a93066cf042fE9590EaAB5A7bdbBC7DA84	56709 Bytes	1677763483
Test-Logo-Small-Black-transparent-1.png	image/png	0x2Ca324a93066cf042fE9590EaAB5A7bdbBC7DA84	37257 Bytes	1677751601
Quiz 1 English.pdf	application/pdf	0x2Ca324a93066cf042fE9590EaAB5A7bdbBC7DA84	89743 Bytes	1677682271
sample-3s.mp3	audio/mpeg	0x2Ca324a93066cf042fE9590EaAB5A7bdbBC7DA84	52079 Bytes	1677680234
massa-muscolare-cavallo.png	image/png	0x2Ca324a93066cf042fE9590EaAB5A7bdbBC7DA84	319938 Bytes	1677676917
002.svg	image/svg+xml	0x2Ca324a93066cf042fE9590EaAB5A7bdbBC7DA84	32553 Bytes	1677675236
20.png	image/png	0x2Ca324a93066cf042fE9590EaAB5A7bdbBC7DA84	460779 Bytes	1677675091
Imroman10-bold.otf	font/otf	0x2Ca324a93066cf042fE9590EaAB5A7bdbBC7DA84	107780 Bytes	1677675020

Figura 17 Ultimi dieci eventi sul contratto

5 Conclusioni

Durante il mio stage ho avuto modo di conoscere nuove tecnologie ed in particolar modo il protocollo IPFS e gli Smart Contracts. Questo ha permesso di arricchire la piattaforma Andromeda con questa nuova possibilità di salvare qualsiasi tipo di file nella rete blockchain aziendale, costruendo con un'interfaccia alla portata di tutti senza sapere il meccanismo complesso che c'è dietro.

Mi è piaciuto molto il lavoro che ho fatto perché ho imparato delle nozioni complesse e son riuscito a sviluppare un progetto utilizzabile da chiunque.

Ringraziamenti

Innanzitutto, ringrazio me stesso per il mindset che ho costruito e con cui ho affrontato questa esperienza e tante altre.

Ringrazio calorosamente i miei genitori per il costante supporto in tutto quello che ho fatto e che farò.

Ringrazio i miei amici, quelli veri, per essermi sempre stati vicino.

Ringrazio per la possibilità che mi è stata concessa, per la quale ho potuto vivere l'esperienza Erasmus ad Oslo, che è stata molto impattante nella mia crescita personale ed universitaria.

Bibliografia

- [1] «Alten Italia,» [Online]. Available: <https://www.alten.it>.
- [2] E. Sandrone, «Realizzazione di una piattaforma per l'analisi del flusso di dati in una blockchain Ethereum,» Università degli studi di Torino.
- [3] «Young Platform,» [Online]. Available: <https://academy.youngplatform.com/blockchain/blockchain-internet-innovazione/>.
- [4] «Young Platform,» [Online]. Available: <https://academy.youngplatform.com/blockchain/proof-of-stake-vs-proof-of-work-blockchain/>.
- [5] «BlockGeeks,» [Online]. Available: <https://blockgeeks.com/guides/proof-of-work-vs-proof-of-stake/>.
- [6] «Binance Academy,» [Online]. Available: <https://academy.binance.com/en/articles/proof-of-authority-explained>.
- [7] Ethereum. [Online]. Available: <https://ethereum.org/it/>.
- [8] «Applicature,» [Online]. Available: <https://applicature.com/blog/blockchain-technology/ethereum-programming-tutorial>.
- [9] Ethereum. [Online]. Available: <https://ethereum.org/it/developers/docs/gas/>.
- [10] Ethereum. [Online]. Available: <https://ethereum.org/en/smart-contracts/>.
- [11] Ethereum. [Online]. Available: <https://ethereum.org/en/dapps/>.
- [12] «Moesif,» [Online]. Available: <https://www.moesif.com/blog/blockchain/ethereum/Tutorial-for-building-Ethereum-Dapp-with-Integrated-Error-Monitoring/>.
- [13] IPFS. [Online]. Available: <https://docs.ipfs.tech/concepts/what-is-ipfs/#defining-ipfs>.
- [14] «IPFS Tech,» [Online]. Available: <https://blog.ipfs.tech/2022-06-09-practical-explainer-ipfs-gateways-1/>.
- [15] «Kubo IPFS,» [Online]. Available: <https://github.com/ipfs/kubo>.
- [16] «IPFS,» [Online]. Available: <https://docs.ipfs.tech/concepts/ipfs-gateway/>.
- [17] Metamask. [Online]. Available: <https://metamask.io>.
- [18] «Alphr,» [Online]. Available: <https://www.alphr.com/add-funds-metamask/>.
- [19] «Remix IDE,» [Online]. Available: <https://remix-project.org>.

- [20] «Wallarm,» [Online]. Available: <https://www.wallarm.com/what/what-is-json-rpc>.
- [21] M. Mondini, «Gestione degli accessi tramite consenso distribuito per una blockchain in,» Università degli studi di Torino.
- [22] «Java Spring,» [Online]. Available: <https://spring.io>.
- [23] Vue.js. [Online]. Available: <https://vuejs.org>.
- [24] S. v. Laar, «Deploy a private IPFS network in 5 steps,» [Online]. Available: https://medium.com/@s_van_laar/deploy-a-private-ipfs-network-on-ubuntu-in-5-steps-5aad95f7261b.
- [25] «OpenZeppelin,» [Online]. Available: <https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/AccessControl.sol>.
- [26] Web3j. [Online]. Available: <https://docs.web3j.io/4.8.7/>.

Indice delle figure

Figura 1 PoW vs PoS [2]	10
Figura 2 Bitcon vs Ethereum [4]	12
Figura 3 Sviluppo di una dApp [8]	14
Figura 4 HTTP vs IPFS [10]	16
Figura 5 IPFS API [12]	17
Figura 6 Metamask [14]	18
Figura 7 JSON-RPC Request [16]	20
Figura 8 Esempio di utilizzo reale	22
Figura 9 Workflow	25
Figura 10 Upload File	27
Figura 11 Retrieve File	28
Figura 12 Event Listener	29
Figura 13 Esempio da terminale con tre nodi	31
Figura 14 Generazione iniziale del payload	35
Figura 15 Upload del file	36
Figura 16 Download del file	37
Figura 17 Ultimi dieci eventi sul contratto	38