Indice

Elenco delle figure

Introduzione

Nell'era moderna dell'informatica, ogni anno molte tecnologie emergono in diversi settori della vita quotidiana promettendo di essere la risposta a tutti i problemi. Dal suo anno di lancio, il 2016, questa tecnologia é la blockchain. Il primo utilizzo pratico della blockchain si é visto con la criptovaluta Bitcoin ed in seguito anche con altre criptovalute come Ethereum. Molti addetti ai lavori considerano la blockchain come un vero e proprio punto di svolta nell'informatica moderna; infatti, la blockchain sta cambiando le attivitá giornaliere essendo applicabile in diversi ambiti e contesti. Ad oggi, peró, la blockchain non é (ancora) la panacea di tutti i problemi; le difficoltá maggiori di questa tecnologia sono la scalabilitá, l'integritá dei partecipanti alla rete, la distribuzione della potenza computazionale, la preservazione della confidenzialitá degli utenti e la loro sicurezza.
Sulla falsa riga della blockchain, l'altra grande tecnologia che ha preso piede negli ultimi anni é l'utilizzo dei Non-Fungible-Token (NFT). Gli NFT sono nati nel 2014 ma diventati virali nel 2017 con l'avvento di *CryptoKitties*, gioco online che permetteva agli utenti di collezionare e vendere NFT; successivamente hanno trovato terreno fertile soprattutto nella criptoarte poggiando le loro basi proprio sulla blockchain. Il mercato degli NFT é in continua crescita. Nel 2021, anno del boom degli NFT, questo mercato ha toccato la cifra record di 17 miliardi di dollari e un incremento del 21.000% rispetto al 2020, quando erano poco piú che un settore di nicchia.
L'obiettivo di questo lavoro si tesi é la realizzazione di un' applicazione web decentralizzata (le cosiddette *dApp*) che funziona sulla blockchain Ethereum e che permette agli utenti di creare Non Fungible Token (NFT) partendo da documenti in formato pdf caricati da locale e scambiarli con altri utenti con l'ausilio di un

portafoglio digitale (*MetaMask*) per permettere lo scambio di criptovaluta. Nei capitoli seguenti si illustreranno tutti gli elementi chiave necessari allo sviluppo dell'applicazione appena descritta.

Nel primo capitolo verranno esposti i concetti di base nello sviluppo dell'applicazione. In particolare verrá mostrato lo stato dell'arte delle tecnologie informatiche a supporto dello sviluppo di applicazioni web decentralizzate, introducendo, quindi, i concetti di Blockchain, NFT, Smart Contract e come questi vengono impiegati e implementati in Ethereum.

Successivamente, nel secondo capitolo, saranno illustrate le tecnologie utilizzate nello sviluppo dell'applicazione web decentralizzata, spiegando piú nel dettaglio gli strumenti utilizzati per comunicare con la blockchain e per realizzare l'interfaccia grafica.

Nel terzo capitolo si descrive nello specifico come é stata realizzata l'applicazione oggetto della tesi. Saranno illustrati, dapprima, i casi d'uso presi in analisi per lo sviluppo dell'applicazione, corredando la descrizione di questi ultimi con schermate del lavoro finale. Di seguito si presenterá il System Design dell'applicazione, descrivendo il design pattern utilizzato e l'organizzazione dei vari moduli che compongono l'applicazione. Nel paragrafo successivo si passeranno in rassegna alcune delle funzioni piú importanti sviluppate nel corso del lavoro di tesi.

Il quarto capitolo é dedicato alla fase di testing, mostrando i casi di test effettuati nelle varie fasi dello sviluppo dell'applicazione, in particolare durante la stesura dello smart contract e nella creazione delle pagine web.

Capitolo 1

Concetti di base

In questo capitolo verranno introdotti i concetti chiave necessari per lo sviluppo dell'applicazione oggetto della tesi. In particolare si porrá l'accento sullo stato dell'arte dei vari strumenti utilizzati come blockchain, NFT e smart contract e piú nel dettaglio di come questi ultimi sono implementati ed utilizzati nella blockchain Ethereum.

1.1 Blockchain

La Blockchain (letteralmente catena di blocchi) puó essere considerata come un registro di contabilitá "immutabile" e condiviso tra tutti i partecipanti che facilita la registrazione di qualsiasi tipo di transazione e la tracciabilitá degli asset in una rete commerciale. Tutto ció che ha un valore puó essere validato, rintracciato e scambiato su una rete blockchain, riducendo rischi e costi per tutti i partecipanti alla rete. La blockchain é ideale per trasmettere dati perché fornisce informazioni immediate, condivise e completamente trasparenti archiviate in un registro immutabile a cui possono accedere solo i membri di reti autorizzati, come avviene nelle blockchain private, oppure qualsiasi membro della rete nel caso di blockchain pubbliche. Una rete blockchain puó tracciare ordini, pagamenti, account, produzione ed altro ancora, salvando tutta una serie di informazioni all'interno dei blocchi che la compongono. Sebbene la sua dimensione sia destinata a crescere nel tempo, é immutabile nel senso che il suo contenuto, una volta validato

tramite un meccanismo di consenso, non é piú né modificabile né eliminabile, a meno di non invalidare l'intero processo, ma ció necessita dell'aggiunta di un nuovo blocco.

La tecnologia blockchain appartiene alla famiglia dei Distributed ledger, ossia sistemi che si basano su un registro distribuito, che puó essere letto e modificato da piú nodi o membri di una rete. Non é richiesto che i membri coinvolti conoscano l'identitá reciproca o si fidino l'uno dell'altro perché, per garantire la coerenza tra le varie copie, la validazione di un nuovo blocco é globalmente regolata da un protocollo condiviso. Una volta autorizzata la validazione del nuovo blocco, ogni membro aggiorna la propria copia privata. La natura stessa della struttura dati garantisce l'assenza di una sua manipolazione futura.

La blockchain é quindi un database strutturato in blocchi collegati tra loro e assicurati mediante l'uso della crittografia rappresentabile come una lista, in continua crescita. Ad un blocco possono essere associate una o piú transazioni e ogni blocco, inoltre, contiene un puntatore hash al blocco precedente e una marca temporale in modo da essere sempre rintracciabile. La natura distribuita e il modello cooperativo rendono robusto e sicuro il processo di validazione ma presentano tempi di aggiornamento lunghi, dovuti al processo di validazione dei blocchi, successiva alla fase di mining, e alla sincronizzazione delle rete. L'utilizzo di questa tecnologia consente anche di superare il problema dell'infinita riproducibilitá di un bene digitale e della doppia spesa, senza l'utilizzo di un server o autoritá centrale. Talvolta risulta possibile che alcuni membri della rete producano simultaneamente piú blocchi "concorrenti" (ossia collegati a uno stesso blocco giá esistente, ma diversi tra loro nel contenuto): ció dá origine a una biforcazione nella catena, regolata in seguito da un protocollo che indica il blocco da accettare.

1.1.1 Elementi chiave di una blockchain

Di seguito saranno mostrate alcune delle caratteristiche piú innovative che presenta la blockchain:

- tecnologia di registro distribuito: tutti i partecipanti alla rete hanno accesso al registro distribuito e alle transazioni in esso contenuto; in particolare tutti gli utenti o membri della rete hanno una copia della blockchain e, grazie alla massiva duplicazione del database, si mantiene la qualitá dei dati.

- record immutabili e incorruttibili: nessun membro della rete potrá modificare o manomettere una transazione, una volta annotata nel registro condiviso. Se un record di transazione contiene un errore dovrá essere aggiunto un nuovo blocco per correggerlo.

- decentramento: sfrutta un networking distribuito per fare in modo di memorizzare i dati su tutta la sua rete ed evitare di avere un *single point of failure* che potrebbe essere preda di hacker per far collassare l'intera rete. La crescita della blockchain decentrata va di pari passo con il rischio della centralizzazione dei nodi perché le risorse informatiche richieste per operare e gestire dati sempre piú grandi diventano sempre piú onerose. Infatti molti miner si aggregano in gruppi per riuscire a validare i blocchi piú velocemente e ottenere ricompense in criptovalute.

- trasparenza: tutti i membri della rete possono verificare in qualsiasi momento le transazioni registrate e i blocchi validati.

1.1.2 Struttura del blocco

Le transazioni sono organizzate in blocchi e il numero di transazioni all'interno di ognuno di questi blocchi varia in base alla dimensione della transazione stessa. La dimensione della transazione varia in base al numero di input e di output della stessa. Un blocco é composto da due parti principali: l'*header* e il *body*. Le transazioni sono racchiuse nel *body* del blocco e nell'*header* sono presenti sette campi per la gestione del blocco stesso.
Alcuni dei campi piú importanti all'interno di un blocco sono mostrati nella tabella sottostante.

Versione	02000000
PrevHash	E87C17C45768w7e1643fsd5481sd3f4131df681
Merkle root	697we168t4v1a4rv3v1e3r43c4er14ca8c4168a
Timestamp	358b0553
Bits	535f0119
Nonce	48750933
Numero di transazione	64

Il campo *Versione* dipende dalla versione del software utilizzato, il campo *Prev-Hash* (ovvero l'hash del blocco precedente) é un hash di 256 bit che serve per fare riferimento al precedente blocco, il *Merkle root* é l'hash di tutti gli hash di tutte le transazioni nel blocco, il campo *Timestamp* rappresenta la marca temporale dell'ultima transazione, il campo *Bits* rappresenta il corrente valore target: l'hash dell'header di un blocco dev'essere minore o uguale al corrente valore di target per essere accettato dalla rete, il campo *Nonce* é un valore a 8 byte che viene aggiunto al blocco in modo che l'output della funzione di hash vari facendo in modo che risulti inferiore al valore target, il valore viene ricalcolato finché l'hash del blocco non contiene il richiesto numero di zeri principali e infine il campo *Numero di transazione* identifica il numero della transazione.

1.1.3 Mining e validazione di un blocco

Il mining si pone come obiettivi principali la generazione di nuova criptovaluta e la verifica della legittimitá delle transazioni in criptovaluta sulla relativa blockchain. Il processo di mining é un' operazione onerosa che richiede grandi quantitá di potenza di calcolo ed energia che puó derivare dai computer degli utenti stessi. In pratica un utente puó mettere a disposizione il proprio pc e ricevere criptovaluta in cambio per ogni nuovo blocco validato. Inoltre il processo é molto competitivo perché la quantitá di criptovaluta immessa é prestabilita; se i miner sono troppi, le difficoltá aumentano e i ricavi diminuiscono.
Esistono diversi modi per effettuare il mining, alcuni dei quali vengono elencati di seguito:

- Mining attraverso il proprio pc: attraverso l'utilizzo della propria CPU e della scheda video, sará possibile immettere sulla rete nuova criptovaluta. Il parametro piú importante da considerare é il megahash per second (MH/s) che corrisponde al numero di hash check che la scheda video compie in un secondo.

- Mining tramite hardware specializzato: questi tipi di hardware sono ottimizzati per il mining avendo le giuste proporzioni tra consumo elettrico e potenza a discapito di un costo di produzione molto elevato.

- Mining attraverso pool specializzate: esistono diverse aziende che offrono agli utenti l'opportunitá di affittare la loro potenza di calcolo per il mining. Queste societá hanno stanze intere piene di computer ottimizzati per il processo e ne "affittano" una parte. L'utente potrá quindi acquistare la potenza di calcolo e ricevere una rendita passiva mensile.

Ci sono diversi algoritmi che permettono di effettuare il mining; solitamente sono funzioni hash crittografiche molto complesse. Tra gli algoritmi piú utilizzati ci sono: SHA-256 (nato con Bitcoin), Ethash (risultato dall'unione di diversi algoritmi), Scrypt (é una funzione di derivazione chiave basato su una password), X11 (consiste in un gruppo di algoritmi XNUMX con diverse funzioni di hashing che insieme fungono da algoritmo di mining), Equihash (sviluppato da un gruppo di ricercatori del centro interdisciplinare per la sicurezza, l'affidabilitá e la fiducia di Lussemburgo), CryptoNight (che garantisce un alto livello di sicurezza e anonimato).
Il mining permette quindi di creare un nuovo blocco sulla blockchain e di validarlo di conseguenza, consentendo anche al miner di guadagnare criptovaluta; al blocco appena creato corrisponderá l'hash (ottenuto tramite una delle funzioni descritte in precedenza) che lo rappresenta univocamente all'interno della rete.

1.1.4 Vantaggi della blockchain

Di seguito sono descritti alcuni dei vantaggi derivanti dall'utilizzo della blockchain:

- Maggiore fiducia: con la blockchain, in qualitá di membro di una rete di soli partecipanti, si puó confidare nel fatto di ricevere dati accurati, trasparenti e tempestivi e che i record della blockchain saranno condivisi solo con i membri della rete.

- Maggiore sicurezza: ai membri della rete viene richiesto il consenso sull'accuratezza dei dati e tutte le transazioni convalidate sono immutabili essendo registrate in modo permanente e non piú modificabili.

- Maggiori efficienze: avendo un registro distribuito condiviso tra i membri di una rete, le riconciliazioni di record possono essere eliminate; per accelerare le transazioni, uno smart contract puó essere memorizzato sulla blockchain ed eseguito automaticamente quando necessario.

1.1.5 Tipi di reti blockchain

Di seguito sono elencate alcune dei tipi di rete blockchain piú utilizzati:

- Reti blockchain pubbliche: é una rete a cui chiunque puó accedere e partecipare (come Bitcoin); gli inconvenienti potrebbero includere la necessitá di una notevole potenza di calcolo, poca o nessuna privacy a tutela delle transazioni e scarsa sicurezza. Queste considerazioni si rivelano importanti nell'utilizzo in contesto aziendale.

- Reti blockchain private: é una rete peer-to-peer decentralizzata (similmente a una blockchain pubblica); tuttavia, una singola organizzazione governa la rete, come nel caso di Ethereum, controllando chi é autorizzato a partecipare, eseguire un protocollo di consenso e mantenere il registro condiviso. A seconda del caso di utilizzo, questo tipo di rete puó incentivare in modo significativo la fiducia e la sicurezza tra i membri.

- Reti blockchain con autorizzazioni: le attivitá di business che impostano una blockchain privata solitamente creano una rete blockchain basata su autorizzazioni. É importante tenere presente che anche le reti blockchain pubbliche possono essere basate su autorizzazioni. In questo caso si pongono

dei limiti su chi é autorizzato a partecipare alla rete e a quali transazioni puó partecipare.

- Reti blockchain di un consorzio: piú organizzazioni possono condividere le responsabilitá della gestione di una blockchain. Queste organizzazioni stabiliscono chi puó effettuare transazioni e accedere ai dati. Una blockchain di consorzio é una soluzione ideale per il business in quanto tutti i partecipanti devono essere autorizzati e hanno una responsabilitá condivisa per la blockchain.

1.1.6 Sicurezza della blockchain

La sicurezza della blockchain dipende da svariati fattori e soprattutto dagli effetti simultanei della loro applicazione. Di seguito vengono elencati gli elementi principali:

- crittografia a due fasi: ogni transazione é regolata da una chiave pubblica e una chiave privata, che consente a tutti di verificarla e soltanto ai legittimi proprietari di poterne effettivamente disporre.

- algoritmi di *proof-of-work*: le transazioni vengono validate dai miner, che calcolano l'hash attraverso uno degli algoritmi descritti in precedenza. L'hash identifica univocamente un blocco e dipende dal contenuto dello stesso per cui qualsiasi contraffazione ne comporterebbe una modifica.

- struttura a blocchi concatenati: l'hash di un blocco é riportato nell'header del blocco successivo, quindi é praticamente impossibile violare un blocco senza invalidare tutti i blocchi successivi della catena.

- registro distribuito: la blockchain si basa su un sistema *peer-to-peer*, con una serie di macchine su cui é installato il suo software di gestione.

- trasparenza del software di autoregolamentazione: il software che regola il funzionamento di una blockchain é open source, per cui chiunque puó verificarne ogni dettaglio ma non puó sperare di modificarla con successo senza aver ottenuto il consenso della maggioranza dei membri della rete.

Inoltre, il design della blockchain sfrutta gli effetti combinati di una serie di tecnologie basate sulla crittografia e di un sistema di autoregolamentazione molto solido. Anche tutti i sistemi di autenticazione previsti sono assolutamente robusti grazie alla presenza delle firme digitali.
Ci sono dei punti deboli, piuttosto marginali, che si possono presentare durante le fasi di avvio di una blockchain pubblica; questo perché sarebbe piú facile portare a termine l'attacco del 51% avendo, quindi, un'influenza sulla maggioranza dei nodi della rete, qualora i nodi non fossero sicuri.

1.1.7 Casi d'uso di reti blockchain

La blockchain é quindi uno strumento potente che sta giá rivoluzionando diversi ambiti della vita giornaliera, tra cui:

- ambito finanziario: uno dei modi piú utilizzati soprattutto perché la blockchain permette le transazioni in criptovalute.

- ambito assicurativo: per quanto riguarda l'elaborazione dei sinistri, la blockchain fornisce un sistema ideale per la gestione e la trasparenza senza rischi. La crittografia permette, infatti, agli assicuratori di avere i diversi dati delle proprietá da assicurare in modo tale da evitare imprevisti durante i sinistri.

- per pagamenti oltre oceano: la blockchain permette di abbattere i relativi problemi legati ai metodi tradizionali per gli accordi commerciali di tutto il globo.

- per prestiti intelligenti: gli attuali sistemi di prestito prevedono la presenza di un intermediario che garantisca per colui che percepisce la somma del prestito. Il sistema blockchain permette anche a semplici privati di poter prestare delle somme senza alcun rischio per nessuna delle due parti, abbattendo notevolmente i costi.

- in ambito medico: grazie alla blockchain sarebbe possibile salvare le cartelle cliniche dei pazienti permettendo l'accesso solo al personale autorizzato. Sarebbe possibile anche affidare l'intero sistema sanitario a questa tecnologia come ad esempio la gestione dei farmaci, il rispetto delle normative, i risultati dei test e molto altro.

- per i passaporti: sono giá presenti passaporti che utilizzano il sistema di crittografia, grazie all'utilizzo di una chiave pubblica e una privata in modo da memorizzare il passaporto sulla blockchain.

1.2 Smart Contract

Gli smart contracts sono protocolli informatici che facilitano, verificano o fanno rispettare la negoziazione o l'esecuzione di un contratto. In particolare, uno smart contract é un programma, scritto in uno specifico linguaggio per lo sviluppo di smart contract chiamato Solidity, in esecuzione sui nodi validatori di una rete blockchain e il cui risultato rappresenta una transazione sulla quale i nodi validatori devono trovare un consenso (attraverso algoritmi di tipo *proof-of-work* o *proof-of-stake*). Questa accezione di smart contract non é esattamente quella di contratto classico, ma piuttosto quella di un vero e proprio codice la cui esecuzione e i cui output sono garantiti integri dalle proprietá di una blockchain; il nome deriva dalla scelta del progetto Ethereum di denominare tale codice in esecuzione smart contract. Gli smart contracts funzionano seguendo delle istruzioni condizionali scritte nel codice di una blockchain.
Una rete di computer esegue le azioni quando le condizioni prestabilite sono state verificate e conseguentemente la blockchain viene aggiornata con la validazione di nuovi blocchi. I vantaggi principali sono:

- velocitá;

- efficienza;

- accuratezza (quando una condizione viene soddisfatta, il contratto viene eseguito immediatamente);

- attendibilitá e trasparenza (poiché non sono coinvolte terze parti e i record delle transazioni sono condivisi tra i partecipanti, non é necessario chiedersi se le informazioni sono state modificate per vantaggio personale);
- sicurezza (data dalla natura della blockchain, come descritto in precedenza);
- risparmi (dovuti alla rimozione di intermediari);

1.2.1 Algoritmo proof-of-work

Un sistema proof-of-work (POW) o protocollo proof-of-work é un algoritmo di consenso (soprattutto utilizzato sulla blockchain di Bitcoin) che puó essere utilizzato per scoraggiare attacchi di tipo negazione di servizio e altri abusi di servizio, ad esempio spam sulla rete, imponendo alcuni lavori dal richiedente del servizio, come il tempo di elaborazione di un computer. Una caratteristica chiave di questi schemi é la loro asimmetria: il lavoro deve essere moderatamente complesso (ma fattibile) dal lato richiedente ma facile da controllare per il fornitore del servizio. I sistemi proof of work sono usati come base da altri sistemi crittografici piú complessi.

1.2.2 Algoritmo proof-of-stake

É un tipo di protocollo per la messa in sicurezza di una rete di criptovaluta e per il conseguimento di un consenso distribuito. É basato sul principio che a ogni utente venga richiesto di dimostrare il possesso di un certo ammontare di criptovaluta. Si differenzia dai sistemi proof of work che sono basati su algoritmi di hash che validano le transazioni elettroniche. Gli algoritmi di tipo proof-of-stake vengono utilizzati soprattutto quando si deve scegliere il creatore del blocco successivo; questo avviene ogniqualvolta un nuovo blocco viene aggiunto alla blockchain. Dato che il creatore di un nuovo blocco non puó essere l'account che possiede la maggiore quantitá della criptovaluta (altrimenti questo creerebbe tutti i blocchi), sono stati pensati diversi metodi di selezione, quali:

- selezione casuale;
- selezione basata sull'anzianitá;
- selezione basata sulla velocitá;
- selezione basata sul voto.

1.3 Ethereum

Ethreum é una piattaforma decentralizzata del Web 3.0 di tipo computazionale. Tale piattaforma puó essere usata da chiunque desideri entrare a farne parte, mettendo a disposizione un archivio immutabile e condiviso di tutte le operazioni attuate nel corso del tempo e che allo stesso tempo é concepita per non poter essere fermata o censurata, dato che non esiste un controllo centrale. Ethereum é una *programmable blockchain*, che permette agli utenti di creare proprie operazioni, oltre a fornirne di predefinite e standardizzate. Di fatto é una blockchain platform che permette di dare vita a diverse tipologie di applicazioni blockchain decentralizzate non necessariamente limitate alle sole criptovalute.

1.3.1 Smart contract in Ethereum

Attraverso la blockchain Ethereum si possono vincolare le decisioni prese consensualmente sulla rete, piuttosto che subordinarle a un ente centrale che autorizzi tutte le attivitá. Gli smart contract, come visto in precedenza, servono a eseguire porzioni di codice che vengono interessate da una transazione esercitando un controllo diretto sul proprio conto di valuta ether e sul proprio valore, solitamente, infatti, é necessario un pagamento in criptovaluta affiché il contratto esegua le sue funzioni: l'obiettivo é tenere traccia delle variabili in gioco in modo da garantire tracciabilitá e trasparenza.
All'interno di una transazione sono presenti diverse informazioni importanti, quali:

- il nominativo del destinatario del messaggio;

- la firma del mittente;
- la quantitá di Ether oggetto della transazione;
- un valore che rappresenta il numero massimo di passaggi che possono essere eseguiti nella transazione;
- un valore pari alla commissione pagata dal mittente per lo step computazionale (*gas*).

Dunque all'interno della stessa rete é necessario pagare in Ether lo stesso network per poter usufruire della potenza computazionale; quindi Ether non é solo la blockchain ma anche la criptovaluta necessaria per effettuare le transazioni.
Il contratto, a differenza di un contratto classico, prevede delle variabili quindi reagisce agli input con output consequenziali. Tuttavia lo smart contract rispetta e vincola i contraenti, vincolando le parti in causa e senza alcuna possibiltá di modificare il contratto. I contratti all'interno della blockchain Ethereum permettono di effettuare tutta una serie di operazioni importanti, quali la registrazione di un dominio, l'avvio di un crowdfunding e la tutela delle proprietá intellettuali necessaria nella criptoarte e negli NFT.

1.3.2 Ethereum Virtual Machine

I partecipanti a Etherum possono disporre della *Ethereum Virtual Machine* (EVM) in grado di eseguire diversi algoritmi su una rete globale basata sui nodi di tutti i partecipanti. La EVM rappresenta l'ambiente di runtime per lo sviluppo e la gestione di smart contracts in Ethereum. La EVM opera in maniera completamente separata dalla rete; infatti, il codice gestito dalla Ethereum Virtual machine non ha accesso alla rete e gli stessi smart contracts generati sono indipendenti e separati dagli altri smart contracts. Gli smart contracts sono disponibili sulla blockchain in EVM bytecode, sono scritti in un liguaggio chiamato Solidity, trasformati in bytecode grazie all'utilizzo di un compilatore EVM e caricati sulla blockchain con un client Ethereum, in seguito verranno approfonditi i concetti di deploy e compilazione degli smart contracts.

1.3.3 Ether

I partecipanti a Ethereum lavorano su una rete peer-to-peer e sviluppano e gestiscono contratti di Ethereum utilizzando le risorse computazionali della rete. L'uso di queste risorse viene ricompensato con una speciale moneta virtuale, l'Ether. Ether ha di fatto un doppio ruolo: da una parte é essa stessa la potenza elaborativa necessaria per produrre i contratti e dall'altra rappresenta la criptovaluta che permette di pagare la realizzazione dei contratti. Ether é fondamentalmente un token. Ethereum si avvale di un meccanismo di commissioni denominato *Gas* che ha lo scopo di ottimizzare le risorse della rete in modo proporzionato in funzione delle richieste. Le commissioni totali da pagare per validare un nuovo blocco equivalgono a: *Gas unit* · (commissioni di base + mancia), dove *Gas unit* si riferisce all'ammontare massimo di ether che si vuole pagare per una transazione, le commissioni di base si riferiscono all'ammontare minimo di ether da pagare per aggiungere una transazione sulla blockchain Ethereum e la mancia si aggiunge per velocizzare la transazione.

La coniazione é il processo con cui vengono creati e immessi in rete nuova criptovaluta ether. Il protocollo sottostante di Ethereum crea i nuovi ether, cosa impossibile da fare per un utente. L'ether é coniato quando un miner crea e valida un nuovo blocco sulla blockchain di Ethereum. Come incentivo ai miner, il protocollo concede una ricompensa in ogni blocco, aumentando il saldo di un indirizzo impostato dal miner del blocco, come visto in precedenza. Oltre alla possibilitá di creare ether, quest'ultimo puó essere anche distrutto o "bruciato". L'ether bruciato viene rimosso dalla circolazione in via permanente. La bruciatura di ether ha luogo in ogni transazione su Ethereum. Quando gli utenti pagano per le proprie transazioni, la loro commissione sul gas viene distrutta dal protocollo. Poiché molte transazioni su Ethereum sono di dimensioni ridotte, l'ether ha diverse denominazioni che possono essere utilizzate per importi di piccola entitá, essendo 1 Ether equivalente a circa 1200 €. Tra queste denominazioni, le sottomisure Wei e gwei (prendono il nome da Wei Dai, attivista del mondo cripto, che ha diffuso l'utilizzo di criptovalute), che equivalgono rispettivamente a 10^{-18} e 10^{-9} Ether, sono particolarmente importanti e frequentemente utilizzate.

1.3.4 Decentralized Application

Un'applicazione decentralizzata (abbreviato dApp) é un'applicazione costruita su una blockchain che coniuga uno smart contract (in Solidity) con un'interfaccia utente. In Ethereum gli Smart Contract sono accessibili e trasparenti (come le API aperte) quindi una dapp puó anche includere Smart Contract scritti da altri ed importati, come avviene con *OpenZeppelin*. Il codice backend di una dapp viene eseguito su una rete blockchain. L'opposto di quello che succede con una qualsiasi app a cui la maggior parte degli utenti é abituata il cui codice backend viene eseguito su server centralizzati. Una dapp puó avere codice frontend e interfacce utente scritti in qualsiasi linguaggio (come qualsiasi app) che utilizzano gli smart contract per interfacciarsi con il backend.
Le caratteristiche principali di una dApp sono:

- decentralizzazione: le dApp sono eseguite su Ethereum, nel quale nessuna organizzazione detiene un qualsiasi tipo di controllo centrale;
- determinismo: eseguono la stessa funzione a prescindere dall'ambiente dove vengono eseguite;
- Turing complete: le dApp possono eseguire qualsiasi azione una volta fornite le risorse necessarie;
- isolamento: le dApp sono isolate rispetto ad altri programmi grazie alla Ethereum Virtual Machine. In questo modo, se lo smart contract ha un bug, non ostacolerá il normale funzionamento della rete blockchain.

I vantaggi nello sviluppare dapp sono: nessun tempo di inattivitá, privacy, resistenza alla censura, completa integritá dei dati, comportamento verificabile. D'altro canto lo sviluppo di dapp porta anche degli svantaggi quali difficoltá di manutenzione, overhead delle prestazioni e congestione della rete.

1.3.5 Differenze tra Web2 e Web3

Web2 si riferisce alla versione di Internet che la maggior parte degli utenti conosce e utilizza. Una rete dominata da aziende che offrono servizi in cambio dei dati

personali. Il Web3, nel contesto di Ethereum, si riferisce alle applicazioni decentralizzate che vengono eseguite sulla blockchain. Le dapp consentono a chiunque di partecipare senza monetizzare i propri dati personali.
Molti sviluppatori Web3 hanno deciso di sviluppare dapp per via delle decentralizzazione intrinseca di Ethereum, per usufruire dei diversi vantaggi offerti in generale dalle blockchain, e in particolare da Ethereum, quali:

- chiunque sia in rete ha il permesso di utilizzare il servizio. In altre parole, non serve chiedere un permesso;
- nessuno puó bloccare un utente o impedirgli l'accesso al servizio;
- i pagamenti sono incorporati tramite il token nativo, ether (ETH);
- Ethereum é Turing completa, significa che si puó programmare praticamente qualsiasi cosa.

Per fare un confronto pratico si pensi, ad esempio, a Twitter. Nel contesto del web2 un tweet o un account possono essere bloccati o censurati, ma questo non puó accadere nel web3 perché non esiste un controllo centrale.
Come spesso accade con le nuove tecnologie, anche il Web3 soffre di diverse limitazioni come la difficoltdi interazione con l'interfaccia utente, la quale potrebbe richiedere diversi passaggi, software e formazione aggiuntivi; la mancanza di integrazione nei browser moderni rende l'utilizzo da parte degli utenti meno accessibile; per via del costo elevato, le dapp di maggior successo mettono porzioni piccole del loro codice sulla blockchain, tenendo la parte consistente su altre piattaforme come, ad esempio, in cloud.

1.3.6 Account

Un account Ethereum é un'entitá con un saldo in ether (ETH) che puó inviare transazioni e ricevere criptovaluta su Ethereum. I conti sono controllabili da utenti o distribuibili come Smart Contract. Ethereum ha due tipi di account: di proprietá esterna, controllato da qualsiasi utente in possesso di chiavi private,

oppure basato su un contratto, ovvero gestito da uno smart contract. Ogni tipologia di account puó inviare e ricevere ETH e interagire con smart contract.
I conti Ethereum hanno 4 campi:

- *Nonce*: un contatore che indica il numero di transazioni inviate dal conto. Questo assicura che le transazioni siano elaborate una volta. In un account basato su contratto, questo numero rappresenta il numero di contratti creati dall'account.

- *Balance*: il numero di wei posseduti da questo indirizzo.

- *codeHash*: questo hash si riferisce al codice di un account sulla Ethereum Virtual Machine (EVM). Gli account basati su contratto contengono frammenti di codice programmati per eseguire diverse operazioni. Non é modificabile, a differenza degli altri campi dell'account. Tutti i frammenti di codice sono conservati nel database di stato sotto gli hash corrispondenti, per riferimento futuro. Questo valore dell'hash é noto come codeHash.

- *storageRoot*: detto anche hash di archiviazione. É un hash a 256 bit che codifica il contenuto dello spazio di archiviazione dell'account (una mappatura tra valori interi a 256 bit).

1.3.7 Ethereum Foundation

Ethereum Foundation é un'organizzazione nata nel 2014 dalle idee degli sviluppatori Vitalik Buterin, Gavin Wood, Jeffrey Wilcke, heikoheiko, e altri, e ha come obiettivo la gestione di tutte le attivitá di sviluppo, di ricerca e di supporto della piattaforma Ethereum. Ethereum é stata caratterizzata da una serie di prototipi e di azioni di sviluppo finanziati e gestiti da Ethereum Foundation sulla base del concetto e del progetto Proof of Concept sino al lancio del progetto Frontier Network con l'obiettivo ultimo di migliorare sicurezza e usabilitá.
Tra gli altri progetti ed iniziative della Ethereum Foundation ci sono da segnalare: il progetto Olympic (mette alla prova le performance e i limiti della rete

blockchain Ethereum); il progetto Homestead (pensato per migliorare la componente transazionale, le logiche di Gas per la gestione dei prezzi e la sicurezza); il progetto Metropolis (per semplificare l'utilizzo della Ethereum Virtual Machine e permettere agli sviluppatori di agire con una maggiore flessibilitá e velocitá); il progetto Serenity (per portare innovazione nelle logiche di gestione dell'algoritmo che gestisce il consenso di Ethereum).

1.3.8 Ethreum Classic

Ethereum Classic é costituito dai membri Ethereum che hanno deciso di dare vita a una nuova versione di Ethereum. Ethereum Classic é un network che resta al 100% compatibile con la tecnologia Ethereum, con l'aggiunta di servizi atti a migliorare la sicurezza e l'usabilitá. Ethereum Classic é basato sullo sviluppo di una blockchain non hackerabile e ha sviluppato una strategia di emissione dei token in proporzione allo sviluppo della rete nel corso del tempo. Nello specifico va ricordato che ETC é un token basato sulla piattaforma blockchain di Ethereum Classic e che il protocollo utilizzato é quello di proof-of-work. Ethereum Classic ha creato una serie di opportunitá per i miners, in quanto la potenza di calcolo necessaria a validare nuovi nodi della blockchain puó essere inferiore a quella tradizionalmente necessaria per la piattaforma Ethereum.

1.3.9 Differenze tra Etherem ed Ethereum Classic

Ethereum rappresenta la versione ufficiale della blockchain, Ethereum Classic, invece, é una blockchain che, partendo da Ethereum, si pone come una sorta di evoluzione o alternativa.

Il motivo che ha portato a questa divisione é legato ad un attacco hacker che ha colpito il progetto DAO (Decentralized Autonomous Organization) in cui furono persi 70 milioni di dollari, inducendo la comunitá di Ethereum a cambiare il codice della blockchain stessa per rimediare alle conseguenze di questo attacco. Questa decisione ha aperto una frattura sul concetto stesso di blockchain: da una parte c'erano coloro che sostenevano che le blockchain vivono sul principio della

community che decide sulle possibili evoluzioni della blockchain stessa; dall'altra c'era chi, invece, sosteneva che la blockchain non puó essere modificata e deve essere saldamente protetta da qualsiasi forma di manomissione. Ad oggi ci sono due blockchain Ethereum che lavorano in parallelo.

1.4 Non-Fungible-Token

I Non-Fungible-Token (NFT) sono uno speciale token crittografico che rappresenta l'atto di proprietá di un bene unico; gli NFT non sono quindi reciprocamente intercambiabili perché essi, a differenza delle altre criptovalute, rappresentano un pezzo unico. Un NFT puó essere un qualsiasi file digitale come un video, una foto, un file audio, un testo e piú in generale qualsiasi cosa che possa essere certificata come originale, proprio come se ci fosse la firma dell'autore. Gli NFT si appoggiano sulla blockchain (tipicamente Ethereum) che funge da registro sul quale vengono memorizzate le transazioni, come descritto in precedenza. La blockchain, quindi, registra ufficialmente l'unicitá e la non copiabilitá degli NFT.
Quindi gli NFT sono progettati per fornire un qualcosa che non puó essere copiato, ovvero la proprietá dell'opera, sebbene l'autore ne possa mantenere i diritti e la possibilitá di replicazione, come avviene con un' opera d'arte fisica; ad esempio, chiunque puó acquisire la stampa di un quadro, ma solo una persona possiede l'originale. L'opera puó, dunque, continuare a circolare sulla rete ma il titolare dell' NFT é l'unico a poter vantare i diritti dell'opera essendo certificati sulla blockchain. L'NFT costituisce quindi una prova di autenticitá e di proprietá dell'opera. La sicurezza di questi certificati deriva dal fatto che gli stessi sono ospitati su una blockchain.
Il luogo principe per custodire gli NFT é Ethereum. Se la blockchain era nata per consentire spostamenti di criptovaluta ed Ethereum é nata per consentire lo sviluppo di smart contract, gli NFT sono un'ulteriore evoluzione di questa tecnologia. In particolare, sono stati sviluppati degli smart contract per tenere traccia di chi crea, vende e compra specifiche sequenze di numeri. La blockchain di Ethereum quando ospita un NFT garantisce da un lato che questo non cambi

nel tempo, e dall'altro certifica i passaggi di proprietá tra i vari utenti degli hash gestiti dall'NFT. Chi acquista un'opera legata a un NFT, non acquista l'opera in sé, ma semplicemente la possibilitá di dimostrare un diritto sull'opera, garantito tramite uno smart contract.

Tutto comincia con una versione digitale dell'opera che viene poi compressa attraverso algoritmi di hashing in una sequenza alfanumerica; questo avviene quando, ad esempio, l'opera viene salvata su IPFS. É importante sottolineare che chi possiede il documento digitale puó facilmente calcolarne l'hash, mentre é praticamente impossibile per chiunque altro ricostruire un documento digitale a partire da un hash. Il passo successivo é la memorizzazione dell'hash sulla blockchain sfruttando le proprietá degli smart contract, con tutte le informazioni utili all'interno di un blocco. Il creatore dell'hash puó usare il token per aggiungere al suo interno il proprio hash e successivamente venderlo in cambio di un pagamento in criptovaluta. L'NFT ha al suo interno traccia delle vendite dell'hash, in modo che risulta possibile tracciare i passaggi di proprietá dell'hash tra gli utenti dimostrandone, quindi, il possesso. Questo meccanismo fornisce un prova di autenticitá e allo stesso tempo di proprietá dell'opera digitale. Il possessore dell'hash, secondo quanto riportato nell'NFT, puó dimostrare i suoi diritti senza necessitá di intermediari e senza limiti di tempo.

Per acquistare un NFT bisogna servirsi di una blockchain e di un portafoglio virtuale. Quindi memorizzare l'hash e la sua marca temporale dentro la blockchain, attraverso la creazione di un nuovo blocco, é un modo per assicurare la proprietá dell'NFT. Se si esamina cosa contiene l'NFT, si puó notare che i dati inseriti sono pochi. Soprattutto per una questione di energia impiegata (la creazione di un nuovo blocco tramite gli algoritmi di hashing é un'operazione dispendiosa dal punto di vista energetico) e di spazio disponibile (non é possibile inserire nella blockchain file di grandi dimensioni, ma solo pochi elementi, come visto in precedenza). Alcuni NFT contengono anche le condizioni contrattuali di compravendita, ma piú spesso queste si trovano solo sul sito che la intermedia.

1.4.1 Creare un NFT

Creare un NFT non é molto difficoltoso e il costo, solitamente, é limitato a quello della transazione necessaria per sigillare sulla blockchain prescelta il prodotto digitale. É possibile inserire in un NFT qualsiasi tipo di file, anche se alcune piattaforme di compravendita consentono la creazione di NFT solamente in un limitato numero di formati immagine, audio, video (come accade su Nifty Gateway). Di solito ci sono anche limiti di dimensione; ad esempio su Opensea, uno dei marketplace piú utilizzati per la compravendita di NFT, il limite per la dimensione di un file é di 100 MB, ma si consiglia di limitare la dimensione a 40 MB. Il costo della creazione dell' NFT dipende dal costo della transazione su Ethereum (il costo del gas, che oscilla tra 10 e 100 euro). All'inizio spesso viene richiesta una doppia transazione, la prima di inizializzazione del wallet Ethereum e la seconda per la creazione dell' NFT vero e proprio. In seguito, il costo puro della creazione dell'NFT é pari al costo del gas su Ethereum. Una volta che il wallet é accreditato, alcune piattaforme consentono di vendere NFT senza pagare ulteriori commissioni; l'effettiva creazione dell'NFT avverrá solo una volta che questo sará venduto sul marketplace. La vendita dell'opera puó avvenire pagando sia un prezzo fisso sia all'asta.

1.4.2 Standard Token

Uno dei molti standard di sviluppo di Ethereum si concentra sulle interfacce dei token. Questi standard aiutano a garantire che gli Smart Contract rimangano componibili, in modo che, ad esempio, quando un nuovo progetto emette un token, esso rimanga compatibile con gli scambi decentralizzati giá esistenti.
Alcuni degli standard token piú utilizzati su Ethereum sono:

- ERC-20: Un'interfaccia standard per token fungibili (intercambiabili), come i token di voto, i token di staking o le valute virtuali.
- ERC-721: Un'interfaccia standard per token non fungibili, come un atto relativo a opere d'arte o canzoni.

- ERC-777: consente alle persone di creare funzionalitá extra sui token come un contratto di mescolamento per una migliore privacy della transazione o una funzione di recupero d'emergenza per salvarti se perdi le tue chiavi private.

- ERC-1155: consente scambi e aggregazioni di transazioni piú efficienti, risparmiando sui costi. Questo standard del token consente di creare token d'utilitá (come $BNB o $BAT) e token non fungibili come CryptoPunks.

1.4.3 Standard ERC-721

Quando si parla di uno standard ci si riferisce all'identificazione univoca di un token rispetto ad altri dello stesso smart contract. Per ottenere un token non fungibile si utilizza uno standard denominato "ERC-721" (Ethereum Request for Comments 721) . Quest'ultimo descrive come creare token non fungibili sulla blockchain di Ethereum ed é possibile immaginarlo come una sorta di template che gli utenti compilano in modo totalmente gratuito e autonomo affinché venga generato un NFT. Lo standard ERC-721 é impiegato in mercati costantemente in crescita quali, ad esempio, quelli relativi agli oggetti di gioco online, come CryptoKitties. Un nuovo tipo di standard, ERC-721R, permette all'utente un diritto di ripensamento rispetto al proprio acquisto e, quindi, ottenere un rimborso corrispondente al prezzo dell'NFT coniato.

1.4.4 Il problema del copyright

La maggior parte degli NFT rappresentano un file di metadati che é stato codificato utilizzando un'opera che puó o meno essere soggetta alla protezione del copyright o potrebbe anche appartenere alla categoria di pubblico dominio. Cé sempre un alone di confusione ed incertezza rispetto ai diritti che gli acquirenti acquisiscono quando comprano un NFT. Solitamente si pensa di comprare l'opera d'arte sottostante e tutti i diritti ad essa associati. In realtá, si stanno semplicemente acquistando i metadati associati al bene e non all'opera stessa. Il copyright potrebbe entrare in gioco, almeno per alcuni NFT; questo perché,

mentre la maggior parte degli NFT non comporta un trasferimento di diritti, in alcuni casi il venditore si offre di trasformare il token in un effettivo trasferimento della proprietá del diritto d'autore dell'opera originale.
In qualche modo, tutti gli NFT potrebbero essere visti come una forma di registrazione. Questa idea si imbatte rapidamente in problemi pratici, tra cui il fatto che chiunque abbia sufficienti conoscenze tecniche e i mezzi appropriati é in grado di generare il proprio token, il quale potrebbe includere qualsiasi informazione inserita dall'autore.
Questo significa che chiunque puó fare dichiarazioni di proprietá errate e scriverle nella blockchain. Inoltre, se consideriamo una licenza come un documento legale che permette a un utente di eseguire una prestazione, la quale sarebbe limitata dal diritto d'autore, allora questo puó essere ottenuto anche con un NFT. Tuttavia, ad oggi, le principali piattaforme e progetti collezionabili non offrono licenze di alcun tipo, oppure presentano termini e condizioni contraddittorie.
Infine c'é il potenziale problema della violazione del copyright. Alcuni artisti sostengono che le loro opere siano state coniate come NFT senza il loro permesso. La questione é comunque molto intricata e di difficile interpretazione; dal punto di vista del copyright, é difficile vedere come il conio di un NFT possa essere considerato una violazione del diritto d'autore. Poiché l'NFT non rappresenta l'opera in sé, ma un hash (e quindi una stringa alfanumerica) generato in relazione ad essa, il file risultante non potrebbe essere considerato come una riproduzione o anche un adattamento dell'opera.
In generale, affinché la violazione abbia luogo, devono essere soddisfatti tre requisiti:

- l'utente che ha coniato l'NFT deve aver tratto profitto da uno dei diritti esclusivi dell'autore senza autorizzazione;
- deve verificarsi una connessione causale tra la NFT e l'opera originale, ovvero l'NFT deve essere stato creato direttamente dall'originale;
- l'opera nel suo complesso, o parte sostanziale di essa, deve essere stata copiata.

É difficile vedere come un NFT possa soddisfare questi requisiti. I diritti esclusivi di cui gode l'autore di un'opera coprono la riproduzione, la pubblicazione, il prestito e il noleggio, la pubblica riproduzione, l'adattamento, la comunicazione al pubblico e l'autorizzazione a eseguire uno qualsiasi dei suddetti diritti. Solo il diritto di comunicazione al pubblico potrebbe essere violato attraverso un link in un NFT, poiché, in tal caso, si verificherebbe una connessione causale tra l'NFT in questione e l'opera. Tuttavia, poiché un NFT é semplicemente codice e non una riproduzione sostanziale dell'opera, non violerebbe tutti gli altri diritti.

1.4.5 Esempi di NFT piú redditizi

Secondo le ultime statistiche, risalenti all'inzio del 2022, il mercato degli NFT oggi vale piú di $40 miliardi e ha registrato volumi di vendita pari a circa $24.9 miliardi nel 2021. La maggior parte degli scambi finora sono avvenuti su OpenSea, il piú grande marketplace di NFT, con vendite del valore di $6.5 miliardi, seguito a ruota da Nifty Gateway.
Di seguito sono elencati gli NFT piú costosi:

1. The Merge, di Pak – 91,8 milioni di $;
2. Everyday, 5000 days, di Beeple – 69 milioni di $;
3. Serie da 9 CryptoPunks – 16,9 milioni di $;
4. Hashmasks – 16 milioni di $;
5. CryptoPunk #7523 – 11,7 milioni di $;
6. CryptoPunk #7804 – 7,5 milioni di $;
7. Crossroads, di Beeple – 6,6 milioni di $;
8. War Nymph, di Grimes – 6 milioni di $;
9. NFT primo tweet della storia – 2,9 milioni di $;
10. CryptoPunk #6865 – 1,5 milioni di $.

Capitolo 2

Tecnologie utilizzate

Per lo sviluppo del progetto si é deciso di creare un'applicazione web decentralizzata che potesse funzionare sulla blockchain Ethereum. Al fine di raggiungere tale scopo sono stati utilizzati alcuni strumenti, fondamentali nello sviluppo di applicazioni decentralizzate, di cui verranno approfonditi gli aspetti peculiari in questo capitolo.

2.1 ReactJS

React é una libreria open source scritta in Javascript per la creazione di pagine web. React puó essere utilizzato come base nello sviluppo di applicazioni a pagina singola ma é utilizzabile anche su mobile tramite React Native. Tuttavia, React si occupa solo del rendering dei dati sul DOM (document object model), pertanto la creazione di applicazioni React piú complesse richiede l'uso di librerie aggiuntive per ottenere, ad esempio, il routing.
Le caratteristiche principali di React sono:

- componenti: il codice di React é costituito da entitá denominate componenti. I componenti possono essere sottoposti a rendering su un particolare elemento nel DOM usando la libreria React DOM. Quando si esegue il rendering di un componente, si possono passare argomenti noti come "props".I due modi principali per dichiarare i componenti in React sono: tramite componenti funzionali, che vengono dichiarati con una funzione che restituisce

alcuni JSX; e tramite componenti basati su classi, che vengono dichiarati usando classi ES6; questi ultimi sono anche noti come componenti "stateful", perché il loro stato puó contenere valori in tutto il componente e puó essere passato ai componenti figlio tramite props.

- DOM virtuale : React crea una cache della struttura dati in-memory, calcola le differenze risultanti e quindi aggiorna il DOM visualizzato dal browser. Ció consente al programmatore di scrivere codice come se l'intera pagina fosse renderizzata su ogni modifica, mentre le librerie React eseguono il rendering solo dei sottocomponenti che cambiano effettivamente, senza ricaricare l'intera pagina.

- JSX: JSX, o JavaScript XML, é un'estensione della sintassi del linguaggio JavaScript. Simile nell'aspetto all'HTML, JSX fornisce un modo per strutturare il rendering dei componenti usando una sintassi familiare a molti sviluppatori. I componenti di React sono in genere scritti usando JSX, sebbene possano essere scritti in JavaScript puro.

- attributi: JSX offre una gamma di attributi di elemento progettati per rispecchiare quelli forniti da HTML. Possono essere passati al componente anche attributi personalizzati. Tutti gli attributi verranno ricevuti dal componente come props.

- espressioni Javascript: le espressioni JavaScript (ma non le istruzioni) possono essere utilizzate all'interno di JSX con parentesi graffe.

- dichiarazioni condizionali: le istruzioni if-else non possono essere utilizzate all'interno di JSX ma é possibile utilizzare espressioni condizionali.

- hooks: gli hooks sono funzioni che consentono agli sviluppatori di "agganciarsi" allo stato di React e alle caratteristiche del ciclo di vita dei componenti delle funzioni. Rendono i codici leggibili e facilmente comprensibili. Gli hooks permettono di utilizzare React senza classi. Alcuni degli hooks utilizzati frequentemente nello sviluppo dell'applicazione sono useState e

useEffect, utili per il controllo degli stati e degli effetti collaterali nei componenti React. Inoltre gli hooks devono essere chiamati solo al livello piú alto e solo dai componenti di funzione React.

2.2 MetaMask

MetaMask é un software wallet, non-custodial (solo l'utente ha la chiave privata del wallet, senza delegarla a terze parti) creato da ConsenSys, una delle societá leader nel settore dello sviluppo di software blockchain, nel 2016; Metamask puó essere installato come plugin all'interno di numerosi browser tra cui Chrome, Brave e Mozilla Firefox. Nel tempo MetaMask é diventato il wallet piú utilizzato, nonché il piú sicuro e semplice, per interagire con il Web3 e diverse blockchain, tra cui Ethereum, Polygon e Binance.
MetaMask mette in comunicazione le dApp con la blockchain consentendo anche agli utenti meno esperti di utilizzare facilmente le applicazioni decentralizzate e di operare con gli Smart Contracts. Grazie alle sue funzionalitá permette di eseguire qualsiasi operazione di trading senza avviare, ad esempio, il software di Ethereum: basta infatti collegarsi alla rete internet, aprire il proprio browser e accedere all'estensione MetaMask.
L'installazione del plugin di MetaMask é semplice e necessita di pochi passaggi; tra i piú importanti ci sono l'impostazione della password e la frase seed, una sequenza di 12 parole generata automaticamente che viene inserita durante l'installazione. Quest' ultima é di estrema importanza e deve essere conservata con attenzione in quanto é l'elemento che puó far recuperare i fondi in caso di perdita dell'account o inaccessibilitá del dispositivo su cui é installato MetaMask.
Tra le funzionalitá messe a disposizione da MetaMask ci sono:

- possibilitá di inviare/ricevere e detenere criptovalute;
- possibilitá di importare nuovi token;
- possibilitá di accedere e connettersi a tutte le dApps;

- possibilitá di aggiungere nuove blockchain in base al token di proprio interesse.

2.3 HardHat

Hardhat é un ambiente di sviluppo per compilare, distribuire, testare ed eseguire debug del software Ethereum. Aiuta gli sviluppatori a gestire e automatizzare le attivitá ricorrenti inerenti al processo di creazione di smart contract e dApp. Hardhat é integrato con HardHat Network, una rete Ethereum locale progettata per lo sviluppo; le sue funzionalitá si focalizzano sul debug di smart contract.
Hardhat puó essere messo in esecuzione su un server locale per effettuare il testing; infatti da un terminale apposito, ad esempio su Visual Studio Code, con il comando npx hardhat node possono essere visualizzati una serie di account falsi rappresentati da una chiave pubblica e una privata e un ammontare di ETH fisso posto a 10000. Da questo stesso terminale é possibile visualizzare anche tutte le transazioni effettuate e i blocchi creati come si vede in figura 2.1.

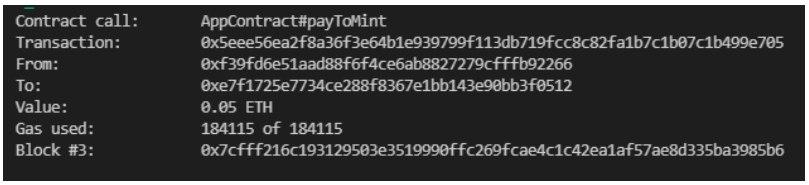

Figura 2.1: Esempio di blocco con Hardhat

Altre funzionalitá fondamentali di HardHat sono la distribuzione e compilazione di smart contract, di cui si discuterá dettagliatamente in seguito, fondamentali per ottenere, rispettivamente, l'ABI relativo a quest'ultimo e l'indirizzo necessario per far funzionare qualsiasi applicazione sulla blockchain Ethereum.

2.4 Solidity

Solidity é un linguaggio di programmazione di alto livello orientato agli oggetti, staticamente tipato, utilizzato per implementare smart contracts su varie blockchain. É stato sviluppato da Christian Reitwiessner, Alex Beregszaszi e altri

sviluppatori che giá avevano contribuito alla creazione di Ethereum. I programmi scritti in solidity funzionano sulla Ethreum Virtual Machine. Solidity usa una sintassi simile a ECMAScript (Javascript) ma con tipizzazione statica. Solidity é diverso dagli altri linguaggi che funzionano su EVM innanzitutto perché supporta variabili complesse per i contratti, inclusi struct e altre strutture gerarchiche arbitrarie. I contratti scritti in Solidity supportano l'ereditarietá multipla.
Altra nota importante é l'introduzione dell'ABI (Application Binary Interface, definisce l'interfaccia tra il sistema operativo e le proprie applicazioni a livello di linguaggio macchina) che facilita l'inserimento all'interno di un singolo contratto di piú funzioni type-safe.

2.4.1 OpenZeppelin

Uno strumento molto utile che facilita la stesura di smart contracts é OpenZeppelin. OpenZeppelin é una collezione di smart contract che implementa funzionalitá che sono comunemente utilizzate nella stesura degli smart contracts stessi, permettendo quindi di utilizzare codice scritto e verificato da altri programmatori. OpenZeppelin implementa i token piú utilizzati come ERC20 e ERC721. Uno dei vantaggi maggiori nell'utilizzo di OpenZeppelin é la sicurezza; infatti, essendo una libreria open source il codice é sempre aggiornato rendendolo quindi privo di vulnerabilitá.

2.5 Pinata Cloud

Pinata cloud é un servizio di *pinning* (letteralmente fissare) che permette agli utenti di conservare i dati sul network IPFS.
L'*InterPlanetary File System* (IPFS) é un protocollo di comunicazione e una rete peer-to-peer per l'archiviazione e la condivisione di dati in un file system distribuito. L'IPFS utilizza uno spazio di archiviazione associativo per identificare univocamente ogni file in uno spazio di nomi globale (CID) che connette tutti i dispositivi di calcolo in modo da avere un sistema decentralizzato senza alcun controllo centrale. Pinata cloud viene utilizzato quasi esclusivamente per

il salvataggio degli NFT; infatti, una volta caricati su IPFS, ad ogni NFT viene assegnato una stringa alfanumerica ottenuta tramite una funzione hash che funge da identificativo univoco all'interno della rete. Quindi in ogni momento é possibile recuperare il proprio NFT postponendo l'identificativo univoco a `ipfs://`.

2.6 Node.JS

Node.JS é una piattaforma event-driven ideata da Ryan Dahl nel 2009 per lo sviluppo di applicativi web scalabili. Tra i motivi principali che portarono alla creazione di questa piattaforma si citano i limiti dei web server piú utilizzati e famosi, come Apache HTTP Server, nel gestire numerose richieste contemporanee e nell'eccessiva richiesta di risorse al sistema.
Node.JS ha un modello di networking che, piuttosto che essere basato su processi concorrenti, é basato sull' I/O event-driven. Node.JS, quindi, richiede al sistema operativo di ricevere notifiche a seguito di determinati eventi e attende in uno stato di sleep fino al giungere di una notifica. A questo punto torna attivo, processa il comando ricevuto e fornisce l'output per poi tornare in uno stato di sleep. Tale approccio, definito event loop, consente un'ottima scalabilitá che processi o thread non consentirebbero di ottenere ed inoltre consente di gestire migliaia di richieste contemporaneamente. Un'altra caratteristica molto apprezzata di Node.JS é la modularitá. La libreria, infatti, é composta da numerosi moduli che vanno ad integrare ed implementare nuove funzionalitá per la piattaforma. Il gestore dei pacchetti di Node.JS é npm, utile, tra l'altro, per avviare il server in locale ed eseguire il testing.

Capitolo 3

Un'applicazione decentralizzata per la creazione e lo scambio di Non-Fungible-Token

Come giá anticipato, l'applicazione decentralizzata oggetto del lavoro di tesi permette di validare dei documenti pdf sulla blockchain, sottoforma di NFT, e di scambiarli con altri utenti in possesso del wallet Metamask. Nello specifico, gli strumenti utilizzati per sviluppare l'applicazione sono:

- **React.js**: per sviluppare il frontend dell'applicazione web decentralizzata é stato usato React, le cui caratteristiche principali sono descritte nel capitolo precendente;
- **Hardhat**: Hardhat permette di simulare una blockchain in locale mettendo a disposizione una serie di account con un ammontare di Ether fisso; per mettere in esecuzione la blockchain di prova si utilizza il comando `npx hardhat node`. Da notare che la blockchain simulata da hardhat funge da backend e per comunicare con essa si utilizzano le funzioni definite nello smart contract;
- **Metamask**: come visto in precedenza, Metamask permette agli utenti di interagire con il Web3, quindi con la blockchain, creando un proprio account; é possibile importare gli account creati con hardhat su Metamask

selezionando come rete quella di test fornita da hardhat (nel caso specifico dell'applicazione la rete locale sulla porta 8545); in questo modo si possono creare e scambiare NFT, dato che si avrá una grande quantitá di Ether a disposizione;

- **Pinata**: Pinata viene usato per salvare i file su IPFS; a questo scopo é stato creato un account apposito in modo da utilizzare le API messe a disposizione da Pinata e avere le chiavi pubblica e privata che saranno utilizzate per caricare i file, come viene spiegato nella sezione in cui verranno descritte le funzioni piú importanti implementate.

Nei paragrafi successivi, dunque, si mostreranno gli artefatti e le scelte implementative intraprese al fine di sviluppare al meglio l'applicazione.

3.1 Use Case diagram

In questo paragrafo si descriverá lo *Use Case diagram*. UML (Unified Modeling Language, linguaggio di modellazione e specifica) mette a disposizione questo strumento per facilitare la descrizione dei servizi e delle funzioni offerti da un software. Per facilitarne la comprensione, la descrizione dello Use Case diagram, mostrato in figura 3.1, verrá corredata con le schermate dell'applicazione.

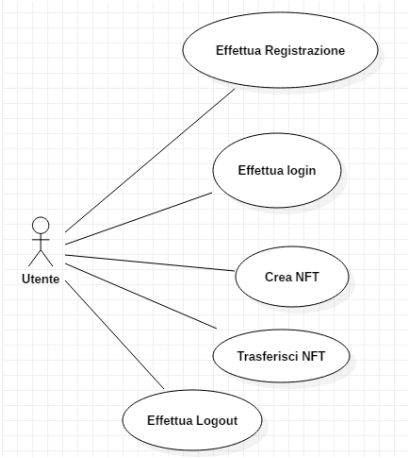

Figura 3.1: diagramma dei casi d'uso

Di seguito verranno passati in rassegna tutti i casi d'uso:

- **caso d'uso Effettua Registrazione:**

  dato che per interagire con il *Web3* e la *blockchain Ethereum* é necessario l'utilizzo di un *wallet*, la registrazione altro non é che il download e l'installazione dell'estensione per il browser *MetaMask*.

Figura 3.2: Schermata principale dell'applicazione

Infatti, cliccando su *Sign in* (in figura 3.2), si verrá reindirizzati sulla pagina di download di MetaMask. Dalle considerazioni appena fatte, ne risulta che l'estensione di MetaMask é fondamentale per avere accesso all'applicazione; infatti, se l'estensione non é installata, verrá mostrato un messaggio di errore nel quale si chiede di installare Metamask.

- **caso d'uso Effettua Login:**

  cliccando su *Login*, si aprirá una finestra di notifica che permetterá all'utente di effettuare il login su MetaMask (figura 3.3).

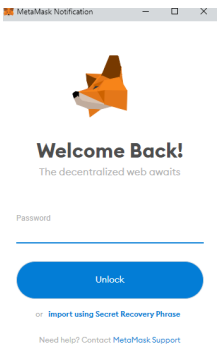

Figura 3.3: Login con Metamask

A questo punto sará necessario entrare con le proprie credenziali di MetaMask. Ci sono due modi per accedervi: inserendo la password impostata

in fase di registrazione, oppure inserendo la frase seed (cliccando su *import using Secret Recovery Phrase*) che viene fornita da Metamask stesso sempre durante la fase di registrazione. Una volta effettuato l'accesso si aprirá la schermata principale come nelle due figure sottostanti a seconda del caso in cui l'utente appena loggato possieda NFT (figura 3.5) o meno (figura 3.4). Il login puó essere effettuato anche cliccando su *Get Started*.

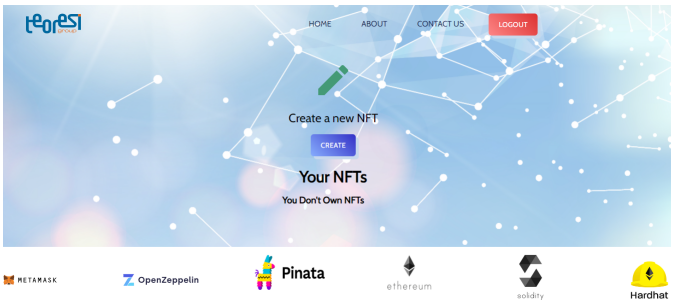

Figura 3.4: Home Page in cui l'utente non possiede NFT

Figura 3.5: Home Page con un NFT posseduto

- **caso d'uso Crea NFT:** dalla schermata principale é possibile sia creare un nuovo NFT, sia trasferire gli NFT che si posseggono ad altri account (figure 3.4 e 3.5). Cliccando su *Create* si verrá indirizzati verso una nuova pagina, come si puó vedere nelle figure 3.6 e 3.7.

  Da qui l'utente puó selezionare un file pdf da locale tramite il pulsante *Choose* in figura 3.6 (il file deve essere rigorosamente pdf, qualsiasi tentativo di inserire file di tipo diverso fallisce mostrando un messaggio di errore), caricarlo su Pinata Cloud (pulsante *Upload* in figura 3.6), e creare il nuovo NFT cliccando su *Mint* (figura 3.7). Per creare un nuovo NFT é necessario

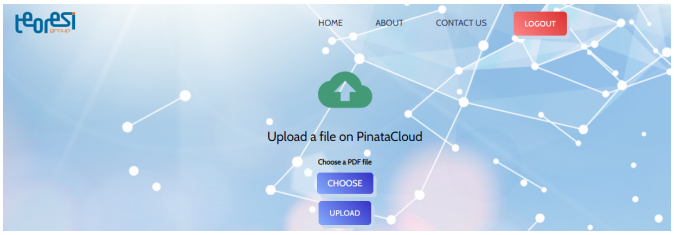

Figura 3.6: Pagina per la creazione di un nuovo NFT, parte 1

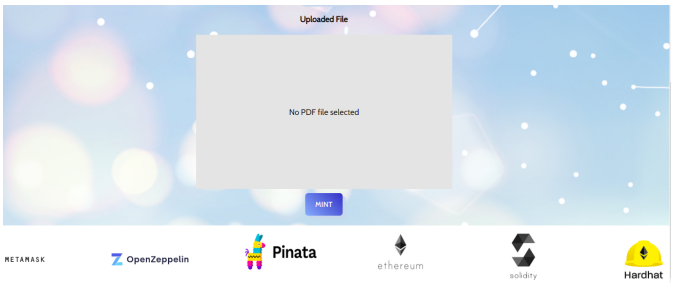

Figura 3.7: Pagina per la creazione di un nuovo NFT, parte 2

interagire con MetaMask e in particolare con lo smart contract. Infatti, cliccando su *Mint*, si aprirá una finestra di MetaMask; da quest'ultima

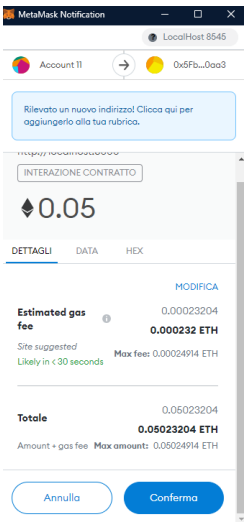

Figura 3.8: Notifica di Metamask per la creazione di un nuovo NFT

(figura 3.8) si puó cliccare su *Conferma* e creare il nuovo NFT; da notare che per la creazione di un nuovo NFT c'é da pagare un costo fisso, pari a 0.05 ETH, a cui si somma il costo del gas, che varia di volta in volta. Dopo aver creato l'NFT, si verrá indirizzati di nuovo sulla pagina principale.

- **caso d'uso Trasferisci NFT:** cliccando su *Transfer* si aprirá un modale sulla pagina principale come in figura 3.9; sul modale sono descritte le ope-

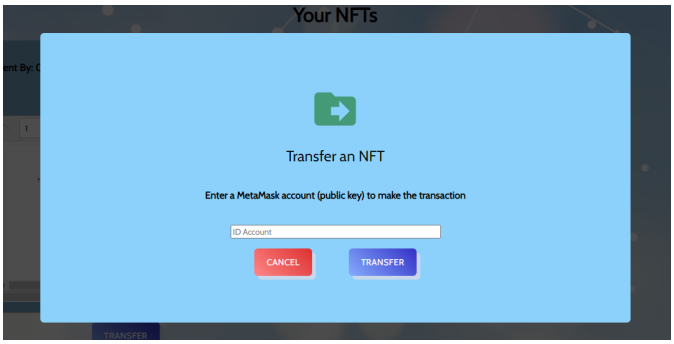

Figura 3.9: Modale per il trasferimento di un NFT

razioni da effettuare per trasferire un NFT; in particolare bisogna inserire l'indirizzo di un account di Metamask, corrispondente alla chiave primaria, e cliccare su *Transfer*. Anche per l'operazione di trasferimento di un NFT é necessario interagire con Metamask e con lo smart contract; infatti cliccando su *Transfer* all'interno del modale, si aprirá una finestra di Metamask per confermare il trasferimento, pagando solo le commissioni per il gas, a differenza della creazione in cui si paga un costo extra. Dopo aver effettuato il trasferimento si ritornerá sulla Home page.

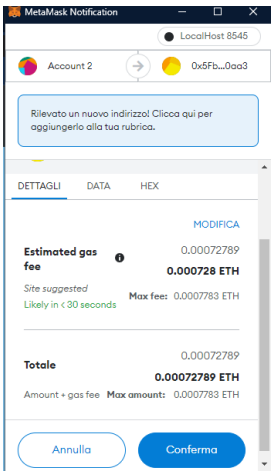

Figura 3.10: Notifica di Metamask per il trasferimento di un NFT

- **caso d'uso Effettua Logout:** per effettuare il logout basta cliccare su *Logout*; quest'azione porterá l'utente sulla prima pagina (figura 3.2).

3.2 System design

In questo paragrafo si mostrerá l'architettura dell'applicazione, vedendo nel dettaglio il pattern architetturale utilizzato e ponendo l'accento su alcuni dei moduli e librerie piú importanti di cui si é fatto uso nello sviluppo dell'applicazione.

3.2.1 MVC

Model View Controller é un pattern architetturale utilizzato per separare la logica di presentazione dei dati dalla logica di business.
Il pattern si basa sulla separazione dei compiti fra i componenti software distinguibili in tre ruoli principali (come si vede in figura 3.11):

- il *model* fornisce i metodi per accedere ai dati utili all'applicazione;
- la *view* visualizza i dati contenuti nel model e si occupa dell'interazione con gli utenti;
- il *controller* riceve i comandi dell'utente (solitamente attraverso la view) e li attua modificando lo stato degli altri due componenti, manipolando il model e aggiornando di conseguenza la view.

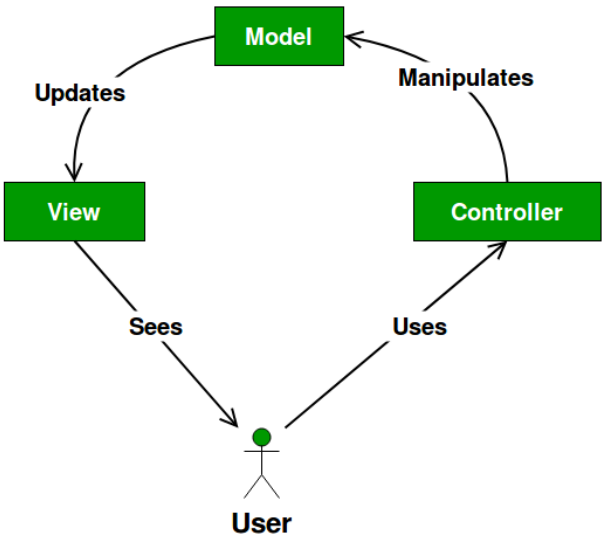

Figura 3.11: schema design patter MVC

Nonostante React.js non sia nato per adattarsi al meglio al pattern MCV come concepito originariamente, essendo quest'ultima una libreria piú dedicata alla presentazione della parte grafica, le ultime evoluzioni e versioni vanno proprio nella direzione di questo design pattern. Se nelle versioni precedenti le componenti di React mescolavano la logica di rendering con il rendering (comportandosi contemporaneamente come View e Controller), con l'introduzione degli *hooks* (principalmente UseState e UseEffect, di cui si é fatto ampio utilizzo nello sviluppo del frontend) i compiti sono stati separati; infatti, gli hooks si comportano come Controller e i componenti come View.

Detto questo, nello sviluppo dell'applicazione si é cercato quanto piú possibile di seguire il pattern MVC. Proprio in quest'ottica, e sfruttando gli strumenti messi a disposizione da React, le componenti grafiche (che possono essere considerate come view) sono state scritte utilizzando il linguaggio CSS (Cascading Style Sheets) e la parte del controller, scritta in Javascript, sfruttando gli hooks all'interno delle varie pagine e componenti. Inoltre, le pagine Javascript comunicano anche con il *backend*, ovvero la blockchain, tramite funzioni definite all'interno dello smart contract.

Come giá visto nella parte introduttiva, i dati che si possono registrare su un blocco della blockchain sono limitati per questioni di spazio e costi di validazione del blocco, ragion per cui i file pdf sono salvati su IPFS (Pinata Cloud); una volta salvati su Pinata Cloud, ad ogni file viene assegnato un identificativo unico, il che li rende a tutti gli effetti degli NFT, ma é la blockchain a fare da *backend* registrando tutte le transazioni effettuate, ovvero ogni chiamata a funzioni dello smart contract, quindi sia quando viene creato un nuovo NFT, sia quando quest'ultimo viene trasferito. Infatti, ogniqualvolta viene creato o trasferito un NFT, vi é l'aggiunta di un nuovo blocco alla blockchain, salvando, nel caso dell'applicazione sviluppata, l'indirizzo del mittente, l'indirizzo del destinatario e l'identificativo corrispondente all'NFT (ottenuto da Pinata).

L'utente che interagisce con l'applicazione cliccando, ad esempio, sui pulsanti *MINT* o *TRANSFER* (figura 3.7 e 3.5), attiva le funzioni dello smart contract che comunicano direttamente con la blockchain. In generale, a differenza del clas-

sico uso del pattern MVC, dove é il *controller* a comunicare direttamente con il *model*, nel caso di un'applicazione decentralizzata c'é un "livello" in piú; infatti, il *controller* comunica con lo smart contract che a sua volta comunica con la blockchain, senza tralasciare il fatto che i file nella loro completezza sono salvati su IPFS.

Come si evince dalla figura 3.12, le pagine Javascript, nel quale si utilizzano gli

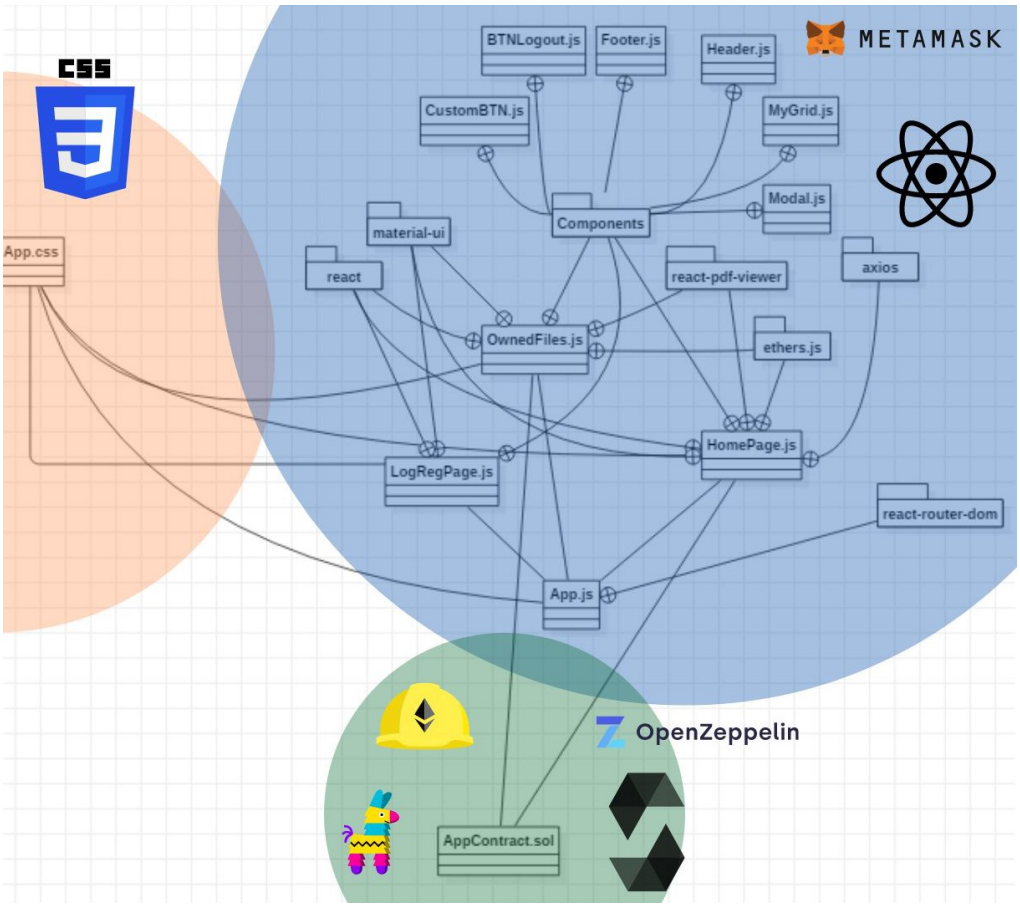

Figura 3.12: System design dell'applicazione

hooks, con i moduli piú importanti, formano la parte di *controller*; in particolare, dalla pagina `App.js` grazie all'utilizzo della libreria `react-router-dom` si puó navigare verso tutte le pagine dell'applicazione: la pagina iniziale dell'applicazione da cui ci si puó loggare o registrare `LogRegPage.js`, la pagina principale dell'applicazione, dove sono presenti tutti gli NFT posseduti dall'utente loggato, `OwnedFiles.js` e la pagina per la creazione di un nuovo NFT `HomePage.js`. Nel file `App.css` sono definiti tutti gli stili e le classi utilizzati nelle pagine descritte

in precedenza, come, ad esempio, il modale e lo sfondo. `AppContract.sol` rappresenta lo smart contract scritto in Solidity; nella sezione dove si descrivono il codice e le funzioni piú importanti, si vedrá come il controller interagisce con lo smart contract che a sua volta comunica con la blockchain, al fine di effettuare le operazioni chiave di creazione e trasferimento di un NFT.

3.2.2 Moduli, librerie ed API utilizzati

In questo paragrafo si fará una panoramica dei moduli e librerie piú importanti utilizzati nello sviluppo dell'applicazione.

- `react-router-dom`: é una delle principali librerie per il routing per React permettendo di navigare tra le pagine di un'applicazione web. React Router DOM consente di implementare il routing dinamico in un'app web. A differenza della tradizionale architettura di routing, in cui viene gestito in una configurazione esterna all'app in esecuzione, React Router DOM facilita il routing basato sui componenti in base alle esigenze dell'applicazione e della piattaforma. Di seguito viene mostrato un esempio di come é utilizzato `react-router-dom`:

```
1        [...]
2        <Route path="/" element={<LoginRegPage />}/>
3        <Route path="/OwnedFiles" element={<
            OwnedFiles />}/>
4        <Route path="/HomePage" element={<HomePage
            />}/>
5        [...]
```

  dove *path* rappresenta il percorso partendo da `localhost3000::`, e *element* rappresenta le pagine verso il quale si puó navigare.

- `ethers.js`: la libreria `ethers.js` si pone lo scopo di essere una libreria completa e compatta per interagire con la blockchain Ethereum e il suo

ecosistema. La libreria mette a disposizione alcune caratteristiche interessanti quali: la sicurezza delle chiavi private, la connessione ai nodi Ethereum attraverso JSON-RPC, INFURA, Etherscan, Alchemy, Cloudflare e MetaMask, la possibilitá di creare oggetti Javascript partendo dall' ABI di un contratto attraverso delle metaclassi, l'importazione e l'esportazione di JSON wallets e HD (Hierarchical Deterministic) wallets (come Metamask). Nei paragrafi successivi si vedrá come é possibile interagire con il Web3 e il wallet Metamask con l'utilizzo di `ethers.js`.

- `axios`: é un promise-based client HTTP per Node.js e il browser; permette di effettuare richieste XMLHttpRequests dal browser e richieste http da node.js, supporta le API Promise e fornisce supporto lato client contro attacchi di richiesta intersito.

- `react-pdf-viewer`: permette la visualizzazione di file pdf, mettendo a disposizione diversi stili e plugin, tra cui la possibilitá di scaricare, stampare ed ingrandire il documento pdf.

- `Material-ui`: permette di dare uno stile ai componenti di React; si é fatto uso anche di `makeStyles` per la creazione, tra gli altri, di pulsanti, utilizzando componenti di CSS all'interno di file Javascript.

- `hook`: come visto in precedenza, gli hooks sono funzioni che consentono agli sviluppatori di "agganciarsi" allo stato di React e alle caratteristiche del ciclo di vita dei componenti delle funzioni. Nell'applicazione si fa uso di useState per controllare lo stato delle varibili e useEffect per eseguire effetti collaterali nei componenti:

```
1          [...]
2          const [file, setFile] = useState();
3          [...]
4          const [strutturaTransazioni,
              setStrutturaTransazioni] = useState([]);
5          [...]
```

```
6           const [accountConnesso, setAccountConnesso]
                = useState();
7           [...]
```

3.2.3 La cartella *Components*

All'interno della cartella *Components* sono presenti i vari componenti riutilizzati piú volte nella creazione delle pagine dell'applicazione. In questo paragrafo si passeranno in rassegna i vari componenti:

- `CustomBTN` e `BTNLogout`: questi pulsanti sono utilizzati piú volte all'interno dell'applicazione. In particolare `CustomBTN` é stato usato per quasi tutti i pulsanti. Con l'ausilio della libreria `makeStyles` viene dato uno stile ad entrambi i pulsanti, come si evince dalle figure presentate nella spiegazione dello Use Case diagram. Inoltre, ad entrambi i pulsanti, vengono passati in input degli argomenti (`props`) come il testo da visualizzare, il link ad un'altra pagina e la funzione `onClick()` nel caso siano necessari.

- `Footer`: il footer é presente in ogni pagina mostrando le icone degli strumenti utilizzati nello sviluppo dell'applicazione; inoltre, tutte le icone sono cliccabili e collegate alle pagine ufficiali.

- `Header`: presente nella home page e nella pagina per la creazione di un nuovo NFT (figure 3.4 e 3.6). Per dare uno stile agli elementi che compongono l'header sono stati creati degli appositi componenti CSS; in questo modo é stato possibile, ad esempio, dare l'effetto di *hoover* agli elementi.

- `MyGrid`: é utilizzato per creare dei componenti formati da una icona e un testo esplicativo che vengono passati in ingresso come argomenti.

- `Modal`: viene utilizzato quando si trasferisce un NFT (figura 3.9). Nel Modal viene definito uno stile sia per la parte del modale (sovrapposto alla Home Page), con icone, pulsanti e testi esplicativi, sia per la parte di background che viene offuscata. Vengono, inoltre, passati come argomenti l'indirizzo

del contratto, necessario per trasferire l'NFT, l'identificativo dell'NFT da trasferire (il CID fornito da Pinata), e la funzione `onClose` per chiudere.

3.3 Implementazione delle funzioni piú importanti

In questo paragrafo verranno passate in rassegna alcune delle funzioni fondamentali che interagiscono con i principali strumenti utilizzati nello sviluppo dell'applicazione, come Metamask, Pinata Cloud, lo smart contract e la blockchain simulata da Hardhat, mostrandone ed esponendone l'implementazione.

3.3.1 Login con metamask

```
1   async function loginWithMetaMask() {
2       if (!window.ethereum) {
3           alert("Click Sign IN to download MetaMask")
              ;
4           return;
5       }
6       const accounts = await window.ethereum.request(
          { method: 'eth_requestAccounts' })
7       .catch((e) => {
8           console.error(e.message)
9           return
10      })
11      if (!accounts) { return }
12      window.userWalletAddress = accounts[0]
13      window.open ("/OwnedFiles", "_self")
14  }
```

Il login con Metamask viene effettuato attraverso il metodo `eth_requestAccounts` fornito dalle API di Metamask. Questa é una funzione asincrona che permette a Metamask di restare in attesa fin quando un account non si sia connesso, inviando una richiesta RPC (chiamata di procedura remota) ad Ethereum. L'oggetto `window.ethereum` viene *"injected"* da Metamask nelle pagine web. Se il plugin di Metamask non é installato viene mostrato l'alert *Click Sign IN to download MetaMask*. Una volta che l'accesso é andato a buon fine, l'account viene salvato in una variabile e si passa sulla Home Page, giá vista in figura 3.4.

3.3.2 Caricamento di un file su Pinata Cloud

In precendenza si é visto come l'IPFS sia il luogo principe per conservare i propri NFT. Di seguito si vedrá come caricare un file su Pinata Cloud.

```
1    async function caricaFile(file) {
2        const formData = new FormData()
3        formData.append("file", file)
4        const pinataKey = "d49e40a938354f763485"
5        const pinataSecretKey = "6ab4e27ec144914a6aba05
             5ec37fac1f449a0684626c58a0a9428ba6ad722f38"
6        const url = `https://api.pinata.cloud/pinning/
             pinFileToIPFS`
7        const response = await axios.post(
8            url,
9            formData,
10           {
11               maxContentLength: "Infinity",
12               headers: {
13               "Content-Type": `multipart/form-data;
                     boundary=${formData._boundary}`,
14               'pinata_api_key': pinataKey,
```

```
15                  'pinata_secret_api_key':
                        pinataSecretKey
16              }
17          }
18          )
19          setVariabile_IpfsHash(response.data.IpfsHash);
20      }
```

La funzione prende in ingresso il file pdf scelto da locale. Per caricare un file su Pinata sono necessari:

- la chiave pubblica e la chiave privata associate ad ogni account registrato su Pinata;
- l'url, che corrisponde all'endpoint `/pinning/pinFileToIPFS`, una funzione delle API di Pinata che permette di caricare e salvare (operazione di **pinning**) qualsiasi tipo di file su un nodo IPFS di Pinata.

A questo si fa una richiesta asincrona con axios. Axios é un promise-based client HTTP per Node.js e il browser. Sul lato server usa il modulo Node.js http nativo, mentre sul lato client usa XMLHTTPRequest. Una volta caricato un file su IPFS, a quest'ultimo viene associato un content ID (CID) in modo da poter essere identificato univocamente all'interno della rete. Il content ID viene salvato in una variabile usando l'hook di React useState e utilizzato successivamente in fase di creazione dell'NFT.

3.3.3 Smart contract

Come visto nella sezione intoduttiva, uno smart contract é a tutti gli effetti un programma. Tale programma viene scritto in Solidity. In questa sezione si approfondiranno alcuni degli elementi chiave dello smart contract e come quest'ultimo viene messo in esecuzione sulla blockchain.
Nella stesura dello smart contract OpenZeppelin svolge un ruolo molto importante; infatti, vengono importati lo standard token *ERC721*, per la creazione di

nuovi NFT, e i contratti *Ownable* e *Counters*. In questo modo si possono utilizzare contratti giá scritti dagli sviluppatori in precedenza e programmare solo le funzioni di cui si necessita.

La struttura *TransferStruct*

Come giá visto nel paragrafo 1.1.2 in ogni blocco presente sulla blockchain sono salvate alcune informazioni. In quest'ottica, la struttura *TransferStruct* assume un ruolo fondamentale nella comunicazione con la blockchain, svolgendo il ruolo di tramite tra la blockchain stessa, che rappresenta il backend, ed il frontend. All'interno della struttura sono presenti tre campi:

- `addressFrom`: corrisponde all'indirizzo Metamask che invia l'NFT;
- `addressTo`: corrisponde all'indirizzo Metamask che riceve l'NFT;
- `Ipfs_Hash`: corrisponde al content ID dell'NFT su IPFS;

Quindi, viene tenuta traccia di ogni transazione effettuata sulla blockchain in un array. In questo modo la blockchain fa da vero e proprio database e l'unico modo per interagirvi é attraverso lo smart contract. In particolare, la funzione `getAllTransactions` permette di avere l'array delle transazioni effettuate sulla blockchain.

Le funzioni `payToMint` e `NFTtransaction`

Queste due funzioni permettono, rispettivamente, di creare un nuovo NFT (`payToMint`) e di trasferirlo (`NFTtransaction`).

- `payToMint`:

```
1         function payToMint(address payable receiver
            , string memory Ipfs_hash) public payable
            returns (uint256) {
2             require (msg.value <= 0.05 ether, "
                Fondi Insufficienti!");
```

```
3             uint256 newItemId = _tokenIdCounter.
                 current();
4             _tokenIdCounter.increment();
5             _mint(receiver, newItemId);
6             _setTokenURI(newItemId, Ipfs_hash);
7             string memory addressFromStr = toString
                  (msg.sender);
8             transactions.push(TransferStruct(
                 addressFromStr, addressFromStr,
                 Ipfs_hash));
9             return newItemId;
10        }
```

é un metodo *payable*, ció vuol dire che *Ether* puó essere inviato dall'utente allo smart contract; prende in ingresso due argomenti: `recipient` che rappresenta l'indirizzo dell'account Metamask che effettua la transazione, `Ipfs_Hash` che rappresenta il content ID generato da Pinata al momento del caricamento del file pdf. Il metodo usa `require` per controllare che l'account in questione abbia Ether a sufficienza per effettuare la transazione. Con la chiamata al metodo `_mint` (metodo definito all'interno delle librerie di Solidity importate da OpenZeppelin) l'utente paga per coniare il nuovo NFT.

- `NFTtransaction`:

```
1         function NFTtransaction(address payable
             receiver, string memory IpfsHash) public
             payable {
2             (bool sent, ) = receiver.call{value:
                 msg.value}("");
3             require(sent, "Failed to send Ether");
4             transactionCount += 1;
```

```
5               string memory addressFromStr = toString
                    (msg.sender);
6               string memory addressToStr = toString (
                   receiver);
7               transactions.push(TransferStruct(
                   addressFromStr,  addressToStr,
                   IpfsHash));
8               emit Transfer(msg.sender, receiver,
                   IpfsHash);
9       }
```

anche `NFTtransaction` é un metodo payable. Il metodo prende in ingresso `receiver` che rappresenta l'indirizzo dell'account Metamask a cui si vuole trasferire l'NFT e `Ipfs_Hash`; quindi viene effettuata la chiamata a `Transfer` per trasferire l'NFT.

In entrambe le funzioni, una volta superati i controlli, viene aggiunta una nuova riga all'array delle transazioni.
Gli ultimi passi da compiere affinché l'applicazione possa comunicare con la blockchain sono la compilazione e il *deployment* del contratto. La compilazione é necessaria per generare l'ABI, che altro non é che un file `JSON` al cui interno sono definiti gli attributi e le funzioni presenti nello smart contract. Il *deployment* sulla blockchain simulata da Hardhat, invece, permette di generare l'indirizzo del contratto, una stringa alfanumerica, che verrá utilizzato nell'interazione tra il frontend dell'applicazione e la blockchain, come si vedrá successivamente. L'indirizzo é utile quando viene inviato ether a seguito di una chiamata a una funzione dello smart contract; infatti, l'ether viene inviato all'indirizzo dello smart contract.

3.3.4 Creare un NFT

Una volta caricato il file da locale su IPFS, sará possibile creare un nuovo NFT cliccando su *MINT*, come visto nella descrizione dello Use Case diagram (figura

3.7).

Questo passaggio é cruciale perché necessita dell'interazione con lo smart contract che a sua volta interagisce con la blockchain.

```
1    const contractAddress = "0x5FbDB2315678afecb367f032
       d93F642f64180aa3";
2    const provider = new ethers.providers.Web3Provider(
       window.ethereum);
3    const signer = provider.getSigner();
4    const contract = new ethers.Contract(
       contractAddress, AppContract.abi, signer);
```

In particolare servono:

- contractAddress: é l'indirizzo del contratto ottenuto dopo il deploy, come descritto in precendenza;
- provider: fornito da una funzione di ethers.js, la libreria che interagisce con il portafoglio dell'utente MetaMask e la blockchain Ethereum;
- signer: rappresenta la chiave pubblica dell'utente MetaMask correntemente connesso;
- contract: viene istanziato grazie all'oggetto ethers.Contract che prende l'indirizzo del contratto, l'ABI e il signer.

```
1    const mintToken = async () => {
2        const connection = contract.connect(signer);
3        const addr = connection.address;
4        const result = await contract.payToMint(addr,
           Variabile_IpfsHash, {
5            value: ethers.utils.parseEther('0.05'),
6        });
7        await result.wait();
```

```
8           window.open("/OwnedFiles",  "_self");
9   }
```

Nella funzione viene stabilita innanzitutto la connessione tra il contratto e l'account Metamask in modo che possa essere inviato *Ether* dall'account connesso al contratto, poi viene effettuata una chiamata asincrona al metodo `payToMint` dello smart contract passando anche l'*Ether* da pagare per coniare il nuovo NFT. Una volta che viene confermata la transazione attraverso la notifica di Metamask, si passa alla Home Page (figura 3.5) dove sono presenti tutti gli NFT posseduti dall'utente connesso.

3.3.5 Trasferimento di un NFT

Come visto in figura 3.9, l'utente ha la possibilitá di trasferire gli NFT che possiede cliccando su *Transfer*. Come nel caso della creazione di un nuovo NFT, anche il trasferimento necessita dell'interazione con lo smart contract.

```
1     const transact = async (Ipfs_hash) => {
2         if (inputAccount.length !== 42 || inputAccount[
              0] !== "0" || inputAccount[1] !== "x") {
3             alert("Enter a valid account ID to make
                  transactions");
4             return;
5         }
6         const result = await contratto.NFTtransaction(
              inputAccount,  Ipfs_hash);
7         await result.wait();
8         onClose();
9         window.location.reload(false);
10  }
```

Nella funzione viene controllato che l'indirizzo passato sia valido (il controllo é basato sulla forma che hanno gli indirizzi Metamask), quindi, viene effettuata

la chiamata allo smart contract e in particolare alla funzione `NFTtransaction` passando il *content ID* relativo all'NFT da trasferire e l'account Metamask del destinatario dell'NFT. Una volta che il trasferimento va a buon fine, quindi dopo aver confermato la notifica di Metamask (come in figura 3.10), il modale viene chiuso e la pagina ricaricata, in modo da aggiornare gli NFT posseduti sul frontend.

3.3.6 Visualizzazione degli NFT sul frontend

L'hook `useEffect` di React permette di ottenere una serie di effetti collaterali nei componenti, tra cui recuperare dati, aggiornare il DOM e tanto altro.
Nell'applicazione viene utilizzato `useEffect` per aggiornare e tenere traccia di tutte le transazioni effettuate per mostrarle sul frontend ogni volta che si accede alla Home page, quando quest'ultima viene ricaricata oppure quando un NFT viene trasferito.

```
1     useEffect(() => {
2         const transactions = async () => {
3             const result = await contract.
                 getAllTransactions();
4             let temp = [];
5             let tempIpfs = "";
6             if (result.length > 0) {
7                 result.map(res => {
8                     let last = [];
9                     tempIpfs = res[2];
10                    last = res;
11                    result.map(res1 => {
12                        if (tempIpfs === res1[2]) {
13                            last = res1;
14                        }
15                    })
```

```
16                      if (!(temp.includes(last))) {
17                          temp.push(last);
18                      }
19                  })
20              }
21              temp.map(elem => {
22                  return setStrutturaTransazioni(
                        oldStrutturaTransazioni => [...
                        oldStrutturaTransazioni, elem]);
23              })
24          }
25          transactions();
26      }, []);
```

`useEffect` prende in ingresso due argomenti: una funzione (in questo caso la funzione passata é una funzione anonima che chiama la funzione asincrona `transactions`) e un parametro opzionale. All'interno della funzione `transactions` viene creato un array (`strutturaTransazioni`) per tenere traccia di tutti i passaggi di proprietari di ogni NFT. Ancora una volta é cruciale l'interazione con lo smart contract; infatti, attraverso la chiamata a `getAllTransactions` é possibile ottenere tutte le transazioni (salvate in un array) effettuate dagli utenti che fanno uso del contratto sulla blockchain. L'array ritornato dalla funzione del contratto viene, quindi, iterato e vengono salvati solo gli ultimi proprietari di ogni NFT in `strutturaTransazioni`. L'array ha, quindi, la stessa struttura definita in precedenza nello smart contract (indirizzo del mittente, indirizzo del destinatario e ipfs hash per l'NFT) e viene usato per visualizzare gli NFT sul frontend.

Capitolo 4

Testing

Una parte cruciale dello sviluppo di applicazioni software é il processo di testing. Con il testing si verifica che l'applicazione faccia tutto ció che é descritto nello Use case diagram. Affinché l'applicazione sviluppata sia quanto piú robusta possibile sono stati effettuati test in due fasi dello sviluppo:

- nella stesura dello smart contract;
- nell'implementazione delle pagine e componenti in React.

4.1 Testing dello smart contract

Affinché l'applicazione funzionasse nel modo giusto é stato effettuato il testing dello smart contract con l'utilizzo di Remix IDE. Remix IDE é un'applicazione open source utilizzata per lo sviluppo di smart contract. Remix é pensato per scrivere smart contract sia con il browser sia tramite applicazione desktop; é inoltre dotato di moduli che permettono il testing, debugging e distribuzione di smart contract sia su reti di test sia su blockchain come Ethereum.
Un altro strumento utile per il testing sia dello smart contract che dell'intera applicazione é Hardhat. Come giá visto in precedenza, Hardhat permette di simulare una blockchain in locale fornendo una serie di account falsi. Sfruttando, quindi, la blockchain creata da Hardhat, é stato possibile effettuare il testing e deploy dello smart contract con Remix in ambiente locale, connettendosi con il

provider Metamask sulla rete `localhost 8545` creata proprio da Hardhat; dato che il deploy del contratto avviene sulla blockchain, é necessario pagare ETH affinché l'operazione vada a buon fine. A questo punto, é possibile testare

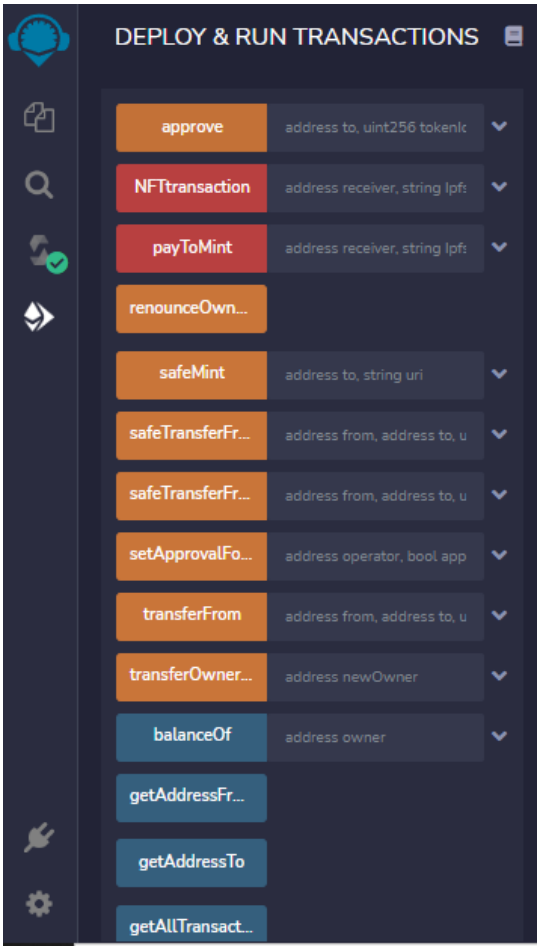

Figura 4.1: funzionamento di Remix IDE

tutte le funzioni scritte nello smart contract come si vede in figura; inoltre, dato che vengono importati gli standard ERC721 da OpenZeppelin, é possibile testare anche quelle funzioni. Nella figura le funzioni sono colorate in modo diverso:

- quelle blu sono funzioni `constant` o `pure` e non creano una nuova transazione quando vengono chiamate;
- quelle arancioni cambiano lo stato del contratto ma non accettano Ether e sono chiamate funzioni `non-payable`;

- quelle rosse sono funzioni `payable` e creano una nuova transazione quando vengono richiamate.

Per testare le funzioni con Remix basta inserire gli input; nel caso di funzioni payable, come `NFTtransaction` e `payToMint` che sono gli elementi chiave per la creazione e il trasferimento di un NFT, una notifica di Metamask e l'aggiunta di un nuovo blocco sul terminale dove é in esecuzione Hardhat stabiliranno il corretto funzionamento del metodo, generando di conseguenza una nuova transazione. Ne risulta che la strategia di testing usata é la *black box*, fondata sui requisiti senza conoscere internamente il codice.

4.2 Testing dei componenti dell' applicazione

In questo paragrafo verranno mostrati i test relativi all'applicazione; in particolare si mostreranno i test relativi ai componenti `Modal.js` e `CustomBTN.js`.
Per effettuare i test dei componenti React dell'applicazione é stata utilizzata la libreria *Jest*. *Jest* é un framework di unit test JavaScript sviluppato da Facebook; trova automaticamente i test da eseguire nel codice sorgente (file salvati con l'estensione `.test.js`, e funziona su progetti Javascript che includono React, Babel, TypeScript, Node, Angular, Vue. *Jest* é basato su Jasmine, rispetto al quale fornisce alcune funzionalitá aggiuntive, come la creazione automatizzata di mock e un ambiente jsdom. Altre caratteristiche peculiari di *Jest* sono l'esecuzione di test in parallelo per aumentarne la velocitá, l'esecuzione tramite un'implementazione di una DOM falsa (grazie a jsdom), e la possibilitá di testare il codice asincrono in modo sincrono.

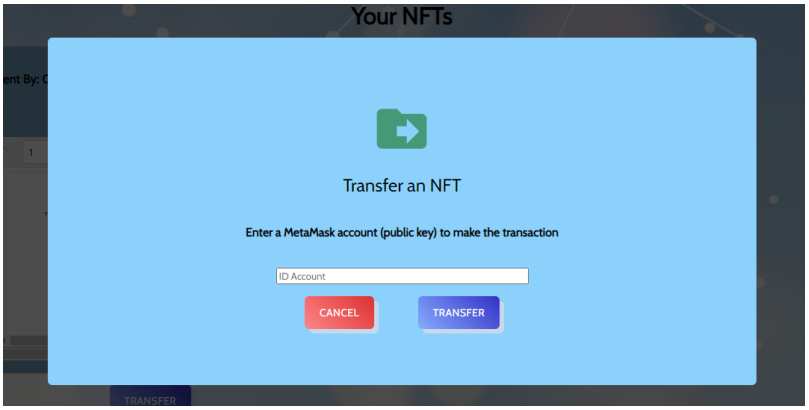

Figura 4.2: Schermata del Modal

Come giá descritto in precedenza nel paragrafo 3.1 dove viene descritto lo Use Case diagram, il modale si apre sulla Home page e permette il trasferimento di un NFT ad un altro account inserito in input.

Per il modale é stata creata una suite di test per controllare il corretto funzionamento dei vari componenti all'interno del modale stesso. In particolare le componenti testate sono: il modale stesso, verificando che venga correttamente visualizzato a schermo, l'icona di trasferimento, i testi, il campo di input e i pulsanti. Per ogni componente é stata creata una variabile apposita per permettere il test.

```
1  describe("Modal Component", () => {
2      it("render modal", () => {
3          const { getByTestId } = render(<Modal />);
4          const modale = getByTestId("modale");
5          expect(modale).toBeTruthy();
6      })
7
8      it("render campo input", () => {
9          const { getByTestId } = render(<Modal />);
10         const input = getByTestId("input");
11         expect(input).toBeTruthy();
12     })
```

```
13
14      it("render testo", () => {
15          const { getByTestId } = render(<Modal />);
16          const testo = getByTestId("testo");
17          expect(testo).toBeTruthy();
18      })
19
20      it("render icona", () => {
21          const { getByTestId } = render(<Modal />);
22          const icona = getByTestId("icon");
23          expect(icona).toBeTruthy();
24      })
25
26      it("render pulsanti", () => {
27          const { getByTestId } = render(<Modal />);
28          const pulsanti = getByTestId("buttons");
29          expect(pulsanti).toBeTruthy();
30      })
31  })
```

Quindi ogni caso di test effettuato sopra verifica che le componenti vengano visualizzate correttamente sulla pagina web.

Un altro elemento testato che assume particolare rilevanza all'interno dell' applicazione é `CustomBTN.js`; quest'ultimo, infatti, viene riutilizzato piú volte all'interno di quest'ultima.

Per l'elemento `CustomBTN.js` sono stati effettuati i seguenti casi di test:

```
1       it("render button", () => {
2           const { getByTestId } = render(<CustomBTN />);
3           const button = getByTestId("customButton");
4           expect(button).toBeTruthy();
5       })
```

In questo test viene verificato che il pulsante venga visualizzato correttamente a schermo;

```
1     it("onClick su CustomBTN", () => {
2         const handleFunction = jest.fn();
3         const { getByTestId } = render(<CustomBTN
              onClick={handleFunction}/>);
4         const button = getByTestId("customButton");
5         fireEvent.click(button);
6         expect(handleFunction).toHaveBeenCalledTimes(1)
              ;
7     })
```

In questo test si verifica la funzione `onClick`, passata come argomento a `CustomBTN`. Viene simulato il click sul pulsante con l'oggetto `fireEvent`. Jest permette di creare funzioni mock, quindi di simulare la chiamata ad una funzione con `jest.fn()`; a questo punto si verifica che la funzione handleFunction sia chiamata una sola volta.

Sviluppi futuri e considerazioni finali

Lo scopo del lavoro di tesi é stato la realizzazione di un'applicazione web decentralizzata che permettesse agli utenti di creare e scambiare Non-Fungible-Token partendo da documenti pdf caricati da locale. Sebbene l'applicazione si pone come obiettivo di essere facile ed immediata da utilizzare e con un'interfaccia grafica semplice ed intuitiva, le vere difficoltá iniziano quando l'utente deve, per forza di cose, interagire con strumenti e plugin aggiuntivi, come Metamask. Infatti, nonostante le applicazioni del Web 3.0 e le blockchain siano in continua affermazione, resta comunque un limite oggettivo il fatto di dover utilizzare diversi strumenti per interagirvi; questa limitazione é evidente soprattutto perché la quasi totalitá degli utenti che utilizzano il web sono abituati all'uso del Web 2.0, dominato da poche aziende che ne detengono il controllo centrale. Le problematiche si presentano anche dal lato dei programmatori e non solo da quello degli utenti finali; infatti, l'utilizzo di queste nuove tecnologie costringe gli sviluppatori non solo a rendere le applicazioni decentralizzate quanto piú usabili possibile per gli utenti finali, ma anche reinventare e ritoccare i pattern architetturali giá esistenti, come accaduto durante lo sviluppo dell'applicazione dovendo interagire con lo smart contract e non direttamente con la blockchain.

Nello sviluppo dell'applicazione sono stati raggiunti tutti gli obiettivi prefissati inizialmente. Un possibile sviluppo futuro potrebbe essere quello di accettare diversi tipi di file da validare e scambiare sulla blockchain, dato che qualsiasi tipo di file digitale puó potenzialmente diventare un NFT; inoltre, in questo modo, si puó ampliare ulteriormente il bacino d'utenza.

Un ulteriore passo in avanti é quello di rilasciare l'applicazione sulla blockchain Ethereum; va ricordato che le applicazioni non possono essere hostate sulla blockchain, vista la natura di quest'ultima, ma, anche in questo caso, tutti gli artefatti realizzati relativi all'applicazione andrebbero salvati su IPFS; in questo modo qualsiasi applicazione creata per essere usata sul Web 2.0 puó essere usata anche sulla blockchain. Ció che, peró, puó essere rilasciato sulla blockchain é lo smart contract.

In conclusione, la blockchain e gli NFT saranno due delle colonne portanti dell'informatica nei prossimi anni; se si supereranno alcuni dei problemi piú importanti rendendo le applicazioni quanto piú usabili possibile, il Web 3.0 soppianterá definitivamente il Web 2.0 e, considerando i passi da gigante che sono stati fatti negli ultimi anni, quel giorno non é poi cosí lontano.

Siti Web consultati

[1] IBM

http://www.ibm.com/

[2] Il Fatto Quotidiano

https://www.ilfattoquotidiano.it/

[3] Wikipedia

https://www.wikipedia.org/

[4] Agenda Digitale

https://www.agendadigitale.eu

[5] The Cryptonomist

https://cryptonomist.ch/

[6] Il Sole 24 ore

https://www.ilsole24ore.com/

[7] Ethereum

https://ethereum.org/

[8] Blockchain 4Innovation

https://www.blockchain4innovation.it/

[9] Studio legale Sperti

https://www.studiolegalesperti.it

[10] Tech4Future

https://tech4future.info/

[11] React

https://it.reactjs.org/

[12] Metamask

https://metamask.io/

[13] Solidity Programming Language

https://soliditylang.org/

[14] HardHat

https://hardhat.org

[15] Pinata Cloud

https://www.pinata.cloud/

[16] Geeks for Geeks

https://www.geeksforgeeks.org/

[17] OpenZeppelin

https://www.openzeppelin.com/

[18] Node.JS

https://www.nodejs.org/

[19] Jest Javascript Testing

https://jestjs.io/

[20] Remix IDE

https://remix.ethereum.org/