Università Politecnica delle Marche
Facoltà di Ingegneria

Corso di Laurea in ingegneria informatica e dell'automazione

Sviluppo di interfacce utente per la gestione della certificazione di dati basata su blockchain

Development of user interfaces for managing blockchain-based data certification

Relatore:
Prof. Marco Baldi

Correlatore:
Dott. Giacomo Zonneveld

Laureando:
Antonio Di Placido

Anno Accademico 2023/2024

Contents

Capitolo 1

Introduzione

Al giorno d'oggi, le aziende, per mantenere la loro competitività sul mercato, devono adattarsi rapidamente a modelli di produzione e distribuzione sempre più complessi. Quindi, oltre alla gestione delle attività interne, occorre migliorare il rapporto con altre aziende per l'acquisizione dei materiali e dei servizi. Da qui nasce l'idea della supply chain, intesa come l'insieme di sistemi autonomi e indipendenti che collaborano tra loro per la creazione di un prodotto o servizio dedicato all'utente finale. Quindi la supply chain comprende tutti gli aspetti di un'azienda come la logistica, la produzione, la qualità delle materie prime, i servizi al cliente e tutto ciò che fa parte del ciclo di vita del prodotto. La gestione di questi processi risulta complessa e onerosa per le aziende, motivo per cui si ricorre sempre più alla creazione di infrastrutture informatiche. La gestione di questi sistemi informatici è molto complicata perché deve tener conto di tutti gli standard, linee guida e regole che compongono la supply chain. Quindi, per far sì che ogni parte possa controllare ogni singolo passaggio del prodotto, ci deve essere la collaborazione tra tutte le parti e l'interconnessione tra i sistemi informatici. Altro aspetto che deve essere preso in considerazione è la possibilità di azioni disoneste che possono riguardare la qualità del prodotto, pagamenti tra aziende , ritardi e interruzioni della produzione. Con le tecnologie tradizionali è quasi impossibile gestire un sistema così complesso, ma la possibile soluzione può essere rappresentata dal sistema Blockchain.
Il presente lavoro di tesi propone lo sviluppo di un'applicazione web per la gestione di una supply chain e certificazione dati basata su tecnologia blockchain. Il codice sviluppato, tramite libreria reactjs, permette la visualizzazione di documenti con ricerca filtrata, l'autenticazione degli utenti e un'area riservata per la gestione dei documenti in un determinato processo produttivo di un prodotto di filiera.
Il lavoro è strutturato come segue: La parte fondamentale del progetto di tesi consiste nello sviluppo dell'applicazione, dove viene spiegato come sono state strutturate le funzioni all'interno dell'interfaccia utente e come vengono gestiti i documenti. Inizialmente viene introdotta la tecnologia blockchain, la struttura principale, protocolli utilizzati e casi di studio in settori differenti che cercano di adottare la tecnologia Blockchain nelle loro supply chain. Nella seconda parte del capitolo viene analizzata la struttura di una web application e la gestione di una single page application. Successivamente viene introdotta la libreria Reactjs, con approfondimento dei componenti utilizzati nel codice. Infine si mostrano le conclusioni e i possibili sviluppi futuri del progetto.

Capitolo 2

Blockchain e Web application

La blockchain è una tecnologia nata principalmente per l'ambito finanziario e che negli ultimi anni viene utilizzata da diversi settori e aziende, ad esempio per migliorare la tracciabilità e la trasparenza delle filiere produttive, certificazioni digitali di proprietà e autenticità delle proprie opere, protezione dei dati di pazienti e facilitarne l'accesso sicuro e privato tra diverse entità mediche. L'obiettivo di questa tecnologia è quello di creare una rete decentralizzata che possa essere funzionale, anche se priva di un ente centrale. Negli ultimi anni molte aziende cercano di monitorare sempre di più il ciclo di vita dei loro prodotti adottando il meccanismo della supply chain, in modo che i propri clienti possano risalire a tutti i processi di lavorazione del prodotto. Questo può essere fatto attraverso dei software o web application messi a disposizione dall'azienda per fornitori, produttori e clienti.

2.1 Come funziona la Blockchain

La Blockchain è un tipo di registro distribuito composto da blocchi concatenati tra di loro e contenenti dei record (transazioni).Ogni blocco è costituito da una serie di transazioni e da un header il quale, tra le varie informazioni, include:

- **Block version:** indica alcune regole per convalidare il blocco
- **Merkle root hash:** l'hash viene calcolato tramite l'applicazione della struttura dati crittografica Merkle Tree alle transazioni. In questo modo è possibile rappresentare in maniera sintetica le transazioni, garantendone al tempo stesso l'integrità.
- **Previous block hash:** è un riferimento al blocco precedente.
- **Timestamp:** ora in cui ogni blocco è stato creato.
- **Nonce:** un numero univoco utilizzato durante il mining di un blocco.

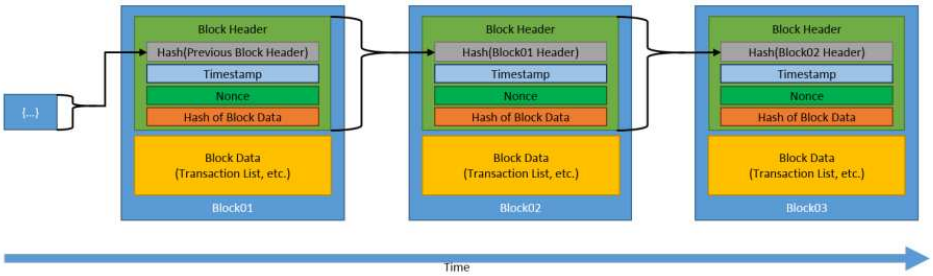

Figure 2.1: Catena dei blocchi [1]

In ogni blocco è contenuta una versione crittografata delle transazioni che viene gestita tramite Merkle tree.Il Merkle tree è una struttura ad albero. Ogni foglia è il risultato dell'applicazione di una funzione hash ad una transazione. Ad ogni "incrocio" viene calcolato l'hash della combinazione dei due nodi precedenti. Si procede fino ad avere un unico nodo chiamato Root Hash, cioè radice dell'albero[2].

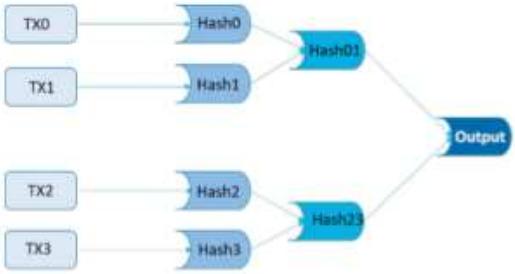

Figure 2.2: Merkle Tree

2.1.1 Decentralizzazione del sistema

La blockchain rientra in quelli che sono i sistemi **Distributed Ledger Technology (DLT)**.Questi sistemi danno la possibilità di decentralizzare la gestione dei dati in modo da non avere la necessità di un ente garante per la convalida di una transazione. Nella rete DLT ogni nodo ha una copia aggiornata del registro delle transazioni, in modo da garantire la sincronizzazione dei dati tra i partecipanti. Le caratteristiche fondamentali della DLT, che sono presenti nella blockchain, sono:

- **Trasparenza e immutabilità:** ogni nodo ha lo stesso accesso alle informazioni registrate, per garantire che le decisioni vengano prese in modo collaborativo.Permette che ogni transazione sia tracciabile e assicura che i dati non vengano modificati.
- **Resistenza agli attacchi:**gestendo e distribuendo dati su nodi differenti, rende il sistema molto sicuro da attacchi malevoli. Questo perché per modificare un dato bisognerebbe ottenere l'autorizzazione da un numero significativo di nodi,operazione che richiederebbe un dispendio di risorse enorme. Quindi un tentativo di manomissione risulterebbe impraticabile e costoso.

2.1.2 Consenso e sicurezza

Nella fase di inserimento di un nuovo blocco, i nodi della rete devono essere d'accordo sull'inserimento del nuovo blocco nella catena. Il consenso tra i nodi viene raggiunto tramite l'uso di particolari protocolli tra cui il Proof-of-work (PoW) e il Proof-of-Stake[3]. In un algoritmo di consenso proof-of-work (PoW), ogni blocco di transazioni è collegato al blocco precedente utilizzando un valore hash crittografico. Il processo di mining è eseguito dai "miner". Un miner deve risolvere una sfida crittografica come prova di lavoro. Il primo miner a risolvere il problema crittografico, acquisisce il diritto a pubblicare il proprio blocco. In un algoritmo di consenso Proof-of-Stake (PoS), i nodi di una rete possono diventare candidati per convalidare nuovi blocchi in base allo stake di cripto-valuta messo a garanzia. Più alto sarà lo stake, maggiore sarà la probabilità che il nodo venga scelto per convalidare il prossimo blocco. Questo algoritmo mira a incentivare il comportamento onesto. Un nodo, per essere candidato alla validazione di nuovi blocchi, viene selezionato da un algoritmo che effettua la scelta in base ai seguenti criteri:

- **Stake:** Rappresenta la quantità di cripto-valuta detenuta. I nodi con stake maggiore hanno probabilità più alte di essere selezionati in quanto maggiormente incentivati a comportarsi in maniera corretta. Infatti, eventuali comportamenti malevoli sarebbero puniti con la sottrazione di parte dello stake o addirittura con la privazione del diritto a partecipare come miner
- **Presenza nella rete:** tiene traccia di quanto tempo un nodo candidato è stato un validatore. Più a lungo un nodo è stato un validatore, maggiori sono le sue probabilità di essere selezionato come nuovo validatore.
- **combinazione hash/stake:** combinazione del valore hash più basso di un blocco e della posta in gioco più alta, se entrambi i requisiti sono soddisfatti il nodo ha più possibilità di essere scelto come validatore.

2.1.3 Smart contract

Gli smart contract sono semplicemente dei protocolli informatici che incapsulano e replicano i termini dei contratti del mondo reale nel dominio digitale. I contratti sono fondamentalmente un accordo legalmente vincolante tra due o più parti, con ciascuna parte impegnata a rispettare i propri impegni definiti nel codice del contratto [4]. Grazie alla decentralizzazione del sistema, gli smart contract non hanno bisogno di una terza parte che faccia da garante. L'esecuzione del contratto infatti è garantita dai nodi i quali effettuano la convalida della transazione contenente il contratto stesso. I due attori stipulano il contratto e decidono i termini, contenuti e clausole, quest'ultime vengo inserite in un blocco tramite chiavi private e successivamente eseguono un processo di cifratura. Una volta aggiunto alla blockchain, il contratto diventa immutabile e auto-eseguibile a patto che le condizioni definite all'interno dello stesso siano soddisfatte. Gli smart contract possono essere deterministici o non deterministici. La differenza è che quelli deterministici non hanno bisogno di interventi da enti esterni alla blockchain, mentre in quelli non deterministici utilizzano come terza parte un sistema chiamato oracolo, il quale si occupa di recuperare informazioni dal mondo esterno. I dati provenienti da varie fonti (servizi meteo, notizie, sistemi bancari, ecc.) vengono quindi inviati alla blockchain come dati

transazionali. L'esecuzione dello smart contract dipende da queste informazioni fondamentali, in cui l'invocazione avviene quando vengono soddisfatte condizioni predeterminate (eventi). Le condizioni possono includere qualsiasi tipo di dato, come pagamenti riusciti, letture della temperatura o fluttuazioni dei prezzi. Gli oracoli offrono lo scambio di dati tra diverse applicazioni software tramite API.[5]

2.1.4 Integrazione della blockchain nella supply chain

Sempre più aziende stanno cercando di aumentare e migliorare la supervisione e il coordinamento di ogni singola fase del processo produttivo garantendo autenticità e trasparenza. L'uso della tecnologia blockchain può essere utile al raggiungimento di questi obiettivi. Alcune soluzioni in fase di adozione riguardano l'uso di smart contract e NFT(Not Fungible Tokens). Questi ultimi sono token crittografici che rappresentano la proprietà o la prova di autenticità di un determinato asset. La loro creazione e gestione avviene tramite smart contract che ne stabiliscono le regole e le proprietà[6]. L'NFT, visualizzato in modo prominente accanto al prodotto, funge da distintivo di autenticità e provenienza. Fornisce ai potenziali acquirenti una panoramica digitale completa del prodotto, mostrandone l'origine, i dettagli di produzione e altri metadati rilevanti. Questa integrazione dell'NFT con la quotazione del prodotto non solo aumenta la fiducia dell'acquirente nella legittimità del prodotto, ma sottolinea anche l'identità unica del prodotto nel vasto mercato[7]. Questo sistema è progettato per salvaguardare gli interessi di tutte le parti coinvolte nel transito del prodotto. Le parti coinvolte, come trasportatori e intermediari, sono tenute a depositare un importo predeterminato come garanzia in uno smart contract. Questa garanzia funge da garanzia finanziaria, assicurando che ciascuna parte adempirà diligentemente alle proprie responsabilità durante il percorso del prodotto. Lo smart contract conserva questa garanzia, contrassegnando la proprietà dell'NFT come "Trasferimento in sospeso". Questo stato rimane fino a quando il prodotto non raggiunge in sicurezza l'acquirente e non viene sottoposto a verifica di successo[7].

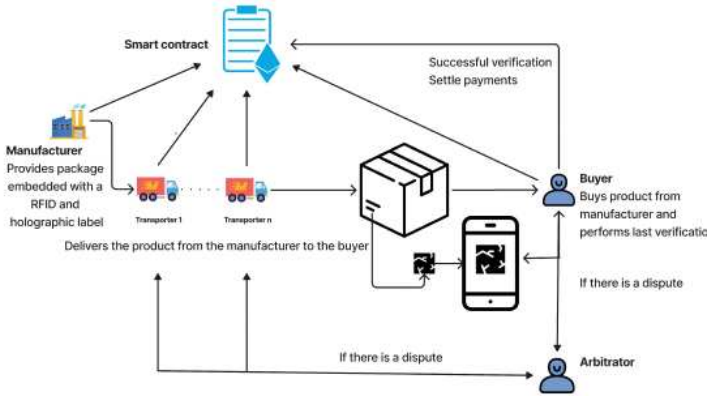

Figure 2.3: Sistema di circolazione dei prodotti basato su blockchain

Anche nelle supply-chain sanitarie si iniziano a condurre studi sull'adozione della blockchain, per affrontare le crescenti sfide legate alla sicurezza dei dati, l'interoperabilità e la privacy dei pazienti e garantire che le informazioni sui pazienti siano mantenute riservate ma accessibili solo alle parti autorizzate. Inoltre, la blockchain facilita la creazione di una struttura affidabile e interoperabile per la gestione dei dati sanitari, contribuendo a migliorare la trasparenza e l'efficienza nei processi, pur mantenendo elevati standard di protezione della privacy.[8].

Molti studi, come [9],[10],[11], hanno analizzato il potenziale della blockchain nella logistica containerizzata globale via mare. Lo studio ha sottolineato i vantaggi della blockchain per i partner commerciali e le compagnie di navigazione, sottolineando la necessità di una sua adozione diffusa nell'industria marittima.Quindi l'adozione delle blockchain permetterebbe di migliorare l'efficienza, la tracciabilità, la trasparenza e l'economicità delle supply chain marittime. Permetterebbe di gestire in magnera più agile le polizze di carico, documenti essenziali per documentare e tracciare il trasporto delle merci. La grande difficoltà di manomissione della blockchain ridurrebbe le frodi burocratiche, nonostante gli inconvenienti tra cui l'elevato consumo di energia e i problemi normativi.

2.2 Sviluppo di Applicazioni Web Moderne

Le web application si differenziano dalle applicazioni desktop perchè non hanno bisogno di essere installate su uno specifico dispositivo, ma possono essere usate da qualsiasi dispositivo che disponga di un web browser. Lo sviluppo di applicazioni web ha subito un'evoluzione significativa con l'introduzione di tecnologie, strumenti e metodologie che migliorano la produttività degli sviluppatori e la qualità del software.

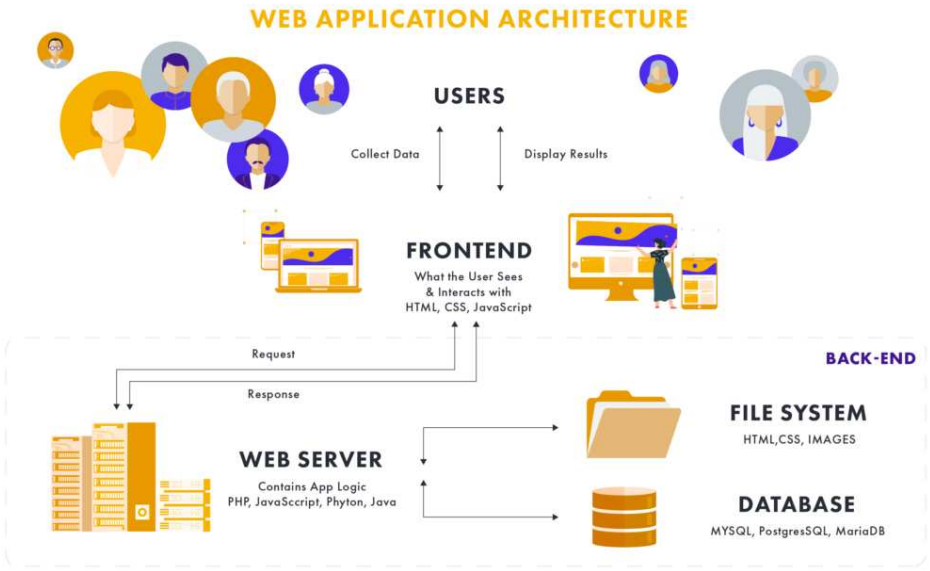

Figure 2.4: Architettura di un'applicazione web

Le moderne applicazioni web sono diventate una parte fondamentale nell'ecosistema quotidiano, dovuto alla crescita sempre maggiore delle esigenze dei clienti. Per questo lo sviluppo del lato front-end di un' application web ha un ruolo importante, deve fare in modo che l'interfaccia utente sia pratica e intuitiva[12]. Il front-end è tutto ciò che viene visualizzato sullo schermo, come il layout, i pulsanti e i form. Viene realizzato utilizzando HTML (per la struttura), CSS (per lo stile) e JavaScript (per l'interattività). Ciò che aiuta gli sviluppatori a creare interfacce front-end altamente dinamiche e interattive sono i Framework e librerie JavaScript, come React, Vue.js e Angular, che semplificando il processo di sviluppo e migliorano l'esperienza utente.

2.2.1 Single Page Application

La Single Page Application (SPA) è una tipologia di applicazione web che prevede il caricamento sul web browser di una sola pagina. Questa viene usata come shell per tutte le altre pagine Web dell'applicazione. Nelle SPA non è previsto alcun postback completo al server, nessun aggiornamento completo di una singola pagina Web e nessun oggetto incorporato.Viene utilizzato un elemento HTML come contenitore, il cui contenuto viene aggiornato e sostituito da una visualizzazione all'altra tramite un meccanismo di routing e template lato front-end[13].

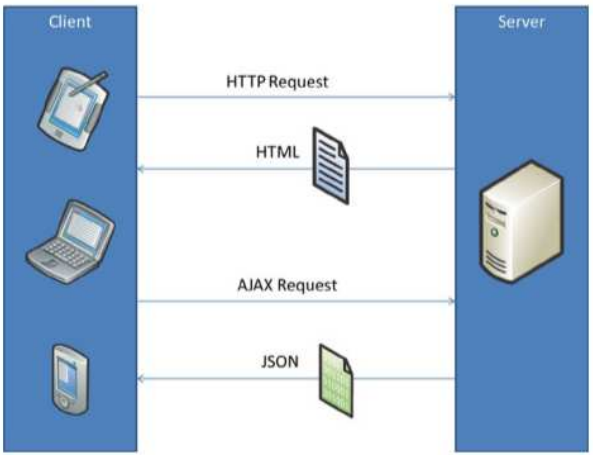

Figure 2.5: Architettura SPA

Nella SPA, per ricevere e inviare dati si utilizza il meccanismo di richieste asincrone, che può essere effettuato tramite AJAX o Fetch API. Le operazioni asincrone non bloccano l'interfaccia utente, che rimane reattiva per altre interazioni dell'utente e, quando l'operazione termina, un callback può aggiornare la parte rilevante della pagina web. Una callback è una funzione che viene passata a un'altra funzione come argomento, per poi essere richiamata al termine di un'operazione, molto difficile da gestire se sono presenti più callback nella stessa funzione. Un meccanismo più semplice da utilizzare è rappresentato dalle promesse, che definiscono un valore non ancora disponibile ma che potrebbe esserlo in futuro, presenta una sintassi più intuitiva e una gestione degli errori più efficace. Le promesse vengono utilizzate nell'interfaccia fetch API, che lo rende molto più semplice nell'effettuare richieste al server.

Capitolo 3

React.js

React.js è una libreria JavaScript lanciata da facebook nel 2013, creata per sviluppare interfacce grafiche di applicazioni web più in modo da renderle più dinamiche e interattive. Negli anni successivi trova un massiccio consenso nella comunity degli sviluppatori, per questo il punto forte che sta migliorando questa libreria è la grande quantità di documentazione presente online, sempre in continua crescita.

Negli ultimi anni React.js si sta consolidando come una delle librerie di riferimento per lo sviluppo frontend. Ecco un'immagine che mostra la crescita e l'utilizzo di React rispetto ad altre librerie JavaScript.

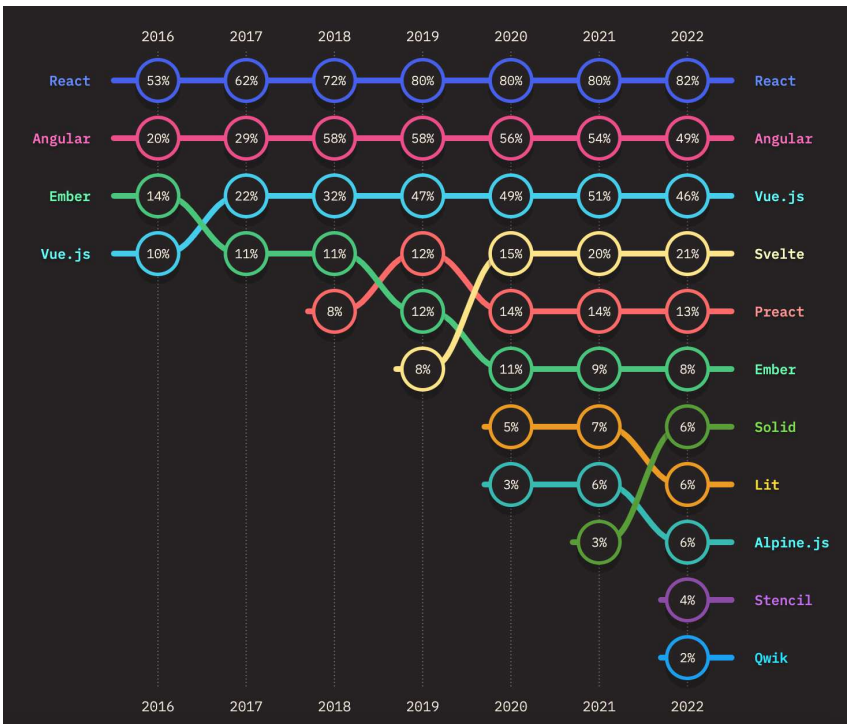

Figure 3.1: Statistiche globali sull'uso di React.js rispetto ad altre librerie JavaScript (2022)

3.1 Componenti

Un componente in React è un'unità modulare e riutilizzabile che rappresenta una parte dell'interfaccia utente. È possibile combinare più componenti per creare strutture complesse, mantenendo modularità e isolamento. Questa separazione facilita lo sviluppo, poiché le modifiche a un componente non impattano direttamente gli altri, e permette di estendere facilmente le funzionalità[14]. Oggi i componenti si scrivono principalmente come funzioni piuttosto che classi. Queste funzioni descrivono sia la grafica che il comportamento del componente e possono ricevere dati immutabili (props) o gestire uno stato interno. Le funzioni usate per i componenti sono spesso "pure", ovvero restituiscono sempre lo stesso output per gli stessi input. I componenti vengono rappresentati come un albero DOM virtuale, poi trasformato e visualizzato nel browser. Per renderizzare l'intera applicazione, si utilizza il metodo ReactDOM.render() nel file index.js. React separa le responsabilità organizzando logica e UI in unità accoppiate debolmente, anziché dividere artificialmente codice e markup in file distinti. È comune avere un file per componente, in cui si importa ed esporta ciò che è necessario. Si può usare export default per esportare un componente principale, o l'export nominale per più elementi. Ogni componente può avere associati file per test o stili CSS specifici. Questa struttura consente di mantenere un progetto ordinato e facilmente scalabile.

3.2 JSX: JavaScript XML

JSX è un'estensione sintattica di JavaScript che permette di scrivere codice simile a HTML all'interno di file JavaScript. Questo rende il processo di sviluppo delle interfacce utente più intuitivo. Formalmente JSX è definibile come "un'estensione della sintassi del linguaggio JavaScript che fornisce un modo per strutturare il rendering dei componenti usando una sintassi familiare a molti sviluppatori.". I browser possono interpretare solo codice JavaScript puro, quindi il codice scritto in JSX deve essere convertito in JavaScript standard attraverso un processo chiamato traspilazione. Questo processo traduce il JSX in chiamate a funzioni JavaScript comprensibili dai browser, garantendo che il codice possa essere eseguito correttamente. L'uso di JSX non è obbligatorio ma offre numerosi vantaggi, in particolare un supporto visivo che rende più intuitiva e chiara la costruzione del codice per l'interfaccia utente. Altro aspetto molto importante è l'utilizzo di React DOM, che effettua automaticamente l'escape di qualsiasi valore inserito in JSX prima di renderizzarlo. In questo modo, garantisce che non sia possibile iniettare nulla che non sia esplicitamente scritto nell'applicazione. Ogni cosa è convertita in stringa prima di essere renderizzata. Questo aiuta a prevenire gli attacchi XSS (cross-site-scripting). La funzione render() di React può restituire un solo nodo. Pertanto, se ci sono più nodi, devono essere racchiusi in un nodo padre, come React.Fragment, che funge da contenitore senza generare tag nell'output finale. JSX, pur somigliando all'HTML, è formalmente JavaScript e ha alcune differenze:

- **I** tag devono essere sempre chiusi (anche in forma autochiudente)
- **A**lcune parole riservate, come class e for, diventano className e htmlFor
- **L**e props (attributi JSX) possono accettare non solo stringhe ma anche espressioni JavaScript

Gli elementi HTML si distinguono dai componenti React perché iniziano con una lettera minuscola. Per definire stili CSS in un componente JSX, si usa un oggetto JavaScript con chiavi in notazione camelCase, associato tramite la proprietà style, permettendo l'incapsulamento degli stili.

3.3 Props

Per la gestione dei dati React utilizza un modello di flusso unidirezionale che adotta un approccio top-down in cui stato e informazioni fluiscono esclusivamente dal componente padre verso i componenti figli. In base a questo modello, le props vengono utilizzate per passare dati e funzioni tra i componenti in una gerarchia. Le props sono immutabili, cioè il componente che le riceve non può modificarle. Ogni props deve avere un nome unico e può contenere diversi tipi di valori, tra cui:

- Stringhe, analoghe agli attributi HTML e racchiuse tra virgolette
- Espressioni JavaScript, come array, numeri, booleani o oggetti, racchiuse tra parentesi graffe. Se si passa un oggetto inline, si utilizzano doppie parentesi graffe
- Funzioni, che consentono al componente figlio di richiamare logiche definite nel componente genitore.

L'uso delle funzioni come props è particolarmente utile per centralizzare la logica applicativa nel componente padre, lasciando al componente figlio il compito di gestire esclusivamente la presentazione. In questo modo, il figlio può invocare le funzioni ricevute senza doversi occupare direttamente della logica sottostante. Per accedere al contenuto delle props in un componente figlio, si utilizza il parametro props, che rappresenta, attraverso un oggetto JavaScript, le informazioni passate dal componente padre. Le chiavi di questo oggetto corrispondono ai nomi degli attributi definiti nel JSX del componente padre. Un metodo alternativo e più sintetico per accedere ai valori delle props può essere fatto attraverso l'assegnamento di destrutturazione. Questo consiste nell'estrarre le chiavi di props direttamente come variabili, specificandole tra parentesi graffe. I nomi delle variabili devono coincidere con le chiavi dell'oggetto props, e ciascuna variabile conterrà il valore associato. La destrutturazione può avvenire all'interno del corpo della funzione oppure direttamente nella lista dei parametri del componente, rendendo il codice più conciso e leggibile.

3.4 Componenti Hooks

Gli Hooks permettono di utilizzare più funzioni di React senza dover ricorrere alle classi. Concettualmente, i componenti React sono sempre stati più vicini alle funzioni. Gli Hooks abbracciano le funzioni, senza però sacrificare lo spirito pratico di React. Gli Hooks offrono accesso a vie di uscita imperative e non richiedono l'uso di programmazione funzionale o reattiva. Gli hooks in React sono funzioni che, per convenzione, iniziano con il prefisso use. In ogni componente funzionale, possono essere chiamati diversi hooks, ma sempre nello stesso ordine. Questo implica che non possono essere utilizzati all'interno di blocchi condizionali o cicli, garantendo così il corretto funzionamento dello stato e degli effetti. Gli hooks sono associati alla specifica istanza di un componente, assicurando che

gli stati e le logiche siano isolati tra loro. Tra gli hooks presenti all'interno della libreria React, i più utilizzati sono:

- `useState`: Utilizzato per gestire lo stato interno di un componente e aggiornarlo in risposta a eventi interni o esterni. Accetta come argomento un valore iniziale dello stato e restituisce un array con due elementi: la variabile che rappresenta lo stato corrente e una funzione per aggiornarlo. Questo hook può essere richiamato più volte all'interno dello stesso componente per definire diverse variabili di stato, permettendo una gestione flessibile e modulare.

- `useEffect`: implementazione del design pattern Observer che viene utilizzato per gestire gli effetti collaterali nelle componenti. Accetta due argomenti: una funzione da eseguire e un array di dipendenze che specifica quali variabili di stato devono essere osservate. La funzione viene eseguita dopo ogni render del componente e ogni volta che una delle variabili incluse nell'array delle dipendenze subisce un cambiamento. Per eseguire la funzione solo dopo il primo render del componente, è sufficiente passare un array vuoto come secondo argomento.

- `useContext`: Consente di condividere lo stato e le funzioni per aggiornarlo tra i componenti senza passare esplicitamente attraverso le props con un flusso top-down, eliminando la necessità di coinvolgere componenti intermedi nella trasmissione dei dati, cioè senza passare per tutta la gerarchia delle componenti.

Per la gestione di particolari comportamenti o esigenze è possibile definire hooks personalizzati, per permettere che tra le varie componenti venga condivisa lo stato e logica applicativa.

3.5 Collegamento al server

Il server gestisce e conserva i dati necessari per il corretto funzionamento dell'app. Quando il client necessita di nuovi dati da visualizzare, invia richieste specifiche al server. Queste richieste sono indirizzate agli URL delle API (Application Programming Interface) pubblicate, che mettono a disposizione le funzionalità richieste dall'applicazione. La comunicazione avviene attraverso i principali metodi del protocollo HTTP, come GET (per leggere dati), POST (per crearne di nuovi), PUT (per aggiornarli completamente), PATCH (per aggiornamenti parziali) e DELETE (per rimuoverli). Per riuscire a gestire le richieste in maniera asincrona, è possibile utilizzare la libreria Axios Hooks, che fornisce degli hook dedicati per facilitare l'interazione con il server. L'hook principale è *useAxios(config, options)*, che accetta due parametri:

- `Config:` può contenere una stringa che rappresenta un URL oppure un oggetto di configurazione di tipo Axios.Config che consente di specificare vari dettagli come un URL o metodo HTTP da utilizzare (ad esempio, GET o POST) e parametri aggiuntivi da includere nella richiesta.

- `Option:` opzionale, i campi indicano se si vuole utilizzare una cache e se la richiesta deve essere eseguita manualmente, ovvero quando il programmatore decide di eseguirla

L'hook restituisce due valori principali:

1. Un oggetto contenente i seguenti campi:
    - **data**: contiene il corpo della risposta del server in caso di successo; è `undefined` in caso contrario.
    - **loading**: un booleano che indica se la richiesta è ancora in corso.
    - **error**: contiene informazioni sull'errore, se la richiesta fallisce.
    - **response**: l'oggetto completo della risposta inviata dal server.
2. Una funzione che consente di avviare manualmente la richiesta, utile quando l'esecuzione non è automatica.

3.5.1 Routing

In un'applicazione React, la gestione delle viste e delle pagine è affidato al meccanismo di routing grazie alla libreria **React Router**, che consente la navigazione dinamica tra componenti diversi senza ricaricare l'intera pagina. React Router utilizza un concetto chiamato *client-side routing*, che permette di manipolare l'URL nel browser e aggiornare la vista corrispondente senza inviare una richiesta al server per ogni cambiamento di pagina. React Router si basa su una serie di componenti e funzioni che permettono di gestire il routing in modo flessibile ed efficiente. Tra i componenti principali troviamo:

- **BrowserRouter**: È il contenitore principale per il routing che utilizza l'API della history del browser per mantenere la UI sincronizzata con l'URL. È il "wrapper" che avvolge l'intera applicazione e permette a React Router di funzionare.
- **Route**: Questo componente definisce le relazioni tra un URL e un componente. Esso associa una determinata rotta (URL) a un componente che verrà renderizzato quando l'URL corrisponde.
- **Link**: Utilizzato per creare collegamenti ipertestuali che permettono la navigazione tra le pagine senza ricaricare l'intera applicazione. Sostituisce il tradizionale tag `<a>` in React.
- **Switch**: Questo componente garantisce che venga renderizzata solo la prima route figlia che corrisponde all'URL corrente. Utile per evitare che più rotte non corrispondano allo stesso percorso.
- **Redirect**: applicato in determinate funzioni per passare ad un'altra pagina.

ll routing può essere di diversi tipi:

- **Routing Dinamico:** vengono passati dei parametri all'interno dell'URL e utilizzarli per determinare quale componente o risorsa caricare.I parametri vengono inseriti nelle rotte utilizzando i due punti (`:`) seguiti dal nome del parametro variabile.
- **Routing Annidato:** consente di avere rotte secondarie all'interno di un'altra rotta.Utile per applicazioni che hanno sezioni interne con più sotto-sezioni.
- **Routing Condizionale:** permette che l'accesso a determinate pagine è limitato o reso disponibile in base a condizioni specifiche, come lo stato dell'autenticazione dell'utente.

3.6 Rendering di liste (.map)

Per la gestione delle liste di dati, come un elenco di prodotti, utenti o messaggi, React mette a disposizione funzioni come il `.map()`, utilizzato per iterare su un array di dati e trasformarli in una serie di elementi React. Questo approccio rendere l'interfaccia utente molto flessibile, adattandosi facilmente ai dati dinamici.Permette di gestire grandi quantità di dati, con React che ottimizza il rendering grazie al "virtual DOM" e fornisce un codice per la gestione delle liste che rimane semplice e modulare. Il ruolo della Proprietà `key` è molto importante nel rendering di liste in React. Le chiavi univoche permettono a React di identificare in modo efficiente quali elementi sono stati aggiunti, modificati o rimossi, migliorando le performance del rendering. È importante che ogni elemento reso dinamico attraverso `.map()` abbia una chiave univoca e stabile, come un identificatore numerico o un ID.Con l'utilizzo delle key e del rendering basato su componenti, React può aggiornare solo le parti dell'interfaccia che cambiano, riducendo il lavoro necessario al browser per modificare l'interfaccia.

Capitolo 4

Sviluppo applicazione web

4.1 Server di certificazione

L'applicazione sviluppata si pone l'obiettivo di fornire un servizio di certificazione dati basata su tecnologia blockchain. Tale servizio sarà utilizzato in contesto supply chain per cui ciascun partecipante alla filiera potrà caricare dati relativi al proprio processo produttivo inerente ad un prodotto di filiera. Tramite un codice di filiera (supply chain ID), sarà quindi possibile ripercorrere a ritroso tutta la filiera per visualizzarne tutte le informazioni dichiarate.

4.1.1 Gestione dati, certificazione e verifica validità

Il documento al su interno presenta tutti i dati raccolti in una determinata fase di filiera, in particolare:

- descrizione informazioni raccolte
- data di inserimento documento
- codice della supply chain
- codice del processo
- numero dei campioni effettuati

Quando il documento ottiene la certificazione, al suo interno comparirà la data in cui è avvenuta la certificazione.

Le informazioni caricate saranno quindi certificate in blockchain a intervalli prestabiliti tramite la seguente procedura di certificazione:

1. Utente inserisce il dato sotto forma di documento
2. Utente richiede la certificazione del documento
3. Documento viene:
    - sintetizzato tramite l'applicazione di una funzione hash
    - messo in una coda di attesa
4. Al raggiungimento del trigger della certificazione (intervallo temporale scaduto)

- I documenti in coda di attesa vengono combinati in una struttura dati crittografica chiamata Merkle Tree(fig. 2.2). Tale struttura ad albero avrà per foglie i singoli documenti sintetizzati (digest hash) a partire dai quali si otterrà un unico dato chiamato Merkle Root (radice dell'albero).
- Al fine di garantire la ripetibilità della verifica di integrità di un documento, vengono salvati su database in maniera persistente:
  - il merkle tree come documento
  - per ciascun documento, viene salvato il riferimento all'albero e alla sua posizione tra le foglie
- Viene costruita la transazione passando come dato da scrivere in chain il merkle root
- Esecuzione della transazione
- Salvataggio su database della ricevuta (hash della transazione, nome blockchain, timestamp, ecc)

Dato un codice di filiera, la piattaforma mostra per ciascun documento associato le informazioni dichiarate e la loro validità. Tale validità viene verificata tramite una procedura di confronto tra i dati salvati su database e il dato salvato su blockchain.
Il processo di verifica è di seguito sintetizzato:

1. Calcolo del digest hash del documento
2. Estrazione della prova crittografica dal merkle tree, chiamata Merkle Proof
3. Calcolo del merkle root tramite la merkle proof
4. Lettura merkle root salvato su blockchain (lettura transazione)
5. Confronto merkle root calcolato con merkle root letto. Se i due dati corrispondono, il documento è da intendersi valido

4.1.2 Autenticazione

Il sever per gestire e identificare l'utente che sta interagendo con l'applicazione utilizza i session cookies, che sono dei file di testo generati dal browser per il controllo della sessione di navigazione.Nel momento dell'autenticazione il server genera il SessionD, un identificatore unico per la sessione.Il SessionID viene memorizzato temporaneamente nel browser che lo includerà nelle richieste successive da inviare al server.
Quando il browser viene chiuso, il session cookie viene cancellato e non è più possibile accedere a una determinata sessione.Attraverso il server si può gestire un timeout session, un limite di tempo oltre il quale la sessione scade e l'utente deve autenticarsi di nuovo.Dopo la fase di autenticazione il server fornisce il ruolo dell'utente in modo da poter gestire l'accesso alle risorse, e le azioni che possono essere effettuate.

4.1.3 API

Per l'interazione del frontend con il server abbiamo utilizzato la libreria Axios, creando delle chiamate API sviluppate per le funzioni dell'applicazione.Di seguito viene mostrata

la tabella delle chiamate divise per utente guest e azienda.

1.1)

Endpoint	/guest/getSupplyChain
Descrizione:	Dato un codice di filiera (ID) e filtri opzionali (date,nome azienda, documento certificato) vengono forniti tutti i documenti relativi al codice e ai filtri.
Parametro	**Valore**
Metodo	GET
Parametri	query: {supplyChainID: ID, filters: {start: S, end: E, orgs: ['org1', ...], isCertified: bool} }

1.2)

Endpoint	/guest/downloadCert
Descrizione:	Ottenere per un determinato documento la prova di certificazione (transaction hash, merkle proof, timestamp)
Parametro	**Valore**
Metodo	GET
Parametri	query: {docID: ID, companyName:CNAME}

1.3)

Endpoint	/guest/login
Descrizione:	Autenticazione utente attraverso l'inserimento di email e password
Parametro	**Valore**
Metodo	POST
Parametri	body: {email: EMAIL, password:PASSWORD}

2.1)

Endpoint	/staff/doc
Descrizione:	Inserimento di un nuovo documento
Parametro	**Valore**
Metodo	POST
Parametri	body: {certData: {field1:value1, field2:value2}}

2.2)

Endpoint	/staff/getAllCompanyDocs
Descrizione:	Recupero di tutti i documenti dell'utente attualmente loggato
Parametro	**Valore**
Metodo	GET
Parametri	query: {page: N1, elemsPerPage: N2, filters: {start:S, end:E} }

2.3)

Endpoint	/staff/doc
Descrizione:	recupero di un documento specifico
Parametro	**Valore**
Metodo	GET
Parametri	body: {docID: ID}

2.4)

Endpoint	/staff/doc
Descrizione:	Modifica un documento esistente. Nella chiamata vengono inseriti solo i campi da aggiornare e devono seguire il percorso del nome del campo.Se la certificazione è stata richiesta o è stata emessa il documento non può essere modificato.
Parametro	**Valore**
Metodo	PATCH
Parametri	body: {docID: ID, editedData:{field1:editedV, ..., complexFieldX: {fieldX1:editedValue}}, toBeDeleted:[pathToField,...]}

2.5)

Endpoint	/staff/doc
Descrizione:	Eliminare un documento esistente. L'operazione non è consentita se la certificazione è stata richiesta o è stata emessa.
Parametro	**Valore**
Metodo	DELETE
Parametri	body: {docID:ID}

2.6)

Endpoint	/staff/requestCert
Descrizione:	Richiedi la certificazione per un determinato documento.
Parametro	**Valore**
Metodo	POST
Parametri	body: {docID:ID}

2.7)

Endpoint	/staff/abortCert
Descrizione:	Interrompere la richiesta di certificazione per un determinato documento. Applicabile solo se è in corso una richiesta di certificazione per il documento.
Parametro	**Valore**
Metodo	DELETE
Parametri	body: {docID:ID}

2.8)

Endpoint	/staff/downloadCert
Descrizione:	Ottenere la prova di integrità come un insieme di informazioni relative all'albero di merkle (merkle proof) e alla blockchain.
Parametro	**Valore**
Metodo	GET
Parametri	body: {docID:ID}

4.2 Progettazione

Si è preferito strutturare l'applicazione su single page application per far si che l'utente riesca a navigare nell'applicazione in maniera agevole. L'applicazione dovrà prevedere le seguenti funzionalità :
Per utente non registrato (Guest)

- Ricerca dei documenti tramite chiamata API 1.1,che restituisce come response tutti i documenti collegati ai filtri inseriti
- Autenticazione effettuata tramite chiamata API 1.3, e fornisce come response il role dell'utente registrato.Se il documento è certificato può essere richiesta la prova di certificazione tramite chiamata API 1.2.

Per utente registrato (caso utente di un azienda)

- Sezione con form per inserimento di un nuovo documento, l'inserimento è gestito dall'API 2.1.
- Sezione documenti azienda, attraverso la chiamata API 2.2 vengono forniti tutti i documenti legati all'utente autenticato e accedere alle informazioni di ogni singolo documento(chiamta API 2.3).
- Gestione dei documenti attraverso le API 2.4(modifica), 2.5(elimina), 2.6(richiesta certificazione), 2.7(richiesta interruzione certificazione) e 2.8(richiesta prova di un documento certificato).

In questa fase si è cercato di organizzare e strutturare i contenuti in modo efficace per:

- ridurre la complessità: si è cercato di rendere le informazioni chiare e fruibili;
- supportare la scalabilità: la struttura organizzativa è stata progettata in modo che risultasse facilmente adattabile all'aumento di nuove funzionalità e carico dei dati sempre più elevato;
- creare familiarità con la piattaforma e rendere piacevole la navigazione sia per utenti con elevata esperienza sia per nuovi utilizzatori.

4.2.1 Organizzazione delle componenti jsx

Il front-end è diviso in 2 package principali: **common** e **component**.

Common contiene tutti i file di componenti considerate riusabili, quindi non appartenenti a una pagina specifica. Tra questi vi sono componenti UI, come bottoni e cards e modal di conferma e nella subdirectory *elements* tutte le componenti di form, come checkbox, select e input.

Component contiene tutti i file delle pagine visualizzabili dell'applicazione, tramite il sistema di **routing** che possiamo trovare nel file *App.jsx*

4.3 Implementazione

4.3.1 Gestione dei file JSON

JSON è una sintassi di testo che facilita lo scambio di dati strutturati tra tutti i linguaggi di programmazione, è una sintassi di parentesi graffe, parentesi quadre, due punti e virgole che è utile in molti contesti, profili e applicazioni. JSON sta per JavaScript Object Notation, nasce ispirandosi ai letterali di oggetto di JavaScript, ovvero ECMAScript come definito nella ECMAScript Language Specification, Third Edition[15]. Tuttavia, non tenta di imporre le rappresentazioni di dati interne di ECMAScript ad altri linguaggi di programmazione ma condivide un piccolo sottoinsieme della sintassi di ECMAScript. La sintassi JSON non è una specifica di uno scambio di dati completo. Uno scambio di dati significativo richiede un accordo tra un produttore e un consumatore sulla semantica associata a un particolare utilizzo della sintassi JSON. Ciò che JSON fornisce è il framework sintattico a cui tale semantica può essere associata[16].
sono stati strutturati principalmente due file JSON, il primo per la visualizzazione del documento, il secondo per la creazione di un documento. Il file JSON, di un documento che viene visualizzato nell'applicazione, presenta tre oggetti principali: certData, detail e blockchainData. Detail contiene informazioni relative al tracciamento di chi ha caricato e richiesto la certificazione dei dati. BlockchainData contiene informazioni relative alla blockchain e vengono creati e visualizzati nel documento dopo la certificazione. La struttura più importante si trova nell'oggetto certData che contiene tutti i dati che verranno certificati, parte da oggetti semplici che contengono delle coppie chiave-valore come descrizione, data, dati numerici del documento,e oggetti più complessi che contengono array di oggetti annidati, creato per contenere i campioni registrati da un'azienda.

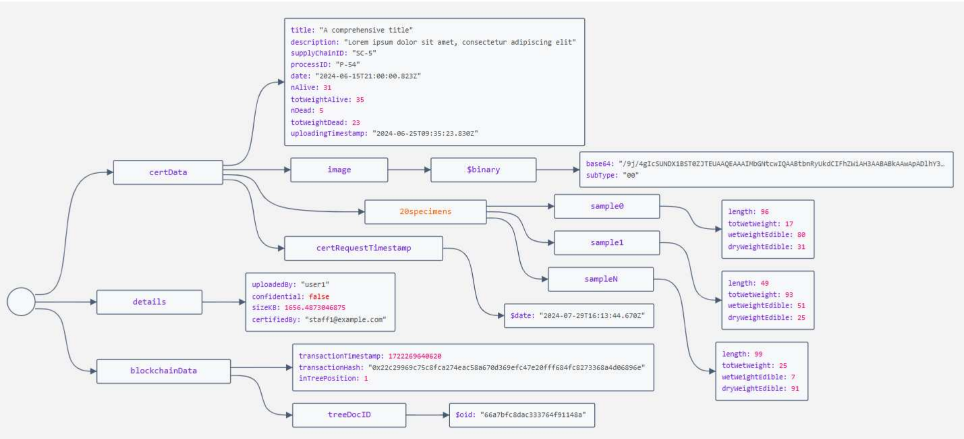

Figure 4.1: Struttura del file JSON

Il secondo file JSON è strutturato per il form di creazione del nuovo documento, contiene un'array con all'interno oggetti che presentano le proprietà che un campo deve avere, cominciando dal tipo ("text", "number", "datetime-local", "file"), etichetta del campo, se si tratta di un campo con inserimento obbligatorio. Il file presenta anche un oggetto complesso con al suo interno un'array di oggetti che riportano le proprietà dei campi sopra citati e il numero di elementi che devono essere generati per contenere i campi nell'array

```
{
```

```
"field_id": "20specimens",
"field_type": "complex",
"n_items": 20,
"partial_id": "sample",
"complexList": [
  {
    "field_id": "length",
    "field_type": "number"
  },
  {
    "field_id": "dryWeightEdible",
    "field_type": "number"
  }
]}
```

4.3.2 Modulo di ricerca dei documenti tramite filtri

Il modulo di ricerca documenti è stato sviluppato in modo da consentire l'inserimento di un codice di filiera seguito dalle date di inserimento dei documenti, quali compagnie hanno inserito il documento e se il documento è validato.

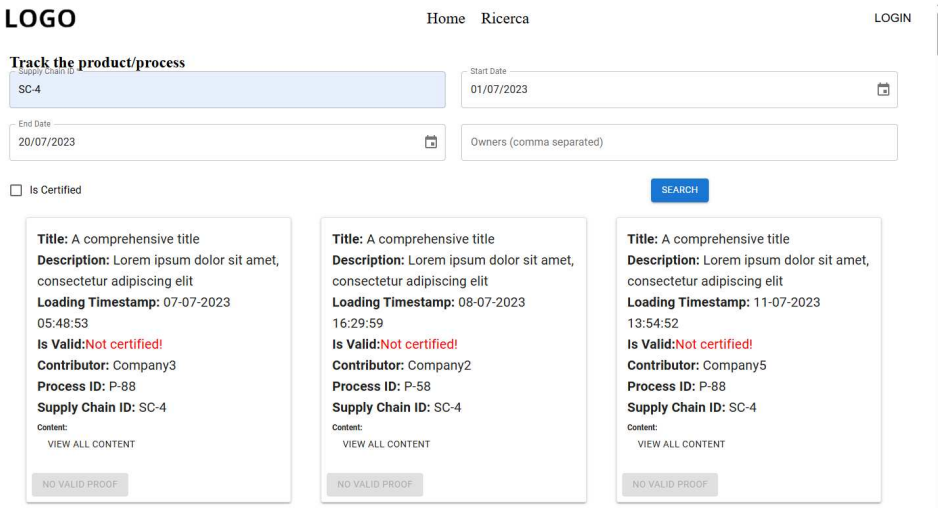

Figure 4.2: Ricerca filtrata

La ricerca viene fatta tramite chiamata API, inserendo il codice di filiera e i campi opzionali come parametri.La chiamata restituirà come response dei file JSON contenenti gli oggetti che devono essere renderizzati e visualizzati. Questo viene fatto tramite il .map() che crea un nuovo array popolato con gli oggetti restituiti dalla chiamata e li gestisce in base al tipo di dato per la vista.

```
{backendData.map((elem, index) => (
            <Grid item key={index} xs={12} md={6} lg={4}>
               <CardSupplyChain
                    id={elem._id}
                    supplyChainID={elem.supplyChainID}
                    content={elem.certData}/>
             </Grid>))}
```

La gestione di certData risulta più complessa a causa della sua natura variabile e non prevedibile. Infatti certData può essere costituito non solo da tipi di dato semplice (stringhe, numeri, booleani), ma anche complessi (array, oggetti annidati). Per questo è stata realizzata una funzione ricorsiva che per ogni oggetto venga presa la coppia chiave-valore associata.In modo da riuscire a gestire:

- Immagini: Chiama una funzione showImage per renderizzare le immagini.
- Array: Inserisce ogni elemento dell'array in una griglia flessibile. Se un elemento è un oggetto, lo renderizza ricorsivamente.
- Valori semplici: Ritorna il valore e lo inietta nella vista.

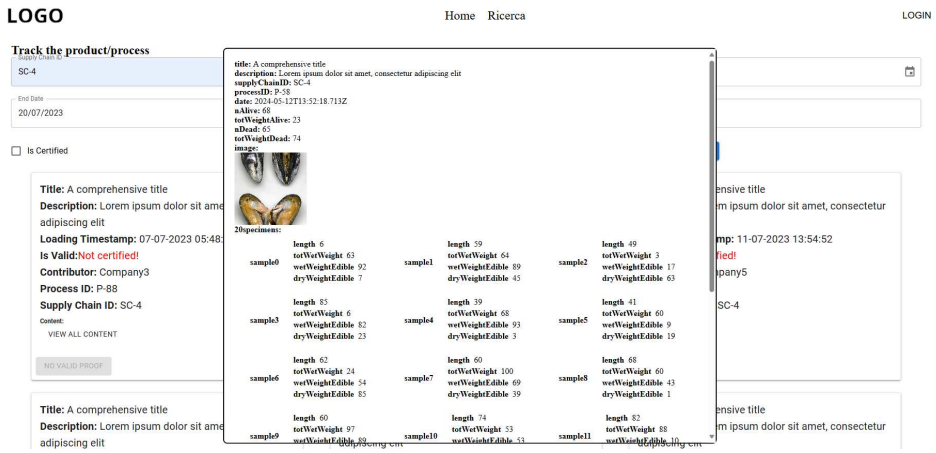

Figure 4.3: Campioni contenuti nel documento

4.3.3 Modulo di autenticazione

Permette agli utenti di autenticarsi e accedere alla propria area riservata per la gestione dei loro documenti.

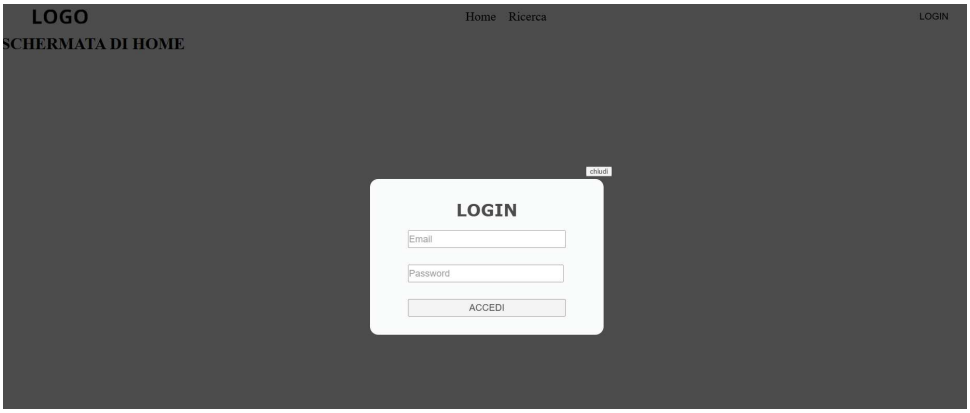

Figure 4.4: Form di login

L'autenticazione viene gestita tramite chiamata API, passandogli come parametri *email* e *password*, al success della chiamata il response fornirà al browser il session-cookie

Figure 4.5: Navbar utente autenticato

(link alla sezione autenticazione) e il ruolo (role) del utente loggato che in questo caso per l'azienda abbiamo chiamato staff.

Per la visualizzazione di questi button ci facciamo aiutare dal ruolo dell'utente:

```
const [role, setRole] = useState('');

const handleLoginSuccess = (userRole) => {
  setRole(userRole);
  setShowModal(false);
};

{role === 'staff' && (
  <NavLink to="/documenti">Miei Documenti</NavLink>
)}
{role === 'staff' && (
  <NavLink to="/newdocument">Nuovo Documento</NavLink>
)}
```

4.3.4 Inserimento nuovo documento

Tramite form permette di inserire dati e salvarli come nuovo documento.

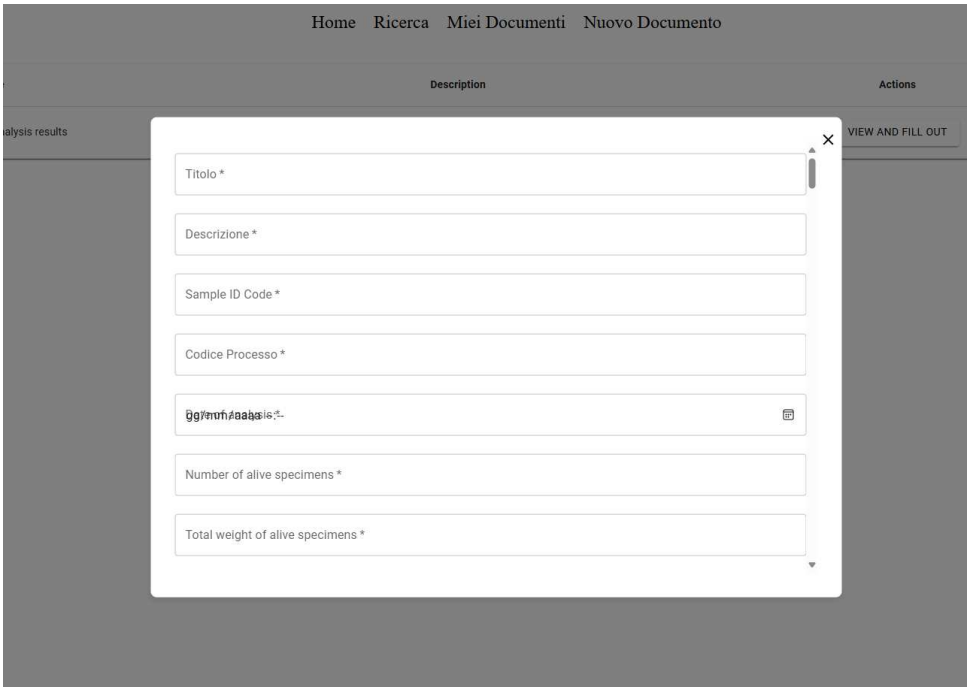

Figure 4.6: Form per il nuovo documento

Abbiamo iniziato con lo strutturare un documento che definisce la struttura di un modulo per l'analisi dei campioni di cozze. È un JSON con dettagli specifici sui campi

del modulo chiamati fields, un array di oggetti che descrivono ciascun campo del modulo, con attributi come field.id, field.label, field.type, field.value, ecc. I campi possono essere di diversi tipi: text, number, datetime-local, file e complex. Il tipo complex rappresenta un gruppo di campi, come nel caso dei campioni. Attraverso il .map andiamo a gestire tutti gli oggeti che si trovano nell'array field in modo che uno per uno posso essere passati alla funzione renderFormField che gestisce i campi in base al tipo, quindi:

- per i campi di tipo text, number, e datetime-local userà il TextField di Material-UI .
- per i campi di tipo file Userà un input file .
- per i campi complex, genera un insieme di sotto-campi basati sulla struttura definita.

```
{Array.from({ length: field.n_items }, (_, index) => (
       <div key={`${field.field_id}_sample_${index + 1}`}>
         <h4>Sample {index + 1}</h4>
          {field.complexList.map((subField) => (
           <TextField
           key={`${field.field_id}_sample_${index + 1}_${subField
              .field_id}`}
              fullWidth
              label={`${subField.field_label}`}
              type={subField.field_type}
              variant="outlined"
              margin="normal"
              onChange={(e) => handleChange(`${field.field_id}
                 _sample_${index + 1}_${subField.field_id}`, e.
                 target.value, 'complex')}
              required={subField.field_mandatory === "yes"}/>
```

- Array.from crea un array con field.nitems elementi, specificati nel documento JSON, ciascuno rappresentante un campione.
- field.complexList.map((subField) da qui vengono rappresentati i sotto-campi del campione (ad esempio, length, totWetWeight, ecc.).
- onChange gestisce il cambiamento del valore del campo e chiama handleChange con l'ID completo del campo (composto da field.field.id, l'indice del campione e subField.field.id), il nuovo valore e il tipo complex.

In handleChange è presente la funzione setFormData:

```
if (type === 'complex') {
            const [mainId, sample, sampleIndex, subId] = fieldId.
               split('_');
            setFormData(prevState => ({
                ...prevState,
                [mainId]: {
                    ...prevState[mainId],
```

```
                    [`${sample}${sampleIndex}`]: {
                        ...prevState[mainId]?.[`${sample}${
                           sampleIndex}`],
                        [subId]: value,
                    }
                }
            }));
```

Quindi se il campo è complex , fieldId sarà scomposto in quattro parti:

- mainId: l'ID principale del campo complesso.
- sample: la stringa fissa "sample".
- sampleIndex: l'indice del campione.
- subId: l'ID del sotto-campo.

La funzione setFormData aggiorna lo stato formData, che verrà passato nel body della chiamata API. prevState rappresenta lo stato precedente. Lo stato viene aggiornato in modo annidato per mantenere la struttura dei dati complessa.Quindi nel handleSubmit la chiamata sarà fatta in questo modo:

```
 const handleSubmit = async () => {
        // Validation
const requiredFields = row.fields.filter(field => field.
   field_mandatory === "yes");
const emptyFields = requiredFields.filter(field => !formData[
   field.field_id] || formData[field.field_id] === '');
if (emptyFields.length > 0) {
      alert("Non sono stati compilati tutti i campi!");
      return;
   }
   const dataToSubmit = { certData: formData };
        try {
            const response = await apiClient.axiosPost('/staff/
               doc', dataToSubmit);
            if (response) {
```

La prima parte controlla che tutti i campi siano compilati se il contrario genera un alert che avvisa della mancata compilazione di un campo, se tutti i campi sono compilati partirà la chiamata POST che nel body conterrà il campi compilati.

4.3.5 Visualizzazione documenti

Permette all'utente attualmente loggato di accede all'area in cui sono contenuti i propri documenti .

Figure 4.7: Documenti utente autenticato

Dopo l'autenticazione, cliccando sul pulsante "Miei Documenti", viene effettuata una chiamata API che, in base all'utente in sessione, restituisce tutti i documenti associati al prodotto di filiera per la fase gestita dall'azienda. La visualizzazione dei dati del documento utilizza lo stesso meccanismo impiegato nella ricerca documenti, con la funzione .map per visualizzare l'oggetto certData. Se i documenti sono numerosi, la tabella dei documenti viene gestita con la paginazione per evitare uno scroll continuo durante la ricerca dei documenti.

4.3.6 Modifica documento

Viene visualizzato un form con i campi compilati dai dati attualmente presenti sul documento, i dati nei campi possono essere modificati e salvati nel documento.

Cliccando sul button di modifica si aprirà un modal in cui saranno presenti tutti i dati contenuti nel certData passati al renderFormFields

```
const renderFormFields = (obj, parentKey = '') => {
 return Object.entries(obj).map(([key, value]) => {
    const inputName = parentKey ? `${parentKey}.${key}` : key;
```

La funzione renderFormFields prende due argomenti: obj, l'oggetto che rappresenta i dati, e parentKey, una stringa opzionale che tiene traccia delle chiavi genitore negli oggetti annidati. Object.entries(obj) converte l'oggetto in un array di coppie [chiave, valore].inputName viene creato concatenando parentKey e key con un punto se parentKey è presente, altrimenti inputName è semplicemente key.Per evitare che nei campi vengano inseriti caratteri non utili andiamo a controllare il tipo di dato e gli diamo il proprio campo

```
{typeof value === 'number' ? (
 <Form.Control type="number" name={inputName} value={value}
    onChange={handleChange} />
  ) : (
```

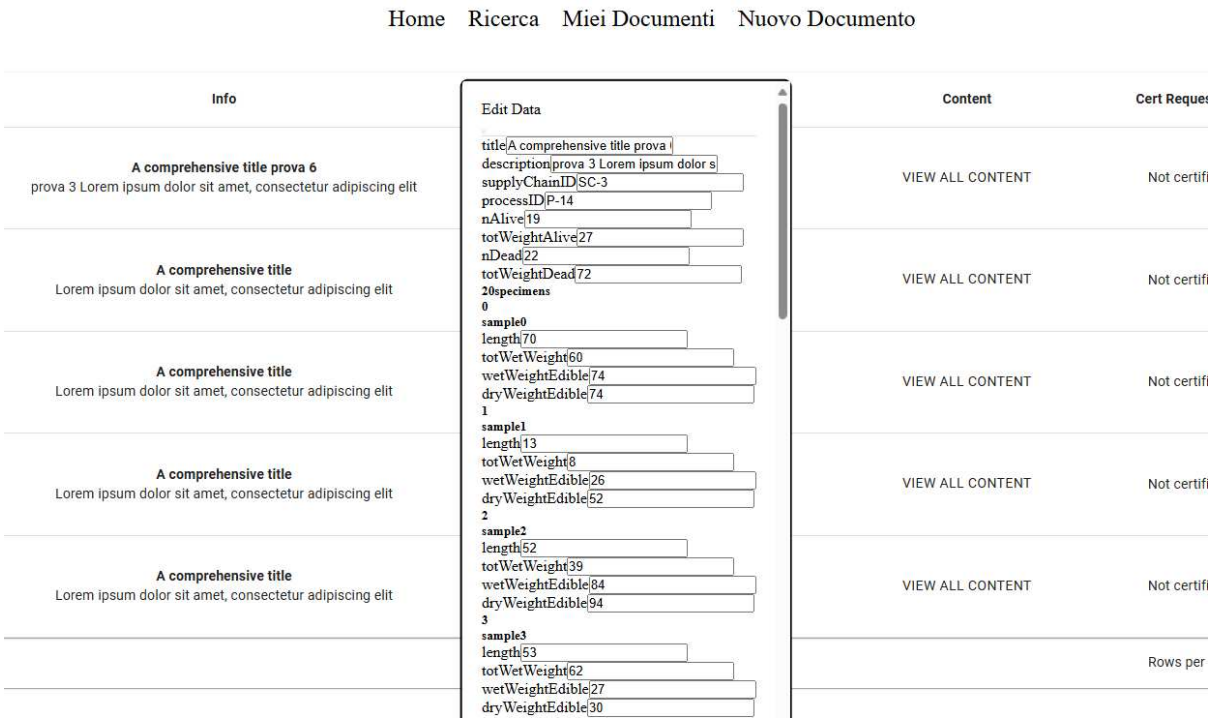

Figure 4.8: Forn di modifica del documento

```
<Form.Control type="text" name={inputName} value={value} onChange
   ={handleChange} />
   )}
```

Quando cambiamo un campo viene attivata la funzione handleChange.La sua responsabilità principale è aggiornare lo stato del modulo (formData) e tenere traccia delle modifiche effettuate dall'utente (changes).

```
const { name, value } = e.target;
```

- e.target:contiene il riferimento al campo di input che ha generato l'evento.
- name:contiene il nome del campo di input.
- value:contiene il nuovo valore del campo di input

```
setFormData((prevState) => ({
            ...prevState,
            [name]: value
        }));
```

Utilizziamo setFormData per aggiornare lo stato formData. Creiamo un nuovo oggetto di stato copiando l'oggetto precedente (prevState) usando lo spread operator (...). Aggiorniamo la proprietà dell'oggetto formData corrispondente al nome del campo (name) con il nuovo valore (value).

```
setChanges((prevChanges) => ({
            ...prevChanges,
            [name]: value
        }));
```

Utilizziamo setChanges per aggiornare lo stato changes.Al click su salva viene chiamata la funzione handleSubmit:

```
const handleSubmit = async () => {
        if (!changes || Object.keys(changes).length === 0) {
            alert("Non sono stati modificati campi!");
            return;}
        const payload = {
            docID: rowID,
            editedData: changes,
            toBeDeleted: toBeDeleted
        };
     apiClient.axiosPatch(‘/staff/doc‘, payload)
            .then(response => {
                alert("Dati modificati correttamente");
                handleClose();
                reloadTable();
            })
```

Questa funzione viene chiamata quando l'utente invia il modulo. Prepara un payload con i dati modificati e quelli da eliminare e invia una richiesta PATCH all'API.

4.3.7 Certificazione documento

Tramite chiamata API permette di fare la richiesta di certificazione dei dati e salva data e orario della richiesta.

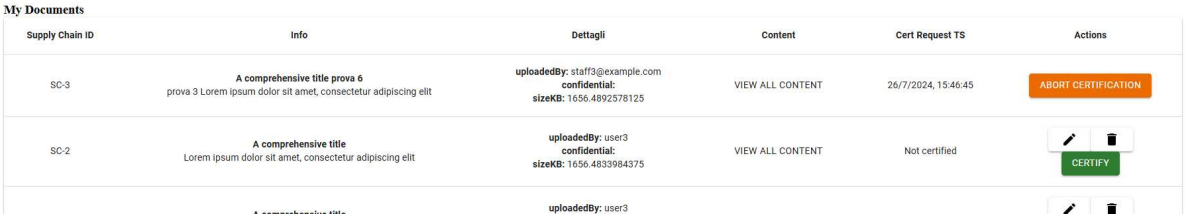

Figure 4.9: Richiesta certificazione

Questa funzione ,tramite chiamata API, permette di inviare una richiesta di certificazione del documento.Dopo aver richiesto la certificazione verrà inserita la data della richiesta e non sarà possibile modificare o eliminare il documento, potrà essere fatto solo annullando la richiesta di certificazione.Quando la certificazione viene accetta sarà possibile visualizzare i dati della certificazione tramite un modal.

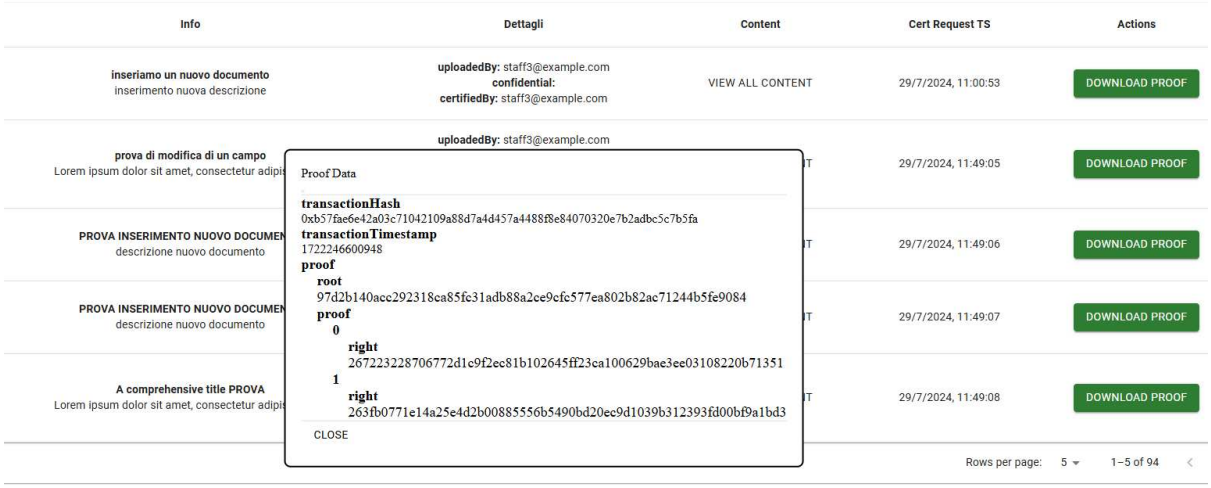

Figure 4.10: Dati della certificazione

Capitolo 5

Conclusioni e sviluppi futuri

5.1 Conclusioni

Il presente elaborato ha illustrato lo sviluppo del lato front-end di un'applicazione web che consente di gestire delle supply chain e di poter ottenere la certificazione dei documenti attraverso la blockchain. Si è partiti dallo studio della tecnologia blockchain, capire com'è strutturata e i vari componenti e protocolli che la formano.
Sono poi stati analizzati degli studi in diversi settori, che cercano di applicare la blockchain nelle supply chain per una maggiore tracciabilità e trasparenza delle transazioni.
Successivamente si è passati allo sviluppo della web application, partendo dal mockup, in cui sono state illustrate le strutture di ogni pagina, la struttura visiva del documento. Si è deciso di gestire il frontend in single page application utilizzando la libreria React. L'approccio con React è stato molto semplice, presenta una vasta documentazione, il codice è molto intuitivo e facile da gestire.
La parte più complessa da gestire, nello sviluppo dell'applicazione, è stata la costruzione di funzioni per la generazione e la visualizzazione di file JSON.

5.2 Sviluppi futuri

Delle migliorie attuabili al codice potrebbero essere riferite alla gestione di errori nel momento in cui si effettuano chiamate al sever, non gestiti con dei pop-up ma con la visualizzazione dell'errore. Magari un campo non compilato, o compilato male, sarà evidenziato in rosso e comparirà il messaggio di errore. Migliorare la visualizzazione di un documento,magari con l'immagine che compaia in alto a destra del modal. Per la parte del form di inserimento di un nuovo documento, migliorare la parte dei campi per l'inserimento dei campioni,non serve che sia tanto lungo, deve contenere solo un numero. Per la parte dei documenti di un'azienda, nella pagina miei documenti, la possibilità di ricercare dei determinati documenti utilizza dei filtri come date e parole chiave nel documento. Una funzionalità aggiuntiva può essere quella di poter ottenere la certificazione di un documento con download in pdf. Il proprietario di un documento può visualizzare quanti download della certificazione siano stati fatti.

Bibliography

[1] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. *Blockchain Technology Overview*. NIST, 2018.

[2] Nazanin Moosavi, Hamed Taherdoost, Nachaat Mohamed, Mitra Madanchian, Yousef Farhaoui, and Inam Ullah Khan. *Blockchain Technology, Structure, and Applications: A Survey*. ELSEVIER, 2020.

[3] Ziad Hussein, May A. Salama, and Sahar A. El-Rahman. *Evolution of blockchain consensus algorithms: a review on the latest milestones of blockchain consensus algorithms*. Springer Open, 2023.

[4] Hamed Taherdoost. *Smart Contracts in Blockchain Technology: A Critical Review*. MDPI, 2023.

[5] Shahinaz Kamal Ezzat, Yasmine N. M. Saleh, and Ayman A. Abdel-Hamid. *Blockchain Oracles: State-of-the-Art and Research Directions*. IEEE, 2022.

[6] Hamed Taherdoost. *Non-Fungible Tokens (NFT): A Systematic Review*. MDPI, 2023.

[7] Gokuleshwaran Narayanan, Ivan Cvitić, and S. P. Raja Dragan Peraković. *Role of Blockchain Technology in Supplychain Management*. IEEE, 2024.

[8] Kevin A. Clauson, Elizabeth A. Breeden, Cameron Davidson, and Timothy K. Mackey. *Leveraging Blockchain Technology to Enhance Supply Chain Management in Healthcare: An Exploration of Challenges and Opportunities in the Health Supply Chain*. Blockchain in Healthcare Today, 2018.

[9] Ashraf Shirani. *Blockchain for global maritime logistic*. Issues in Information Systems, 2018.

[10] Morra Luigi. *Application of Blockchain Technologies To Logistics and To Container's Transportation Industry*. Luiss Guido Carli, 2019.

[11] Oloruntobi Olakunle, Mokhtar Kasypi, Gohari Adel, Asif Saira, and Chuah Lai Fatt. *Sustainable transition towards greener and cleaner seaborne shipping industry: Challenges and opportunities*. Cleaner Engineering and Technology, 2018.

[12] Mochammad Fariz Syah Lazuardy and Dyah Anggraini. *Modern Front End Web Architectures with React.Js and Next.Js*. IRJAES, 2022.

[13] G. Fink and I. Flatow. *Introducing Single Page Applications*. Apress, Berkeley, CA, 2014.

[14] React. *Documentazione React*. React.dev, 2024.

[15] ECMA-262. *ECMAScript® Language Specification*. ECMA, 2017.

[16] Crockford, Douglas, Morningstar, and Chip. *Standard ECMA-404 The JSON Data Interchange Syntax*. ECMA, 2017.