Politecnico di Torino

POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Progettazione e sviluppo del front-end web di una cartella clinica informatizzata

Relatori
Prof.ssa Carla Fabiana CHIASSERINI
Prof. Guido PAGANA

Candidato
Antonio CAMPOBASSO

Anno Accademico 2022-2023

Sommario

Il presente lavoro di tesi si concentra sull'implementazione e lo sviluppo del front-end di un'applicazione web dedicata alla gestione informatizzata delle cartelle cliniche presso il reparto di medicina interna dell'Ospedale Molinette di Torino. L'obiettivo principale è fornire un sistema avanzato, aggiornato e scalabile che non solo soddisfi le esigenze immediate del reparto, ma che sia altresì flessibile e adattabile a eventuali modifiche future, sia in termini di funzionalità che di estensione a nuovi reparti.

La scelta della tecnologia React è stata determinante per la natura dinamica e flessibile del progetto. Il capitolo introduttivo affronta la necessità di un'applicazione moderna e scalabile, presentando React come il framework ideale per rispondere a tali esigenze.

Il nucleo della tesi è dedicato alla generazione dinamica di form, liste, tabelle e flowchart. Vengono esplorate approfonditamente le metodologie e gli strumenti impiegati, con un'analisi dettagliata dei vantaggi della programmazione reattiva e della struttura a componenti di React.

Particolare enfasi è posta sulla progettazione di un sistema che consenta la creazione e la gestione di nuove funzionalità o la modifica delle esistenti senza richiedere modifiche dirette al codice di base. L'utilizzo di JSON schemas per definire la struttura dei dati e l'approccio modulare nell'implementazione rendono l'applicazione altamente estensibile e personalizzabile.

Infine, la tesi esamina lo stato attuale del progetto, evidenziando i successi ottenuti, quali un sistema di autenticazione flessibile e la definizione automatica di percorsi diagnostici attraverso l'implementazione di flowchart. Vengono delineati anche gli sviluppi futuri, tra cui l'implementazione del Server-Side Rendering, l'estensione a dispositivi mobili e la trasformazione in una Progressive Web App per migliorare ulteriormente l'esperienza utente.

In sintesi, la tesi offre una panoramica completa del processo di sviluppo del front-end, evidenziando le scelte progettuali, gli strumenti utilizzati e i risultati ottenuti, con uno sguardo attento alle prospettive future di evoluzione del sistema.

Abstract

This thesis work focuses on the implementation and development of the front-end of a web application dedicated to computerized management of medical records in the internal medicine department of Molinette Hospital in Turin. The main goal is to provide an advanced, updated, and scalable system that not only meets the immediate needs of the department but also remains flexible and adaptable to potential future changes, both in terms of functionalities and extension to new departments.

The choice of React technology was pivotal due to the dynamic and flexible nature of the project. The introductory chapter addresses the need for a modern and scalable application, presenting React as the ideal framework to meet these requirements.

The core of the thesis is dedicated to the dynamic generation of forms, lists, tables, and flowcharts. Methodologies and tools used are thoroughly explored, along with an in-depth analysis of the advantages of reactive programming and React's component-based structure.

Particular emphasis is placed on designing a system that allows creating and managing new features or modifying existing ones without requiring direct changes to the core code. The use of JSON schemas to define data structure and a modular approach in implementation make the application highly extensible and customizable.

Lastly, the thesis examines the current state of the project, highlighting achieved successes such as a flexible authentication system and the automatic definition of diagnostic paths through flowchart implementation. Future developments, including Server-Side Rendering implementation, extension to mobile devices, and transformation into a Progressive Web App to further enhance the user experience, are outlined.

In summary, the thesis provides a comprehensive overview of the front-end development process, showcasing design choices, utilized tools, achieved outcomes, while also keeping an eye on the future prospects for system evolution.

Ringraziamenti

Questa tesi rappresenta un traguardo significativo che porterò sempre nel cuore con grande orgoglio. Tuttavia, riconosco che il merito di questo successo non è soltanto mio. È il frutto del contributo, dell'aiuto e del sostegno di molte persone che hanno condiviso con me questa esperienza. Desidero quindi ringraziare sinceramente tutti coloro che hanno contribuito a rendere possibile questo importante risultato.

Ringrazio il Prof. Pagana e il mio collega Giovanni, per avermi guidato e supportato nella fase più importante del mio percorso accademico.

Ringrazio la Fondazione LINKS per avermi dato la possibilità di svolgere il mio lavoro di tesi in un luogo interessante e dinamico, che mi ha permesso di mettermi in gioco e fare un'esperienza che sarà preziosa per il mio futuro.

La mia famiglia merita il mio più profondo ringraziamento. Nonostante la distanza di oltre 1000 chilometri, siete stati il mio sostegno costante. Ogni esame è stato vissuto insieme, con gioie e sofferenze che avete condiviso, se non superato, in parallelo alle mie emozioni. È grazie a voi se ho potuto realizzarmi in un campo che amo sin dai tempi della scuola media. Mai mi avete imposto limiti, ma al contrario, avete contribuito a superare quelli che mi ero posto. La vostra fiducia in me ha un valore inestimabile.

Ringrazio tutti i miei amici, sia quelli che sono stati al mio fianco fin dall'inizio, sia quelli che si sono aggiunti lungo il percorso. Grazie al vostro sostegno costante, ho sempre avuto la sensazione di essere a casa, ovunque mi trovassi. Non importa quanto lontani, siete rimasti presenti e vicini, specialmente nei momenti più critici. La vostra presenza è stata fondamentale e se oggi non ho rinunciato è anche grazie al vostro supporto indispensabile.

Ringrazio tutti i colleghi che ho avuto l'onore di incontrare lungo il mio percorso, in particolare i membri del Team 14. La collaborazione con voi è stata fondamentale per il mio sviluppo professionale: insieme abbiamo condiviso esperienze che hanno contribuito notevolmente al miglioramento delle mie capacità e all'acquisizione di nuove competenze. La passione e l'impegno che avete dimostrato mi hanno ispirato, e oggi non vedo l'ora di lavorare in gruppo, consapevole del valore e dell'importanza della collaborazione, grazie a voi.

Infine voglio ringraziare la città di Torino, che per questi 5 anni è stata la mia seconda casa. Non so dove mi porterà il futuro, ma spero un giorno di ritornare qui.

Indice

Glossario **7**

Elenco Tabelle **9**

Elenco Figure **10**

1 Introduzione **11**
1.1 Contesto 12
1.2 Obiettivi 13

2 L'ambiente Web **14**
2.1 Pagine web 14
2.1.1 Storia del Web 16
2.1.2 Browser 17
2.1.3 Struttura di una Web page 18
2.1.4 Il ruolo di Javascript 20
2.2 Comunicazione 21
2.2.1 Protocolli di comunicazione 21
2.2.2 L'architettura REST 22
2.2.3 Web Sockets 23
2.3 Applicazioni web 24
2.3.1 Architettura delle Web app 25
2.3.2 Front-end 25
2.3.3 Back-end 26
2.3.4 Modelli di navigazione 27
2.3.5 Ottimizzazioni 28
2.4 PWA 29
2.4.1 Realizzazione 29
2.4.2 Vantaggi 30

3 React e UX **32**
3.1 Principi chiave . . . 33
3.1.1 Componenti . . . 34
3.1.2 Props . . . 34
3.1.3 Hooks . . . 35
3.1.4 Stato . . . 35
3.2 Context . . . 36
3.2.1 Gestione dello stato globale . . . 37
3.3 Routing . . . 38
3.3.1 React Router . . . 38
3.3.2 Elementi di routing . . . 38
3.4 Effetti collaterali . . . 39
3.4.1 Chiamate API . . . 39
3.5 UX Design . . . 41
3.6 User-Centered design . . . 42
3.6.1 Fase di ricerca e analisi . . . 43
3.6.2 Fase di progettazione . . . 43
3.6.3 Processo di validazione . . . 45
3.7 Ant Design . . . 46

4 Sfide e soluzioni **48**
4.1 Interazione con il server . . . 48
4.1.1 Formato dei dati . . . 49
4.1.2 Architettura a microservizi . . . 50
4.2 Autenticazione . . . 51
4.2.1 JWT . . . 51
4.2.2 OAuth . . . 52
4.3 Validazione dei dati . . . 53
4.3.1 JSON Schema . . . 53
4.3.2 Dinamicità degli schemas . . . 55
4.4 Percorsi diagnostici . . . 55
4.4.1 Flowchart . . . 57

5 Implementazione **58**
5.1 Struttura . . . 58
5.1.1 Routing . . . 59
5.2 Pagine . . . 60
5.2.1 Login e Layout generale . . . 62
5.2.2 Pazienti . . . 62
5.2.3 Percorsi . . . 64
5.2.4 Esami e Visite . . . 65

5.3 Autenticazione 66
5.3.1 AuthContext 66
5.3.2 Processo di autenticazione 68
5.4 Middleware 69
5.4.1 Axios 69
5.4.2 Interceptors 70
5.4.3 Refresh automatico 70
5.4.4 React Query 71
5.5 Elementi dinamici 72
5.5.1 Dynamic page 72
5.5.2 Dynamic tab 73
5.5.3 Dynamic Element 73
5.6 Form 76
5.6.1 React JSONSchema Form 76
5.7 Flowchart 77
5.7.1 React Flow 78
5.8 Deployment 79
5.8.1 Docker 79

6 Conclusioni **80**
6.1 Sviluppi futuri 81

Bibliografia **82**

Sitografia **83**

Glossario

AJAX Asynchronous JavaScript And XML. 20

API Application Program Interface. 23

BSON Binary JSON. 49, 50

CSS Cascading Style Sheet. 16–18

DNS Domain Name Service. 22

DOM Document Object Model. 18, 20

FTP File Transfer Protocol. 22

HTML HyperText Markup Language. 16–18, 23

HTTP HyperText Transfer Protocol. 17, 22

IP Internet Protocol. 21, 22

JSON JavaScript Object Notation. 22, 23

JWT JSON Web Tokens. 51, 66

MPA Multi-Page Application. 27, 28, 58

REST REpresentational State Transfer. 22, 23

SEO Search Engine Optimization. 28, 58, 59

SMTP Simple Mail Transfer Protocol. 22

SPA Single Page Application. 27

SQL Structured Query Language. 49

SSR Server Side Rendering. 28, 59

TCP Transfer Control Protocol. 21–23

URL Uniform Resource Locator. 17

Elenco Tabelle

4.1 Architettura del server 50

5.1 Routing dell'applicazione 61

Elenco Figure

1.1 Logo della fondazione LINKS 11

2.1 Modello client-server 15
2.2 Esempio di DOM di una pagina web 19
2.3 Comunicazione attraverso lo stack TCP/IP 21
2.4 Differenze tra applicazione web e sito web 25
2.5 Funzionamento offline delle PWA 30

3.1 Logo di React 33
3.2 Differenze tra UX design e UI design 41
3.3 Esempio di wireframe, mockup e prototipo 44
3.4 Esempio di pagina web realizzata con Ant Design 46

4.1 Struttura di un Json Web Token 52
4.2 Esempio di percorso diagnostico 56

5.1 Menu di navigazione dell'applicazione 60
5.2 Pagina dei pazienti 63
5.3 Selezione di un esame per un nodo del percorso diagnostico 64
5.4 Decodifica del payload di un JWT 67
5.5 DynamicTab generato per la sezione anagrafe di un paziente 74
5.6 Form di un esame ABPM generato in maniera dinamica 77
5.7 Esempio di flowchart generato dinamicamente 78

Capitolo 1

Introduzione

Nel complesso e dinamico panorama dei servizi sanitari, l'adozione di soluzioni informatiche innovative è diventata essenziale per migliorare l'assistenza ai pazienti e ottimizzare l'efficienza operativa negli ospedali.

Una delle sfide principali è la separazione tra le varie unità ospedaliere e persino tra i reparti interni dello stesso ospedale. Questa frammentazione crea spesso barriere alla condivisione tempestiva e accurata delle informazioni sui pazienti, ostacolando la continuità delle cure e aumentando il rischio di errori medici. La necessità di superare queste divisioni e creare un sistema integrato per la gestione delle informazioni cliniche è diventata urgente.

Nel cuore di questa trasformazione, la digitalizzazione delle cartelle cliniche rappresenta un passo fondamentale per garantire l'accesso rapido e affidabile alle informazioni sanitarie.

La Fondazione LINKS [17], in collaborazione con il reparto di medicina interna dell'Ospedale Molinette di Torino, ha intrapreso una missione audace e innovativa per rivoluzionare il modo in cui le informazioni cliniche vengono gestite negli

Figure 1.1: Logo della fondazione LINKS
`https://www.linkedin.com/company/linksfoundation/?originalSubdomain=it`

ospedali. Riconoscendo le sfide legate alla frammentazione delle informazioni e alla mancanza di interoperabilità nei sistemi tradizionali, la Fondazione LINKS ha deciso di adottare un approccio radicalmente nuovo: una web-app avanzata progettata per essere altamente flessibile e facilmente accessibile su diverse piattaforme, inclusi dispositivi mobili.

Questo progetto non è solo un esempio di modernizzazione tecnologica; è un pilastro per la costruzione di un sistema sanitario più integrato e umano.

1.1 Contesto

L'Ospedale Molinette, come molti altri ospedali in Italia e nel mondo, affronta sfide complesse nel campo della gestione delle informazioni cliniche. La frammentazione dei dati e la mancanza di interoperabilità tra i vari reparti, in combinazione con l'esplosione del carico di lavoro, hanno reso urgente la necessità di un cambiamento radicale nel modo in cui le informazioni sanitarie vengono gestite.

Ciò che rende particolarmente problematico l'utilizzo dell'attuale software, noto come *hypermacondo*, è la sua mancanza di centralizzazione dei dati. In un ambiente ospedaliero in cui l'accesso rapido e preciso alle informazioni è cruciale, la mancanza di un sistema dati centralizzato comportava la necessità di aggiornamenti manuali costanti. Questo non solo richiedeva un considerevole sforzo da parte del personale medico, ma aumentava anche il rischio di errori umani e di dati non sincronizzati tra i vari computer del reparto. Inoltre, *hypermacondo* risulta un software datato e mal organizzato, con dati spesso invalidi e ridondanti che rendevano difficile per i medici ottenere una visione accurata e completa delle informazioni del paziente.

Oltre a queste sfide, il software è vincolato a sistemi operativi obsoleti. Ciò significa che l'accesso alle informazioni sanitarie è limitato a specifici computer e dispositivi, rendendo estremamente difficile la condivisione dei dati tra i vari reparti dell'ospedale. Questa limitata accessibilità non solo rallenta il processo decisionale ma crea anche un'esperienza utente insoddisfacente per i medici, che dovevano navigare attraverso un labirinto di interfacce obsolete e inefficienti.

Di fronte a questa situazione critica, è diventato imperativo per l'Ospedale Molinette adottare una soluzione più moderna e efficiente. La creazione di una web app dedicata, progettata specificamente per soddisfare le esigenze del reparto di medicina interna, rappresenta un passo fondamentale verso la rimodernizzazione del sistema di gestione delle informazioni cliniche. Una web app centralizzata e interoperabile garantirà un accesso più agevole ai dati, riducendo il carico di lavoro del personale medico e migliorando significativamente l'efficienza operativa del reparto. Questo nuovo approccio non solo semplificherà la gestione delle informazioni sanitarie ma migliorerà anche l'esperienza complessiva degli operatori

sanitari, consentendo loro di concentrarsi pienamente sulla cura dei pazienti senza essere ostacolati dalle limitazioni del software esistente.

1.2 Obiettivi

L'obiettivo centrale di questa ricerca è progettare e sviluppare un front-end web per una cartella clinica informatizzata che risolva non solo le esigenze specifiche del reparto di medicina interna dell'Ospedale Molinette, ma anche le sfide più ampie legate alla comunicazione interospedaliera e intra-repartimentale.

Tali obiettivi includono:

- **L'analisi approfondita delle esigenze degli utenti**, in particolare con medici, infermieri e altri operatori sanitari per identificare le specifiche necessità di gestione delle informazioni cliniche e comprendere le sfide quotidiane incontrate nel processo di documentazione e accesso alle informazioni.
- **La progettazione di un'interfaccia utente intuitiva e funzionale** con un design che semplifichi il processo di inserimento dati e accesso alle informazioni, garantendo al contempo un'esperienza utente senza intoppi e orientata all'efficienza.
- **La creazione di una base generale per implementazioni future**, affinchè possa essere utilizzata anche da altri reparti interni dell'Ospedale Molinette e, idealmente, da altri ospedali. Questo obiettivo favorisce l'interoperabilità all'interno dell'ospedale e tra ospedali diversi, promuovendo la condivisione delle informazioni cliniche in modo sicuro ed efficiente.

Capitolo 2

L'ambiente Web

Nella società moderna, il web è diventato un pilastro fondamentale su cui si basano le attività quotidiane, le comunicazioni, l'istruzione, il commercio e molte altre sfere della vita umana. La crescente dipendenza dalle tecnologie web riflette il bisogno incessante di connettività e l'accesso immediato alle informazioni in tempo reale.

L'importanza del web si manifesta in diverse forme. Per le organizzazioni, il web rappresenta un veicolo vitale per la comunicazione, consentendo una connessione istantanea con clienti, partner e dipendenti in tutto il mondo. La necessità di accedere in tempo reale agli stessi dati da qualsiasi parte del mondo è diventata cruciale per l'efficienza operativa e la presa di decisioni informate. Grazie alle applicazioni web avanzate e alle infrastrutture di rete potenziate, le organizzazioni possono raccogliere, elaborare e condividere dati in modo rapido ed efficace, rendendo possibile una collaborazione globale senza precedenti.

Nell'era digitale, l'accesso immediato alle informazioni è diventato una necessità per i singoli individui. Il web offre un'ampia gamma di risorse educative, consentendo l'apprendimento continuo e l'accesso a conoscenze provenienti da tutto il mondo. Le persone dipendono dal web per accedere ai servizi pubblici, per gestire le proprie finanze, per lo shopping online e per restare in contatto con amici e familiari. L'interconnessione globale fornita dal web ha creato una società in cui la comunicazione e lo scambio di informazioni avvengono senza confini geografici, riducendo le distanze e aprendo nuove opportunità.

2.1 Pagine web

Le pagine web sono il fondamento di Internet, consentendo agli utenti di accedere e condividere una vasta gamma di contenuti online. Una pagina web è un documento elettronico visualizzato in un browser web, che può contenere testi, immagini,

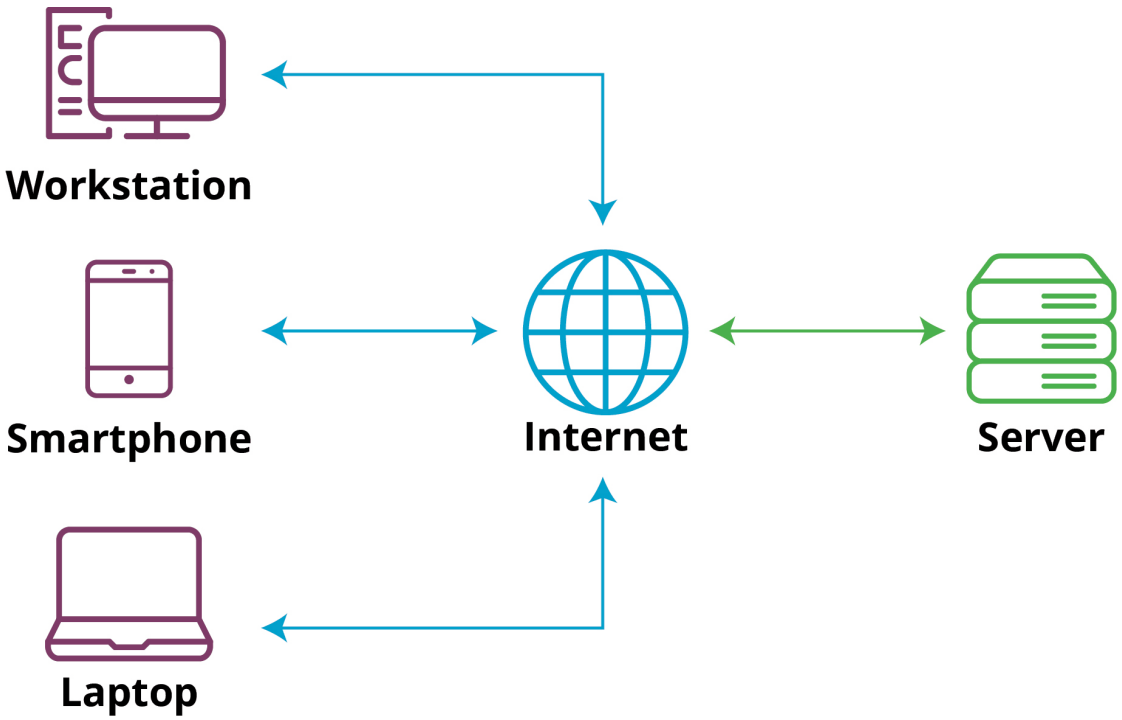

Figure 2.1: Modello client-server
`https://www.liquidweb.com/blog/client-server-architecture/`

video, collegamenti ipertestuali e interattività. Le pagine web sono essenziali per la condivisione di informazioni, l'intrattenimento, il commercio e la comunicazione in tutto il mondo.

Un concetto chiave nel mondo delle pagine web è la separazione tra client e server. Questo modello di architettura divide le responsabilità tra due componenti principali:

- **Client:** Il client è l'utente che utilizza un browser web per accedere alle pagine web. Il browser richiede le pagine al server, interpreta il codice HTML, esegue i file di stile CSS e gli script JavaScript per visualizzare il contenuto e gestire l'interattività.
- **Server:** Il server è un computer remoto che ospita le pagine web e fornisce i dati richiesti al client. Il server è responsabile della gestione delle richieste, dell'elaborazione delle pagine web e dell'invio delle risposte al client attraverso Internet.

La separazione tra client e server consente un'esperienza utente fluida e scalabile. Il client può essere un computer, un tablet o uno smartphone, mentre il server è un potente sistema informatico che ospita le risorse web. Questo modello consente agli utenti di accedere alle pagine web da qualsiasi dispositivo connesso a Internet, rendendo le informazioni e i servizi online facilmente accessibili a livello globale.

2.1.1 Storia del Web

Negli anni '80, Tim Berners-Lee, ricercatore presso il CERN[1], propose un sistema di gestione delle informazioni basato su un concetto chiamato World Wide Web [4]. Nel 1989, insieme al suo collega Robert Cailliau, Berners-Lee progettò l'HyperText Markup Language (HTML), il linguaggio di marcatura che consente la creazione di pagine web basate su link ipertestuali. Questo ha permesso agli utenti di navigare facilmente tra diverse pagine, aprendo la strada alla condivisione e alla disseminazione di informazioni su scala globale.

Successivamente, nel 1996, è stato introdotto il Cascading Style Sheet (CSS), un linguaggio di stile che ha permesso ai designer di separare il contenuto di una pagina web dalla sua presentazione. Questo ha portato a un notevole miglioramento nella progettazione e nell'aspetto delle pagine web, permettendo una maggiore personalizzazione e flessibilità nel design.

Inizialmente, le pagine web venivano create e condivise all'interno di reti locali o aziendali, consentendo agli utenti di accedere a informazioni specifiche all'interno

[1]CERN=Organizzazione europea per la ricerca nucleare

di un'organizzazione. Questo approccio limitato ha fornito le basi per l'adozione di concetti come l'HTML e i browser all'interno di ambienti controllati.

Tuttavia, la vera rivoluzione è iniziata quando Internet è diventato pubblico e accessibile a livello globale. Questo passaggio critico è avvenuto all'inizio degli anni '90, aprendo la porta alla condivisione di pagine web su una scala senza precedenti. Con l'espansione di infrastrutture di rete e l'aumento dell'accessibilità a Internet, le pagine web hanno cominciato a essere ospitate su server remoti, rendendole accessibili a chiunque in tutto il mondo. Questo cambiamento ha richiesto l'adozione di standard aperti e protocolli di comunicazione, tra cui l'HTTP, che ha permesso la trasmissione affidabile di dati attraverso la rete.

Nel frattempo, il mondo dei browser web stava vivendo una rapida evoluzione. Uno dei primi browser popolari fu Netscape Navigator, lanciato nel 1994. Netscape Navigator ha giocato un ruolo chiave nel rendere il web accessibile al grande pubblico, introducendo molte delle caratteristiche che oggi diamo per scontate, come le finestre multiple e i segnalibri.

Negli anni '90, un'altra svolta importante è stata l'introduzione di JavaScript, sviluppato da Brendan Eich per Netscape Communications. JavaScript è diventato il primo linguaggio di scripting lato client ampiamente utilizzato e ha aperto la strada a interazioni dinamiche sul web. Questa nuova capacità ha permesso agli sviluppatori di creare pagine web interattive e dinamiche, migliorando l'esperienza degli utenti e aprendo la strada a nuove possibilità di sviluppo web.

L'avvento di queste tecnologie e concetti ha gettato le basi per l'espansione continua del web, portando a una vasta gamma di sviluppi e innovazioni nel campo della tecnologia web, che continueranno a evolversi nel futuro.

2.1.2 Browser

I browser web rappresentano il ponte essenziale tra gli utenti e il vasto universo di informazioni disponibili online. Queste applicazioni software, come Google Chrome, Mozilla Firefox o Safari, svolgono un ruolo fondamentale nel renderci accessibili il mondo del web.

Quando un utente inserisce un Uniform Resource Locator (URL) (es. `www.google.com`) nel browser esso avvia un processo intricato di comunicazione con il server per ottenere la pagina web richiesta. Questo processo coinvolge diversi protocolli e tecnologie per garantire una connessione sicura e affidabile.

Una volta stabilita la connessione, il browser invia una richiesta HTTP al server. Questa richiesta contiene informazioni sul tipo di risorsa richiesta, come una pagina HTML, un'immagine o un file CSS. Il server riceve la richiesta e prepara la risposta.

Una volta che il browser riceve la risposta dal server, inizia il processo di rendering della pagina web. Questo coinvolge l'interpretazione del codice HTML

per creare la struttura della pagina, l'applicazione di stili CSS per il layout e l'interattività, e l'esecuzione di script JavaScript per aggiungere funzionalità dinamiche.

Durante il rendering di una pagina web, il browser può caricare ulteriori file esterni. Questi includono immagini, video, font personalizzati, script aggiuntivi per l'interattività dinamica. Questi file vengono richiesti e scaricati dal browser per integrare il contenuto nella pagina.

Oltre a facilitare l'accesso alle informazioni online, i browser svolgono un ruolo cruciale nella sicurezza online. Introducono protocolli critici come HTTPS, che crittografa i dati scambiati tra il browser e il server web, proteggendo così la privacy degli utenti durante le transazioni online e la condivisione di informazioni sensibili. Inoltre, i browser implementano misure di sicurezza come sandboxing e controlli degli accessi, che aiutano a prevenire attacchi informatici e a mantenere gli utenti al sicuro mentre navigano online.

2.1.3 Struttura di una Web page

Le pagine web vengono costruite a partire dai file HTML, CSS e JavaScript. L'HTML fornisce la struttura di base del contenuto, mentre il CSS definisce l'aspetto visivo, come il layout, i colori e i tipi di carattere. JavaScript, d'altra parte, aggiunge interattività e dinamicità alle pagine web, consentendo agli sviluppatori di creare esperienze utente più ricche e coinvolgenti [9].

Le pagine web seguono una struttura standardizzata, suddividendosi in due parti principali: `<head>` e `<body>`. Questa suddivisione organizza il contenuto e le informazioni in modo coerente, migliorando l'accessibilità e l'esperienza utente complessiva.

Il tag `<head>` contiene metadati e collegamenti a risorse esterne. Questi elementi forniscono informazioni cruciali ai browser e agli utenti, influenzando aspetti come la presentazione, l'ottimizzazione per i motori di ricerca e le interazioni sociali.

Il tag `<body>` contiene tutto il contenuto visibile agli utenti, come testo, immagini, video e altri elementi multimediali. Questo è il cuore della pagina web, dove gli utenti interagiscono con il contenuto.

Ogni elemento HTML sulla pagina è rappresentato come un nodo nel DOM, una struttura dati ad albero che riporta la struttura gerarchica di una pagina web. Il Document Object Model (DOM) può essere modificato in tempo reale utilizzando JavaScript, consentendo agli sviluppatori di creare interattività, animazioni e dinamicità nella pagina web senza dover ricaricare completamente la pagina.

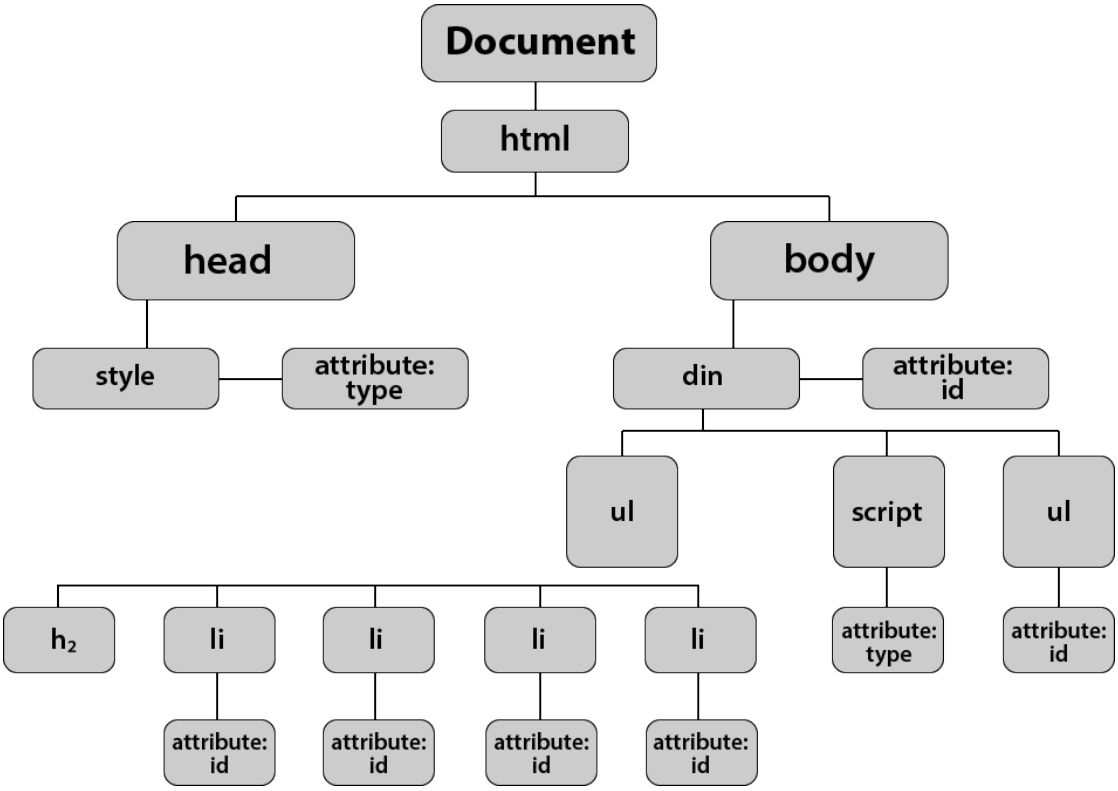

Figure 2.2: Esempio di DOM di una pagina web
`https://shorturl.at/fvAB8`

2.1.4 Il ruolo di Javascript

JavaScript è emerso come uno dei linguaggi di programmazione più influenti e diffusi al mondo, con una storia ricca e complessa che si intreccia con lo sviluppo del World Wide Web. Creato da Brendan Eich mentre lavorava per Netscape Communications Corporation, JavaScript vide la luce nel lontano 1995. Inizialmente concepito come un linguaggio di scripting lato client per migliorare l'interattività delle pagine web, JavaScript ha rapidamente guadagnato popolarità grazie alla sua natura versatile e alle potenti funzionalità di manipolazione del DOM.

Il nome "JavaScript" potrebbe trarre in inganno, poiché è solo lontanamente correlato al linguaggio di programmazione Java. Tuttavia, la scelta del nome è stata in parte dovuta a considerazioni di marketing, approfittando del crescente interesse verso Java in quel periodo. JavaScript è stato originariamente introdotto in Netscape Navigator 2.0 nel 1995, portando un nuovo livello di dinamicità e interattività al web, che prima era in gran parte statico [3].

Nel corso degli anni, JavaScript è diventato uno standard de facto per lo sviluppo web, consentendo agli sviluppatori di creare una vasta gamma di applicazioni, da semplici pagine interattive a complessi framework di sviluppo front-end. Con l'introduzione di Asynchronous JavaScript And XML (AJAX), JavaScript ha permesso la creazione di applicazioni web in tempo reale, ridefinendo l'esperienza dell'utente online e aprendo la strada a applicazioni web complesse come Gmail e Google Maps.

Per garantire la coerenza nel linguaggio e per evitare frammentazioni, JavaScript è stato standardizzato dall'ECMAScript [16], un comitato che stabilisce gli standard per il linguaggio. L'ECMAScript ha guidato l'evoluzione di JavaScript attraverso varie versioni, introducendo nuove funzionalità, sintassi migliorata e prestazioni ottimizzate. Queste standardizzazioni hanno contribuito a consolidare JavaScript come uno dei linguaggi di programmazione più affidabili e potenti per lo sviluppo web.

Negli anni più recenti, JavaScript si è arricchito di nuove funzionalità e concetti, come le promesse per la gestione asincrona, gli arrow function per una sintassi più concisa e il sistema di moduli per una migliore organizzazione del codice. Inoltre, l'avvento di Node.js ha consentito agli sviluppatori di utilizzare JavaScript anche lato server, ampliando ulteriormente l'ecosistema di questo linguaggio dinamico e versatile.

In conclusione, JavaScript non è solo il linguaggio di scripting che ha alimentato l'interattività del web, ma è anche un linguaggio che ha visto una continua evoluzione e adattamento, rimanendo al passo con le esigenze dei moderni sviluppatori web e contribuendo in modo significativo all'ecosistema digitale in costante cambiamento.

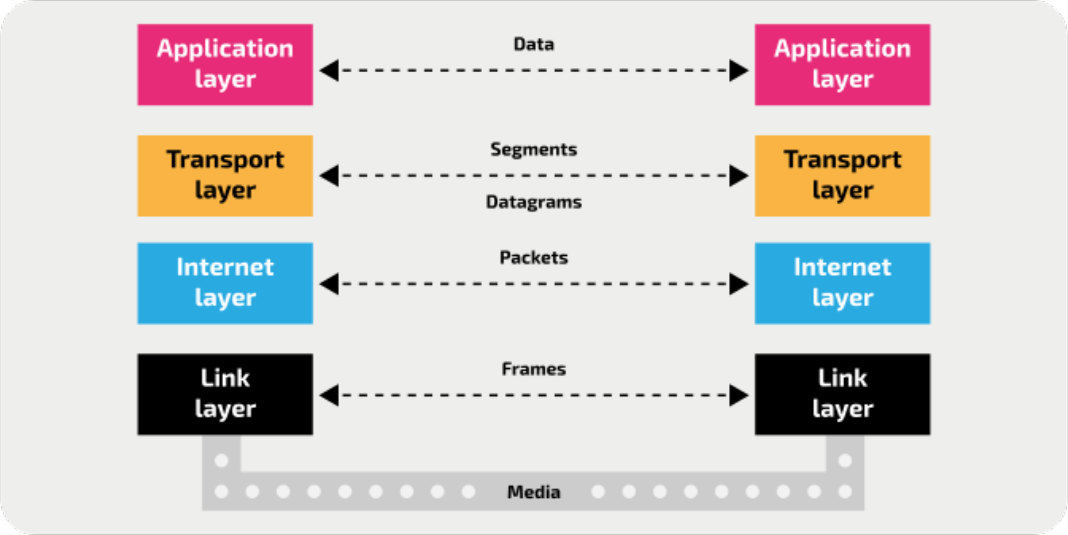

Figure 2.3: Comunicazione attraverso lo stack TCP/IP
`https://shorturl.at/enpC2`

2.2 Comunicazione

La comunicazione sul web è il risultato di una complessa infrastruttura di rete che connette milioni di dispositivi in tutto il mondo, permettendo la condivisione istantanea di informazioni e dati. Per comprendere questo processo, è fondamentale avere una conoscenza approfondita delle reti informatiche e dei protocolli che le guidano.

Una rete informatica è un insieme di dispositivi e componenti collegati tra loro per condividere risorse e informazioni. Questi dispositivi possono essere collegati attraverso cavi, fibre ottiche o reti wireless. Ogni dispositivo in una rete ha un indirizzo univoco, chiamato indirizzo Internet Protocol (IP), che viene utilizzato per identificarlo nella rete.

Le reti possono essere connesse tra loro attraverso dispositivi chiamati router, che indirizzano il traffico tra reti diverse. Questo processo consente alle macchine su una rete di comunicare con quelle su altre reti, formando così l'infrastruttura di base di Internet.

2.2.1 Protocolli di comunicazione

Internet è la più grande rete di reti, un vasto sistema di connessioni interconnesse globalmente. L'architettura di rete predominante utilizzata su Internet è lo stack di protocolli TCP/IP, che gestisce la trasmissione di dati attraverso le reti.

Questo stack è diviso in quattro livelli [2], ciascuno dei quali svolge ruoli specifici nel processo di comunicazione e trasmissione dei dati:

- **Livello Link:** noto anche come livello di accesso alla rete, è responsabile dell'indirizzamento fisico dei dati all'interno di una rete locale. Gestisce la trasmissione dei frame di dati attraverso il collegamento fisico, garantendo che i dati siano inviati e ricevuti correttamente tra dispositivi sulla stessa rete locale. Questo livello coinvolge protocolli come Ethernet e Wi-Fi per la trasmissione fisica dei dati.

- **Livello Internet:** è fondamentale per il routing dei dati attraverso reti diverse. Questo livello utilizza gli indirizzi IP per identificare i dispositivi sulla rete e gestisce il routing dei pacchetti di dati tra le reti. Il protocollo principale utilizzato a questo livello è l'IP, che consente l'instradamento affidabile dei dati su Internet.

- **Livello Trasporto:** è responsabile di fornire una trasmissione affidabile dei dati tra i dispositivi su reti diverse. Gestisce la suddivisione dei dati in pacchetti più piccoli, la loro trasmissione e l'assicurarsi che arrivino correttamente a destinazione. Uno dei protocolli più importanti di questo livello è il TCP, che offre una trasmissione affidabile attraverso la conferma di ricezione e la ritrasmissione dei dati se necessario.

- **Livello Applicazione:** Il livello Applicazione è il livello più alto dello stack TCP/IP ed è responsabile di fornire i servizi direttamente agli utenti o alle applicazioni. Questo livello coinvolge una vasta gamma di protocolli che supportano servizi come la navigazione web tramite HyperText Transfer Protocol (HTTP), l'invio di posta elettronica tramite Simple Mail Transfer Protocol (SMTP), il trasferimento di file grazie al File Transfer Protocol (FTP) o la traduzione di indirizzi web in indirizzi IP con Domain Name Service (DNS). Ogni protocollo applicativo è progettato per scopi specifici e offre funzionalità specifiche agli utenti finali o alle applicazioni.

2.2.2 L'architettura REST

L'architettura REpresentational State Transfer (REST) è un approccio fondamentale per progettare servizi web, basato sui principi di semplicità, scalabilità e interoperabilità [1]. REST utilizza i metodi HTTP per operare sulle risorse, che possono essere rappresentate in diversi formati, come pagine web HTML o oggetti JavaScript Object Notation (JSON), facilitando la comunicazione tra client e server. REST fa ampio uso dei metodi HTTP per operare sulle risorse. I quattro metodi principali utilizzati in REST sono:

- **GET:** Utilizzato per recuperare una rappresentazione di una risorsa dal server.
- **POST:** Utilizzato per creare una nuova risorsa sul server.
- **PUT:** Utilizzato per aggiornare una risorsa esistente sul server o creare una risorsa se non esiste.
- **DELETE:** Utilizzato per rimuovere una risorsa dal server.

Nel contesto di REST, i dati possono essere scambiati in vari formati. Le pagine web HTML rappresentano un formato comune, ma JSON è diventato particolarmente popolare per la sua semplicità e flessibilità. JSON è un formato di dati leggero e facilmente interpretabile, basato su coppie chiave-valore. Viene utilizzato per rappresentare oggetti e strutture dati in modo coerente e può essere facilmente letto e scritto da qualsiasi linguaggio di programmazione. I dati JSON possono essere incorporati nelle risposte delle API RESTful, consentendo una facile manipolazione e interpretazione dei dati da parte degli sviluppatori.

Le Application Program Interface (API) sono insiemi di regole e protocolli che consentono a software diversi di comunicare tra loro. Le API RESTful seguono l'architettura REST e utilizzano i metodi HTTP per fornire un'interfaccia standardizzata attraverso la quale le applicazioni possono interagire tra loro. Le API RESTful permettono ai client di effettuare richieste (utilizzando i metodi HTTP) e ricevere risposte in formato leggibile (come HTML o JSON) dalle risorse del server.

2.2.3 Web Sockets

I WebSocket rappresentano una tecnologia avanzata per consentire la comunicazione bidirezionale e in tempo reale tra client e server su Internet. A differenza del tradizionale modello di richiesta/risposta basato su HTTP, i WebSocket creano un canale di comunicazione persistente e a bassa latenza, che permette ai dati di fluire liberamente tra client e server senza dover inviare richieste multiple.

Il funzionamento dei WebSocket è basato su un protocollo di comunicazione a livello applicativo, che stabilisce una connessione TCP persistente tra il client e il server [27]. Dopo l'handshake iniziale, in cui viene stabilito il protocollo WebSocket, entrambe le parti possono inviare e ricevere dati in modo asincrono, senza dover attendere una richiesta specifica da parte del client. Questo flusso di dati bidirezionale rende i WebSocket ideali per le applicazioni che richiedono aggiornamenti in tempo reale, come:

1. **Chat online:** I WebSocket consentono una comunicazione istantanea e in tempo reale tra gli utenti, rendendoli ideali per le applicazioni di chat online. Gli utenti possono inviare messaggi istantanei senza dover aggiornare la pagina o inviare richieste al server.

2. **Giochi multigiocatore:** Nei giochi online multigiocatore, i WebSocket permettono una sincronizzazione continua dei dati di gioco tra tutti i giocatori. Questo consente una reattività immediata alle azioni dei giocatori, creando un'esperienza di gioco fluida e coinvolgente.

3. **Streaming di dati in tempo reale:** I servizi di streaming video, audio o dati possono utilizzare i WebSocket per trasmettere dati in tempo reale ai client, offrendo una visualizzazione in tempo reale senza buffering o interruzioni.

4. **Applicazioni collaborative:** Le applicazioni collaborative, come gli strumenti di elaborazione del testo in tempo reale o le applicazioni di condivisione di file, possono utilizzare i WebSocket per garantire che tutte le modifiche fatte da un utente siano immediatamente visibili agli altri utenti connessi.

2.3 Applicazioni web

Nel panorama digitale odierno, la distinzione tra una pagina web tradizionale e una web app risiede nell'esperienza utente che offrono e nelle funzionalità che possono essere implementate.

Le pagine web tradizionali sono come documenti digitali. Composte principalmente da HTML, contengono testo, immagini, link e altre risorse multimediali. Queste pagine, solitamente visualizzate in un browser web, sono statiche nel senso che il loro contenuto è fisso, non cambia dinamicamente e non consente interazioni significative da parte dell'utente. Sono ideali per visualizzare informazioni statiche, come articoli, documentazione o contenuti informativi.

Le web app, d'altra parte, vanno oltre la staticità delle pagine web tradizionali. Sono applicazioni interattive che offrono una vasta gamma di funzionalità simili a quelle delle applicazioni desktop. Le web app consentono agli utenti di compiere azioni complesse, manipolare dati, interagire in tempo reale e personalizzare l'esperienza in base alle loro esigenze. Queste applicazioni utilizzano tecnologie avanzate come JavaScript per creare interattività dinamica e comunicazione asincrona con i server, consentendo agli utenti di eseguire operazioni complesse senza dover ricaricare l'intera pagina. Le web app sono ideali per servizi online interattivi come applicazioni di gestione, social network, piattaforme di e-commerce e strumenti di produttività.

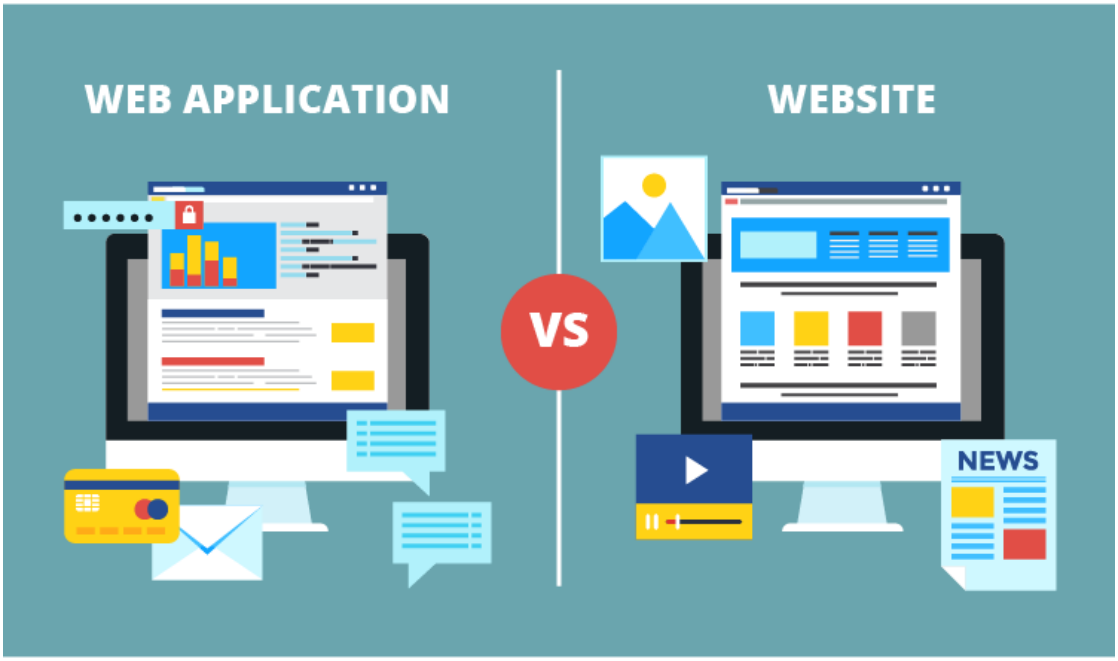

Figure 2.4: Differenze tra applicazione web e sito web
`https://shorturl.at/djJO6`

2.3.1 Architettura delle Web app

Per comprendere appieno il loro funzionamento, è essenziale esaminare l'architettura su cui si basano. Le applicazioni web sono strutturate in modo intricato, con due componenti cruciali che lavorano sinergicamente per offrire un'esperienza utente fluida e coinvolgente: il front-end e il back-end [10]. L'integrazione sinergica tra front-end e back-end è essenziale per creare applicazioni web potenti e intuitive.

2.3.2 Front-end

Il front-end di un'applicazione web si occupa della presentazione dell'interfaccia utente e dell'interazione con gli utenti, rendendo l'applicazione visivamente accattivante, intuitiva e funzionale.

Questo componente è responsabile di tradurre il design e le specifiche dell'utente in una forma visibile e interattiva. Si occupa di garantire che l'interfaccia utente sia intuitiva e facile da usare, permettendo agli utenti di interagire agevolmente con l'applicazione. Inoltre, il front-end deve assicurare che l'applicazione sia responsiva, adattandosi in modo fluido a diverse dimensioni di schermi, dispositivi e orientamenti. Tra i frameworks per lo sviluppo di applicativi Front-End spiccano:

- **React.js:** creato da Facebook, React.js è un framework front-end basato

su componenti, che facilita la creazione di interfacce utente dinamiche e riutilizzabili. La sua architettura agevola la gestione dei dati e degli stati dell'applicazione, rendendolo ampiamente utilizzato per la creazione di Applicazioni Web.

- **Angular:** sviluppato da Google, Angular è un framework completo che offre un'ampia gamma di funzionalità, tra cui il two-way data binding e l'iniezione di dipendenze. Angular è adatto per applicazioni complesse e offre strumenti robusti per la creazione di interfacce utente sofisticate.

- **Vue.js:** framework progressivo e adattabile, con una sintassi intuitiva. È noto per la sua flessibilità e facilità d'uso, consentendo agli sviluppatori di integrare progressivamente le sue funzionalità in progetti esistent

2.3.3 Back-end

Il back-end di un'applicazione web rappresenta il cuore operativo del sistema, gestendo tutte le operazioni di elaborazione dei dati, l'accesso al database e le logiche applicative. Mentre il front-end si occupa dell'interfaccia utente, il back-end lavora dietro le quinte per garantire che l'applicazione funzioni in modo efficiente e sicuro. Tra i componenti chiave del Back-End troviamo:

- **Server:** Il server è il componente centrale del back-end. Si occupa di gestire le richieste provenienti dal front-end, eseguendo le operazioni necessarie e restituendo le risposte appropriate. I server web popolari includono Apache, Nginx e Microsoft IIS.

- **Database:** I database sono utilizzati per archiviare e gestire i dati dell'applicazione. Possono essere di diversi tipi, come database relazionali (come MySQL e PostgreSQL) o database NoSQL (come MongoDB e Cassandra), a seconda delle esigenze specifiche del progetto.

- **Logica applicativa:** La logica applicativa, spesso implementata attraverso linguaggi di programmazione come Python, Ruby, Java, PHP o JavaScript (utilizzando Node.js), gestisce le operazioni complesse e le regole di business dell'applicazione. Questa componente si occupa di processi quali l'autenticazione degli utenti, la gestione delle sessioni e la manipolazione dei dati.

- **API:** Le API consentono la comunicazione tra il front-end e il back-end, consentendo lo scambio di dati e richieste. Le API sono fondamentali per l'integrazione di servizi esterni e l'accesso a funzionalità specifiche dell'applicazione.

Il back-end si occupa di una serie di compiti fondamentali, tra cui la gestione dei dati, l'elaborazione delle richieste utente, l'autenticazione, l'autorizzazione e la gestione degli errori. Queste operazioni sono essenziali per garantire un'esperienza utente sicura, coesa e senza intoppi. Il back-end è responsabile anche della sicurezza dell'applicazione, proteggendo i dati sensibili e prevenendo possibili vulnerabilità. Esistono diversi frameworks per lo sviluppo di Back-End e differiscono sia nell'architettura utilizzata che nel linguaggio:

- **Express.js:** framework per Node.js che semplifica la creazione di server web e API. È noto per la sua semplicità e flessibilità, rendendolo una scelta popolare per lo sviluppo back-end.
- **Django:** un framework back-end per Python che segue il principio "batterie incluse". Offre una vasta gamma di funzionalità integrate, come un ORM potente e un sistema di autenticazione sicuro, facilitando lo sviluppo di applicazioni web complesse.
- **Spring Boot:** framework back-end per Java progettato per semplificare lo sviluppo di applicazioni web e servizi. Creato dal team di Spring, il famoso framework di sviluppo per Java, Spring Boot offre una piattaforma potente e flessibile per gli sviluppatori, consentendo loro di creare applicazioni Java robuste, scalabili e altamente performanti con una facilità di sviluppo significativamente migliorata.

2.3.4 Modelli di navigazione

Due approcci principali delineano il modo in cui gli utenti interagiscono con le applicazioni online: le Single Page Application (SPA) e le Multi-Page Application (MPA). Questi modelli di navigazione presentano differenze significative che influenzano l'esperienza dell'utente e le prestazioni delle applicazioni.

Le SPA sono applicazioni web che caricano una sola pagina HTML e aggiornano dinamicamente il contenuto quando gli utenti interagiscono con l'applicazione.

Utilizzando tecnologie come React.js, Angular o Vue.js, le SPA caricano inizialmente una singola pagina e, successivamente, aggiornano solo le parti necessarie della pagina durante l'interazione dell'utente, senza dover ricaricare l'intera pagina. Questo modello offre un'esperienza utente più fluida e reattiva, riducendo il tempo di caricamento e migliorando la percezione di velocità.
Vantaggi delle SPA:

- Esperienza utente interattiva e dinamica.
- Riduzione dei tempi di caricamento grazie al caricamento differito dei contenuti.

- Struttura di navigazione senza soluzione di continuità.

Svantaggi delle SPA:

- Maggiore complessità nello sviluppo e nel debugging, specialmente per applicazioni complesse.
- Possibilità di caricare grandi quantità di JavaScript all'avvio, influendo sui tempi di caricamento iniziale.

Le MPA sono applicazioni web tradizionali in cui ogni azione o interazione dell'utente comporta il caricamento di una nuova pagina HTML completa dal server. In questo modello, ogni sezione o pagina dell'applicazione è un file HTML separato. Le MPA sono costruite utilizzando l'approccio classico di navigazione web, dove ogni link o azione dell'utente carica una pagina web separata.
Vantaggi delle MPA:

- Search Engine Optimization (SEO) più agevole, poiché ogni pagina ha un URL separato.
- Semplicità nello sviluppo, specialmente per applicazioni con una struttura di navigazione gerarchica.

Svantaggi delle MPA:

- Tempi di caricamento più lunghi, poiché ogni interazione richiede il caricamento di una nuova pagina.
- Esperienza utente meno reattiva a causa del ricaricamento completo della pagina.

In conclusione, la scelta tra SPA e MPA dipende dalle esigenze specifiche dell'applicazione. Le SPA offrono un'esperienza utente più interattiva e veloce, mentre le MPA sono più adatte per applicazioni che richiedono una struttura di navigazione chiara e una migliore indicizzazione sui motori di ricerca. La selezione del modello appropriato è essenziale per garantire un'esperienza utente ottimale e soddisfacente.

2.3.5 Ottimizzazioni

Mentre le Multi-Page Application (MPA) possono offrire un'esperienza utente tradizionale e familiare, spesso presentano sfide legate ai tempi di caricamento e alla reattività, soprattutto quando si tratta di contenuti dinamici. In questo contesto, l'implementazione del Server Side Rendering (SSR) emerge come una soluzione strategica che risolve molte delle problematiche delle MPA [5].

Quando si utilizza SSR in un'applicazione web, il server genera il markup HTML completo per ciascuna pagina richiesta dal browser, inviando il contenuto già reso al dispositivo dell'utente. Questo approccio è particolarmente utile per le MPA, in quanto risolve il problema del ritardo del rendering su dispositivi o connessioni di rete più lente. Gli utenti vedono i contenuti immediatamente, senza dover attendere che il browser scarichi ed esegua il codice JavaScript per generare la pagina.

Inoltre, il SSR può migliorare significativamente l'indicizzazione delle pagine da parte dei motori di ricerca. Poiché i motori di ricerca preferiscono il markup HTML completo e ben strutturato, le pagine SSR sono più facili da indicizzare rispetto alle pagine generate lato client. Ciò significa che i contenuti delle MPA resi tramite SSR possono essere facilmente individuati e mostrati nei risultati di ricerca, migliorando la visibilità online dell'applicazione.

Oltre a migliorare la velocità di caricamento e l'indicizzazione, l'SSR può anche offrire una maggiore accessibilità. Gli utenti con connessioni di rete lente o dispositivi meno potenti beneficiano di tempi di caricamento più rapidi, mentre gli utenti con disabilità o tecnologie assistive possono navigare più facilmente attraverso pagine ben strutturate e caricate rapidamente.

In sintesi, il SSR rappresenta un'opzione preziosa per migliorare l'esperienza utente delle Multi-Page Applications. Integrando il Server-Side Rendering, le MPA possono offrire un'esperienza di navigazione più veloce, accessibile e indicizzabile, consentendo agli utenti di fruire dei contenuti in modo più immediato e soddisfacente. Questa integrazione rappresenta un passo significativo verso l'ottimizzazione e l'evoluzione continua delle web applications.

2.4 PWA

Le Progressive Web Apps (PWA) rappresentano un approccio innovativo nello sviluppo delle applicazioni web, offrendo un'esperienza utente simile a quella delle applicazioni native. La loro realizzazione si basa su una serie di tecnologie e tecniche che ne migliorano la velocità, l'affidabilità e l'interattività [21].

2.4.1 Realizzazione

Affinchè un'applicazione web sia classificata come PWA deve presentare i seguenti elementi

1. **Service Workers:** I Service Workers sono script JavaScript eseguiti in background dal browser. Consentono alle PWA di gestire eventi come le richieste di rete, migliorando così la gestione delle connessioni in condizioni di rete instabile o assente.

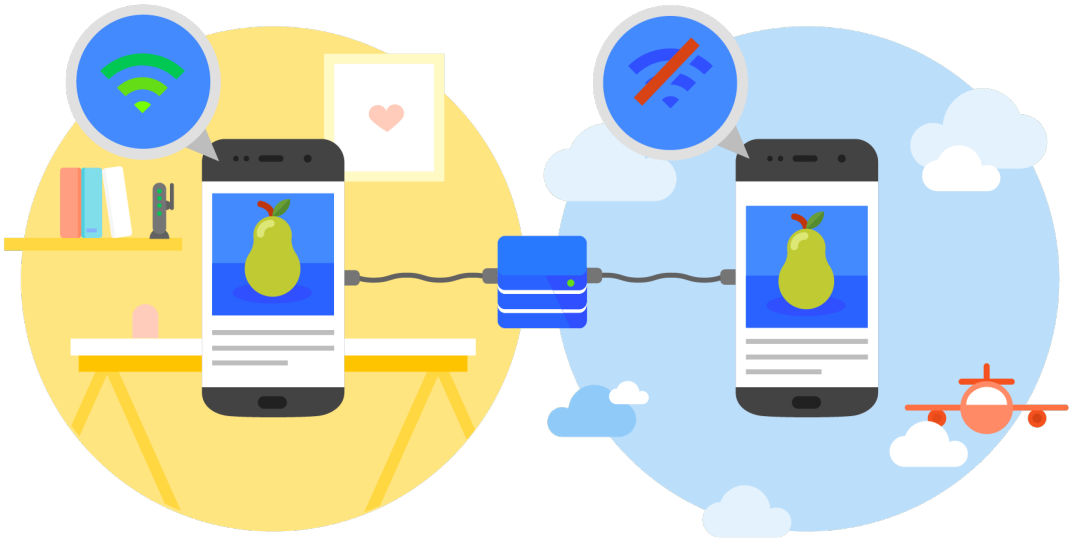

Figure 2.5: Funzionamento offline delle PWA
`https://shorturl.at/tCT46`

2. **Manifesti web:** Un file manifesto, chiamato Web App Manifest, definisce come l'applicazione deve apparire al browser. Contiene informazioni come il nome dell'app, l'icona, il colore di sfondo e altro ancora, permettendo alle PWA di sembrare e comportarsi come applicazioni native quando vengono installate sul dispositivo dell'utente.

3. **Cache e Offline Mode:** Le PWA utilizzano cache per memorizzare i file e le risorse necessarie per l'applicazione. Questo consente alle PWA di funzionare offline o in condizioni di rete instabile, migliorando l'accessibilità e l'affidabilità dell'applicazione.

4. **Notifiche push:** Le PWA possono inviare notifiche push agli utenti, anche quando l'applicazione non è aperta, migliorando l'interazione e mantenendo gli utenti informati sugli aggiornamenti e le attività dell'applicazione.

2.4.2 Vantaggi

Le PWA presentano diversi vantaggi e opportunità difficili da ignorare:

1. **Installazione semplificata:** Le PWA possono essere installate direttamente dal browser senza passare attraverso un app store. Gli utenti possono

aggiungere l'app alla schermata iniziale del loro dispositivo con un semplice clic, migliorando l'accessibilità.

2. **Funzionamento offline:** Grazie alla gestione intelligente della cache e dei Service Workers, le PWA possono funzionare offline o in modalità a connettività limitata, garantendo agli utenti l'accesso ai contenuti anche senza una connessione internet attiva.

3. **Velocità di caricamento:** Le PWA sono progettate per caricarsi velocemente, offrendo un'esperienza utente reattiva e fluida. Il contenuto è pronto per essere visualizzato quasi istantaneamente, migliorando la soddisfazione dell'utente.

4. **Sicurezza e aggiornamenti:** Poiché le PWA sono servite tramite HTTPS, offrono una connessione sicura agli utenti. Inoltre, gli aggiornamenti delle PWA sono automatici e trasparenti per gli utenti, garantendo che sempre utilizzino la versione più recente dell'app.

In conclusione, le Progressive Web Apps rappresentano un passo avanti significativo nell'evoluzione delle applicazioni web. Grazie alla loro realizzazione basata su Service Workers, cache intelligente e manifesti web, le PWA offrono un'esperienza utente avanzata, migliorando la velocità, l'affidabilità e l'interattività delle applicazioni web. Gli sviluppatori possono beneficiare di un processo di sviluppo semplificato e degli aggiornamenti automatici, rendendo le PWA un'opzione attraente per migliorare le interazioni digitali degli utenti.

Capitolo 3

React e UX

React, sviluppato da Facebook, è una libreria JavaScript open-source utilizzata per costruire interfacce utente dinamiche e reattive [22]. Introdotta nel 2013, React è diventata rapidamente una delle tecnologie più popolari nel mondo dello sviluppo web front-end. Ciò è dovuto alla sua flessibilità, efficienza e alla facilità con cui permette di creare componenti riutilizzabili e scalabili.

Nasce per risolvere alcune sfide specifiche nello sviluppo di grandi applicazioni web interattive. L'obiettivo principale era semplificare il processo di creazione di interfacce utente complesse mantenendo allo stesso tempo il codice ordinato e facilmente gestibile. Una delle caratteristiche distintive di React è l'approccio basato sui componenti, che permette agli sviluppatori di suddividere l'interfaccia utente in elementi indipendenti, facilitando così la gestione e la manutenzione del codice.

React è tra i framework più apprezzati e utilizzati dai programmatori a livello globale. La sua comunità attiva e numerosi strumenti di sviluppo di supporto hanno contribuito a creare un ecosistema robusto intorno a questa libreria. Molte aziende leader nel settore tecnologico, tra cui Facebook, Instagram, Airbnb e Netflix, hanno adottato React per le loro applicazioni, confermandone così la sua affidabilità e versatilità.

L'ascesa di React è anche evidente nella vasta adozione da parte della comunità open-source e nelle numerose risorse educative disponibili online. Questo fenomeno ha contribuito a creare una cultura di condivisione e apprendimento, incoraggiando sempre più sviluppatori a utilizzare React per i loro progetti.

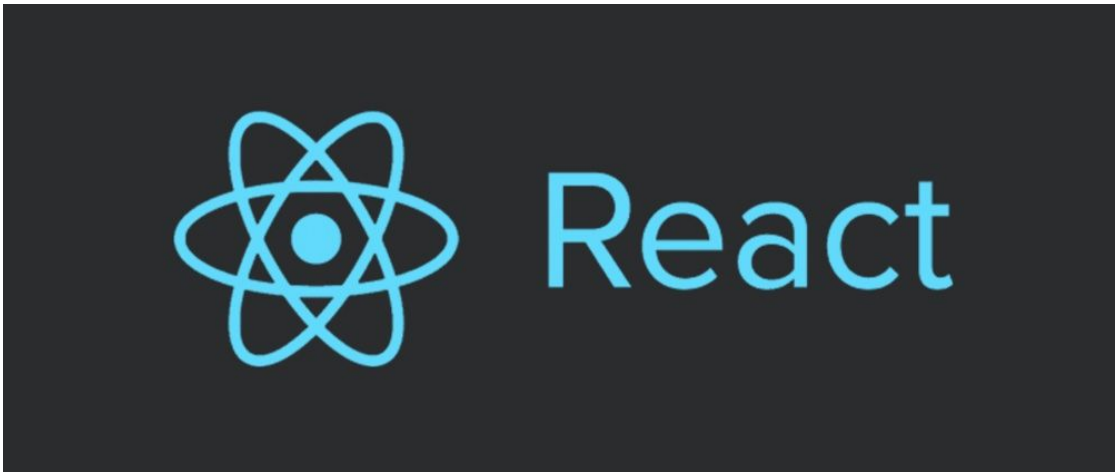

Figure 3.1: Logo di React
https://blog.fellyph.com.br/tutoriais/porque-react-js/

3.1 Principi chiave

React favorisce un approccio dichiarativo alla costruzione delle interfacce utente. Gli sviluppatori descrivono come dovrebbe apparire l'interfaccia utente in uno stato particolare e React si occupa del rendering efficiente e delle eventuali modifiche all'interfaccia utente.

Per fare ciò, React utilizza JSX, una sintassi di estensione di JavaScript che assomiglia a un linguaggio di markup, per definire la struttura degli elementi UI all'interno dei componenti. JSX rende il codice più leggibile e facile da scrivere, consentendo agli sviluppatori di combinare HTML e JavaScript in un'unica sintassi. Ad esempio, un componente React che rappresenta un pulsante potrebbe apparire così in JSX:

```
1 function Pulsante(props) {
2   return <button onClick={props.onClick}>Clicca Qui</button>;
3 }
```

In questo esempio, il codice JSX definisce un elemento `<button>` con un attributo `onClick` che riceve una funzione passata come proprietà (`props`) al componente.

Un altro concetto chiave di React che contribuisce in modo significativo alle prestazioni delle applicazioni è il Virtual DOM. Quando lo stato di un componente cambia, React non aggiorna immediatamente il DOM reale della pagina. Invece, genera una rappresentazione virtuale del DOM, nota come Virtual DOM. Questo Virtual DOM è un albero leggero che replica la struttura del DOM reale.

Quando avviene un aggiornamento dello stato o delle proprietà di un componente, React confronta il Virtual DOM precedente con quello nuovo, determina

le differenze (diffing) e calcola il modo più efficiente per aggiornare il DOM reale. Questo processo di confronto e aggiornamento del Virtual DOM è molto più veloce rispetto alla manipolazione diretta del DOM, contribuendo così a migliorare le prestazioni delle applicazioni React, specialmente in scenari in cui ci sono frequenti aggiornamenti dell'interfaccia utente.

3.1.1 Componenti

Una delle caratteristiche fondamentali di React è il suo approccio modulare all'architettura delle applicazioni, basato sul concetto di componenti.
I componenti in React sono blocchi autonomi e riutilizzabili di interfaccia utente. Ogni componente può avere il proprio stato interno e può essere composto da altri componenti, creando così una struttura gerarchica. Questo approccio favorisce la separazione delle responsabilità e semplifica la gestione del codice, permettendo agli sviluppatori di concentrarsi sullo sviluppo e il mantenimento di piccole parti dell'applicazione, anziché dell'intero monolite.

I componenti in React attraversano diverse fasi nel loro ciclo di vita. Queste fasi consentono agli sviluppatori di eseguire azioni specifiche in momenti specifici:

- **Mounting:** Il componente viene inserito nel DOM.
- **Updating:** Il componente viene re-rendirizzato a seguito di un cambiamento di stato o di proprietà.
- **Unmounting:** Il componente viene rimosso dal DOM.

3.1.2 Props

I props sono una delle caratteristiche fondamentali che consentono il passaggio di dati tra i componenti. I props, abbreviazione di "proprietà", sono argomenti passati da un componente padre a un componente figlio. Questo meccanismo di passaggio dei dati avviene in modo unidirezionale, da componenti genitori a componenti figli, garantendo una gestione chiara e prevedibile dello stato dell'applicazione.

Il passaggio dei dati tramite props segue un flusso unidirezionale. Un componente padre passa i dati al suo componente figlio come props. Il componente figlio può quindi accedere a questi dati e utilizzarli nel proprio render. Tuttavia, i componenti figli non possono modificare direttamente i loro props; sono considerati immutabili. Questo modello di passaggio dei dati contribuisce a mantenere l'applicazione prevedibile e semplifica il debug, poiché è possibile tracciare facilmente l'origine dei dati.

Un aspetto importante del passaggio dei dati tramite props è il re-rendering del componente quando i suoi props cambiano. Quando un componente riceve nuovi

props, React confronta i nuovi valori dei props con quelli precedenti. Se c'è una differenza, il componente viene re-renderizzato per riflettere gli ultimi dati passati attraverso i props.

Questo comportamento è cruciale per garantire che l'interfaccia utente sia sempre allineata con lo stato dell'applicazione. Se un componente figlio dipende da un prop che cambia, il re-rendering assicura che il componente si aggiorni per riflettere l'ultima versione dei dati. Questo aggiornamento dinamico è ciò che consente alle applicazioni React di rimanere reattive e di fornire un'esperienza utente fluida e dinamica.

3.1.3 Hooks

Gli Hooks in React sono una caratteristica introdotta dalla versione 16.8 del framework, che rivoluziona il modo in cui i componenti funzionali gestiscono lo stato, gli effetti collaterali e altre funzionalità di React. Prima dell'introduzione degli Hooks, la logica dello stato e del ciclo di vita era limitata ai componenti di classe. Gli Hooks consentono ora ai componenti funzionali di svolgere compiti complessi senza la necessità di scrivere classi, rendendo il codice più leggibile e riutilizzabile.

Gli Hooks sono funzioni JavaScript speciali che rispettano il ciclo di vita del componente in cui vengono invocate e che iniziano con il prefisso "use" (come `useState`, `useEffect`, `useContext`, ecc.). Gli Hooks possono essere utilizzati all'interno dei componenti funzionali per aggiungere funzionalità senza dover convertire il componente in una classe.

L'utilizzo degli Hooks ha portato notevoli vantaggi allo sviluppo React. Oltre a semplificare il codice, gli Hooks favoriscono la condivisione di logiche complesse tra componenti senza dover ricorrere all'ereditarietà o ai componenti di ordine superiore. Questa flessibilità e potenza hanno reso gli Hooks uno strumento fondamentale nel toolkit di ogni sviluppatore React, consentendo la creazione di componenti più chiari, modulari e facili da testare.

3.1.4 Stato

In React, lo stato di un componente rappresenta i dati che possono cambiare nel tempo durante l'esecuzione dell'applicazione.

```
1 function ExampleComponent() {
2   const [count, setCount] = useState(0);
3
4   // ...
5 }
```

In questo esempio, l'hook `useState(0)` inizializza uno stato con un valore iniziale di 0. `count` è il nome dello stato, mentre `setCount` è la funzione che viene

utilizzata per aggiornare lo stato. Chiamando `setCount` con un nuovo valore, React aggiornerà lo stato e re-renderizzerà il componente per riflettere le modifiche.

```
1 function ExampleComponent() {
2   const [count, setCount] = useState(0);
3
4   const increaseCount = () => {
5     setCount(count + 1);
6   };
7
8   // ...
9 }
```

Anche con l'hook `useState`, React si assicura che lo stato persista tra i re-render del componente funzionale. Ogni volta che chiami `setCount` con un nuovo valore, React mantiene il valore di count tra i vari re-render, garantendo che lo stato sia sempre aggiornato.

3.2 Context

Il Context in React è un meccanismo che consente di condividere dati, come lo stato dell'applicazione o funzioni, tra componenti in un albero di componenti senza dover passare esplicitamente le props attraverso ogni livello dell'albero.

Il Context è utilizzato per condividere dati che devono essere accessibili da molti componenti a vari livelli dell'albero dei componenti senza dover passare manualmente le props da cima a fondo. Questo è particolarmente utile per lo stato globale dell'applicazione, temi, autenticazione utente e altre informazioni che devono essere accessibili in tutta l'applicazione.

Il Context Provider è un componente React che fornisce i dati del Context a tutti i componenti discendenti. Il Provider accetta un valore come prop, che rappresenta il dato che desideriamo condividere. Tutti i componenti figli all'interno del Provider possono accedere a questo valore senza dover passare attraverso le props.

Nell'esempio seguente, `MyComponent` accede al valore fornito dal Context Provider ("Dato dal Context Provider") utilizzando l'hook `useContext`, che consente ai componenti di accedere al valore di un Context specifico. L'hook prende come argomento il Context stesso e restituisce il valore corrente del Context.

```
1  const MyContext = React.createContext();
2
3  const MyComponent = () => {
4    const contextValue = useContext(MyContext);
5
6    return (
7      <div>
8        {contextValue}
9      </div>
10   );
11 };
12
13 const App = () => {
14   return (
15     <MyContext.Provider value="Dato dal Context Provider">
16       <MyComponent />
17     </MyContext.Provider>
18   );
19 };
```

3.2.1 Gestione dello stato globale

In React, la gestione dello stato globale è essenziale per applicazioni complesse in cui più componenti devono condividere e reagire a uno stesso stato. Uno degli approcci è l'uso degli hook `useContext` e `useReducer`.

A differenza dell'hook `useState`, che è ideale per gestire lo stato singolo, useReducer è utile quando lo stato coinvolge oggetti complessi o richiede la gestione di un flusso di azioni.

L'hook `useReducer` accetta due argomenti principali: il riduttore (o funzione di riduzione) e lo stato iniziale. Il riduttore è una funzione che definisce come lo stato dovrebbe essere aggiornato in risposta a diverse azioni. Le azioni vengono inviate al riduttore e, in base al tipo di azione, vengono eseguite le operazioni appropriate sullo stato. Questo approccio rende useReducer particolarmente adatto per situazioni in cui è necessario gestire uno stato complesso che può essere modificato da diverse azioni.

Questa combinazione consente di sfruttare la potenza di `useReducer` per gestire le azioni e le transizioni di stato, mentre `useContext` semplifica la condivisione dello stato tra i componenti senza passare manualmente le props attraverso vari livelli della gerarchia dei componenti.

3.3 Routing

Il routing è un aspetto cruciale di molte applicazioni web, consentendo agli utenti di navigare tra diverse pagine e visualizzare contenuti specifici in base all'URL. React è pensato per la creazione di Single Page Applications, ma il routing può essere gestito utilizzando librerie di terze parti come React Router, che fornisce un'esperienza di navigazione fluida e reattiva.

Implementare un sistema di routing efficace in un'app React offre numerosi vantaggi. Innanzitutto, consente agli utenti di esplorare l'applicazione senza problemi, migliorando l'usabilità e l'accessibilità complessiva. Inoltre, il routing ben gestito consente di mantenere una struttura chiara e organizzata nel codice, facilitando la manutenzione e lo sviluppo continuo dell'applicazione nel tempo.

3.3.1 React Router

Per iniziare, è necessario configurare un componente `BrowserRouter`, fornito da React Router. Questo componente avvolge l'intera applicazione e fornisce un contesto necessario per la gestione del routing. All'interno del componente `BrowserRouter`, vengono definite le diverse route che corrispondono a specifiche URL.

```
1  const App = () => {
2    return (
3      <BrowserRouter>
4        <Routes>
5          <Route exact path="/" component={HomePage} />
6          <Route path="/about" component={AboutPage} />
7          <Route component={NotFoundPage} />
8        </Routes>
9      </BrowserRouter>
10   );
11 };
```

3.3.2 Elementi di routing

Per consentire agli utenti di spostarsi tra le pagine, React Router fornisce il componente `Link`. I link creati con `Link` fungono da collegamenti cliccabili che indirizzano l'utente alla pagina corrispondente. Questi link possono essere inseriti in bottoni, menu o qualsiasi altro elemento interattivo all'interno dell'applicazione, rendendo la navigazione un'esperienza intuitiva per gli utenti.

Una delle funzionalità potenti offerte da React Router è la gestione delle route dinamiche, dove i parametri nell'URL possono essere utilizzati per visualizzare dati specifici agli utenti. Le route dinamiche vengono definite attraverso il com-

ponente `Route` e i parametri dinamici nell'URL vengono preceduti dal simbolo `:` (es. `<Route path="/user/:id">...</Route>`)

Per gestire questa route dinamica estrarre i dati dall'URL, utilizziamo l'hook `useParams` fornito da React Router nel componente indicato nella route. Questo hook ci consente di accedere ai parametri definiti nella route dinamica, come l'ID dell'utente. Estratto il parametro `id`, possiamo personalizzare dinamicamente il contenuto della pagina per mostrare i dettagli specifici di quell'utente.

3.4 Effetti collaterali

In React, gli effetti collaterali si verificano quando le operazioni all'interno di un componente hanno effetti indesiderati su altre parti dell'applicazione, come la modifica di variabili globali, l'invio di richieste di rete o l'aggiornamento del DOM. Questi effetti collaterali possono causare comportamenti imprevisti e problemi di performance.

Gli effetti collaterali si verificano in React a causa della natura asincrona delle operazioni come richieste di rete, lettura/scrittura su file e operazioni di database. Poiché JavaScript è single-threaded, queste operazioni possono richiedere tempo per essere completate, e nel frattempo, il resto dell'applicazione continua a eseguire il suo codice.

Per gestire gli effetti collaterali in React, è possibile utilizzare l'hook useEffect, progettato per gestire attività che richiedono tempo, come richieste di API o sottoscrizioni a eventi esterni, in modo da evitare effetti collaterali indesiderati. Questo hook permette di eseguire il codice specificato dopo ogni rendering del componente, evitando così problemi di sincronizzazione e assicurando che le operazioni asincrone siano gestite correttamente.

3.4.1 Chiamate API

In React, le chiamate alle API sono esempi classici di effetti collaterali. Quando una componente esegue una chiamata API, l'applicazione continua a funzionare e a renderizzare altri componenti mentre aspetta la risposta. Questa asincronicità può comportare effetti collaterali, come la modifica dello stato o l'aggiornamento dell'interfaccia utente basato sui dati ricevuti. Per effettuare chiamate alle API in React, è possibile utilizzare la funzione nativa di JavaScript `fetch()`. Questa funzione consente di inviare richieste HTTP al server e gestire le risposte.

In React, la combinazione di `useState` e `useEffect` offre un modo efficiente per gestire chiamate asincrone alle API.

```
1  const ApiExample = () => {
2    const [data, setData] = useState(null);
3    const [loading, setLoading] = useState(true);
4
5    useEffect(() => {
6      // Funzione per effettuare la chiamata API
7      const fetchData = async () => {
8        try {
9          const response = await fetch('https://api.example.com/
               data');
10         const result = await response.json();
11         setData(result);
12         setLoading(false);
13       } catch (error) {
14         console.error('Errore durante la chiamata API:', error)
               ;
15         setLoading(false);
16       }
17     };
18
19     // Chiamata API al caricamento del componente
20     fetchData();
21   }, []); // L'array vuoto indica che l'effetto viene eseguito
        solo al montaggio del componente
22
23   return (
24     <div>{
25     loading
26         ? <p>Caricamento...</p>
27         : data
28             ? <p>Dati: {data}</p>
29             : <p>Nessun dato disponibile</p>
30     }
31     </div>
32   );
33 };
```

In questo esempio, `useState` viene utilizzato per definire due stati: `data` per memorizzare i dati ricevuti dall'API e `loading` per indicare se la chiamata API è ancora in corso. La funzione `fetchData` è una funzione asincrona che effettua la chiamata all'API e imposta lo stato di `data` e `loading` in base alla risposta ricevuta. L'hook `useEffect` viene utilizzato per invocare la funzione `fetchData` al montaggio del componente (grazie all'array vuoto come secondo argomento), consentendo così di recuperare i dati dall'API in modo asincrono.

Questo approccio permette di mantenere un'interfaccia reattiva durante il recupero dei dati, mostrando un messaggio di caricamento fino a quando i dati non sono pronti per essere visualizzati.

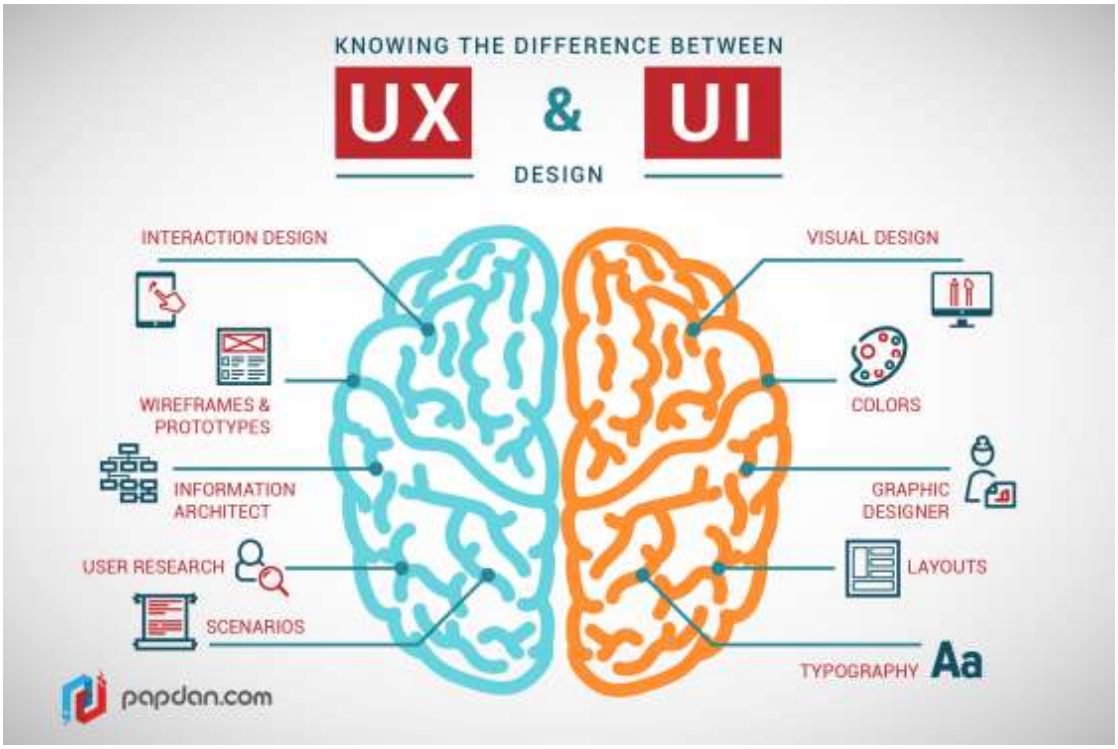

Figure 3.2: Differenze tra UX design e UI design
`http://freelancewebdesigner.it/ux-designer/`

3.5 UX Design

L'UX design, acronimo di User Experience design, si riferisce al processo di progettazione e miglioramento di prodotti, servizi e applicazioni in modo che siano intuitivi, facili da usare e soddisfacenti per gli utenti finali. Questo approccio si concentra sull'esperienza complessiva dell'utente durante l'interazione con un'interfaccia, incorporando aspetti emotivi, pratici e estetici per creare un'esperienza utente positiva e significativa [7].

I Fondamenti dell'UX Design:

- **Usabilità:** L'usabilità si riferisce alla facilità con cui gli utenti possono interagire con un'interfaccia. Un'interfaccia usabile è intuitiva, senza ostacoli, e consente agli utenti di compiere azioni senza difficoltà.

- **Accessibilità:** L'accessibilità garantisce che le persone con disabilità possano utilizzare un'applicazione con facilità. Questo può includere la progettazione per utenti con difficoltà di vista, udito o motorie, assicurando che l'interfaccia sia accessibile a tutti.

- **Attrattività visiva:** Un buon design non è solo funzionale ma anche esteticamente piacevole. L'attrattività visiva coinvolge l'uso del colore, della tipografia, delle immagini e di altri elementi visivi per creare un'esperienza visiva coerente e attraente.
- **Feedback e risposta:** Le interazioni dell'utente dovrebbero essere accompagnate da feedback visivi o sonori per confermare che l'azione è stata eseguita con successo, fornendo all'utente un senso di controllo e comprensione.
- **Consistenza:** La coerenza tra le diverse parti di un'applicazione è essenziale per garantire che gli utenti non si sentano confusi o disorientati durante l'utilizzo. Elementi come l'uso consistente di colori, icone e layout contribuiscono alla coerenza complessiva.

L'importanza dell'UX design nelle applicazioni web non può essere sottovalutata. Un'esperienza utente positiva porta a diversi vantaggi:

- **Riduzione dell'abbandono:** Un'interfaccia intuitiva e facile da usare riduce la frustrazione degli utenti, riducendo così il tasso di abbandono delle applicazioni.
- **Aumento della soddisfazione:** Un'esperienza utente positiva aumenta la soddisfazione degli utenti, incoraggiandoli a utilizzare l'applicazione in modo regolare e continuato.
- **Fidelizzazione degli utenti:** Gli utenti soddisfatti tendono a restare fedeli all'applicazione nel tempo, contribuendo così alla fidelizzazione degli utenti.
- **Feedback positivo:** Un'UX ben progettata porta a feedback positivi dagli utenti, generando reputazione e raccomandazioni positive per l'applicazione.
- **Risparmio di tempo e risorse:** Un'interfaccia utente ben progettata riduce la necessità di supporto e assistenza agli utenti, riducendo così i costi operativi.

3.6 User-Centered design

Il design centrato sull'utente è un approccio alla progettazione che mette gli utenti al centro del processo creativo. Questo metodo si basa sull'osservazione, la comprensione e l'analisi delle esigenze degli utenti per creare soluzioni intuitive e soddisfacenti. Il processo si compone di 3 fasi principali:

1. **Ricerca e analisi**

2. **Progettazione**

3. **Validazione**

3.6.1 Fase di ricerca e analisi

La fase di ricerca e analisi è fondamentale per identificare i requisiti dell'utente e per garantire che il progetto soddisfi le esigenze effettive degli utenti. Comprendere le aspettative e i comportamenti degli utenti aiuta a guidare le decisioni di design, adottando un approccio basato su dati concreto piuttosto che su supposizioni. È composta da analisi competitiva e analisi delle esigenze degli utenti.

L'analisi delle esigenze degli utenti coinvolge interviste, sondaggi, osservazioni e ricerche per comprendere le esigenze, le aspettative e i comportamenti degli utenti. Durante questa fase, è possibile raccogliere informazioni su:

- **Obiettivi degli utenti:** Quali sono gli obiettivi che gli utenti vogliono raggiungere utilizzando l'applicazione?
- **Problemi attuali:** Quali sono i problemi o le sfide che gli utenti stanno affrontando con le applicazioni attualmente disponibili?
- **Preferenze e aspettative:** Quali sono le preferenze degli utenti in termini di design, funzionalità e facilità d'uso?

L'analisi competitiva invece è un processo di ricerca finalizzato a comprendere come altre applicazioni simili nel mercato si comportano e soddisfano le esigenze degli utenti. Questo tipo di analisi aiuta a identificare punti di forza, debolezze, opportunità e minacce presenti nelle applicazioni concorrenti. Durante l'analisi competitiva, è possibile esaminare:

- **Caratteristiche e funzionalità:** Cosa offrono le applicazioni concorrenti in termini di funzionalità e caratteristiche uniche?
- **Design e usabilità:** Quali sono gli aspetti positivi del design e dell'usabilità delle applicazioni concorrenti?
- **Feedback degli utenti:** Quali sono i feedback degli utenti su queste applicazioni? Quali sono le recensioni e le valutazioni degli utenti?

3.6.2 Fase di progettazione

Dopo aver acquisito una comprensione approfondita delle esigenze degli utenti e aver analizzato il contesto competitivo, la fase di progettazione è cruciale per

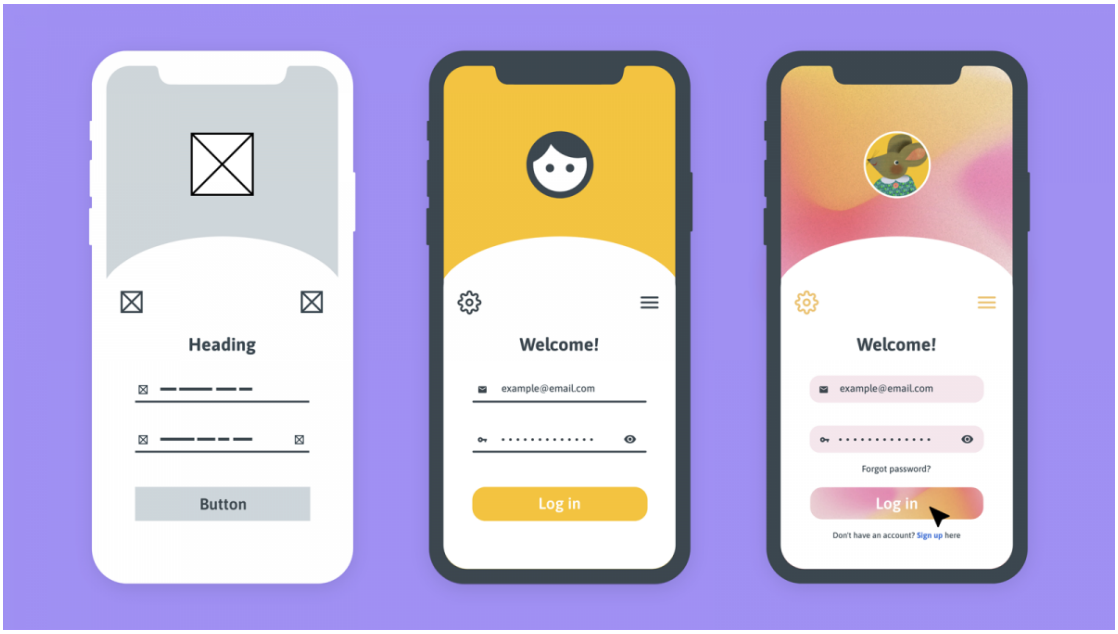

Figure 3.3: Esempio di wireframe, mockup e prototipo
`https://moqups.com/blog/wireframe-vs-mockup-vs-prototype/`

tradurre le informazioni raccolte in soluzioni concrete e visuali. Durante questa fase, gli sviluppatori utilizzano diverse tecniche di programmazione per creare rappresentazioni visive dell'interfaccia utente.

1. **Wireframe:** Un wireframe è una rappresentazione visuale di base dell'interfaccia utente, concentrata sugli elementi strutturali e sul layout. I wireframe sono schizzi schematici che mostrano la posizione degli elementi come i menu, i pulsanti e le caselle di testo, senza dettagli grafici come colori o immagini. L'obiettivo dei wireframe è delineare la struttura dell'applicazione in modo semplice e chiaro.

2. **Mockup:** Un mockup è una rappresentazione visuale più dettagliata dell'interfaccia utente, che include elementi grafici come colori, tipografie e immagini. I mockup forniscono una visione più precisa del design finale dell'applicazione. Creare mockup consente agli sviluppatori di avere una comprensione chiara di come apparirà l'applicazione una volta sviluppata, aiutando a prendere decisioni più informate sui dettagli del design.

3. **Prototipi interattivi:** I prototipi interattivi sono simulazioni interattive dell'applicazione che consentono agli utenti di esplorare e testare l'interfaccia utente in un ambiente simile a quello reale. Questi prototipi possono essere

creati utilizzando strumenti di progettazione e consentono agli sviluppatori di testare l'usabilità dell'applicazione, raccogliendo feedback dagli utenti in modo tempestivo.

L'utilizzo di queste tecniche è fondamentale perché aiuta gli sviluppatori a avere una chiara visione dell'aspetto e del comportamento dell'applicazione. Permette inoltre ai membri del team di comunicare in modo efficace e comprensibile, riducendo i malintesi e migliorando la collaborazione.

Infine consente di testare l'usabilità dell'applicazione prima della sua implementazione, identificando e risolvendo problemi di navigazione e flusso utente. In particolare offre agli utenti la possibilità di fornire feedback sulla progettazione, contribuendo a migliorare l'esperienza utente prima del lancio ufficiale.

3.6.3 Processo di validazione

Una volta che l'interfaccia utente è stata progettata, è essenziale convalidare le soluzioni proposte attraverso test approfonditi con gli utenti. Questo processo di validazione e test gioca un ruolo critico nel garantire un'esperienza utente ottimale per diversi motivi:

- **Rilevamento di problemi:** I test con gli utenti aiutano a identificare problemi reali che gli utenti potrebbero incontrare durante l'utilizzo dell'applicazione, che potrebbero non essere stati previsti durante la fase di progettazione.
- **Raccolta di feedback:** Le interviste e i feedback raccolti durante i test consentono di ottenere una comprensione diretta delle opinioni degli utenti, dei loro bisogni e delle loro preferenze, guidando così le future iterazioni del design.
- **Validazione delle soluzioni:** I test con gli utenti aiutano a convalidare se le soluzioni proposte durante la fase di progettazione sono realmente efficaci e intuitive, o se richiedono modifiche per adattarsi meglio alle aspettative degli utenti.

Tra i metodi di testing con utenti troviamo:

- **Test di usabilità:** I test di usabilità coinvolgono utenti reali che interagiscono con l'applicazione sotto la supervisione di un moderatore. Gli utenti vengono incaricati di completare compiti specifici mentre il moderatore osserva e raccoglie feedback. Questi test forniscono informazioni dettagliate sulla facilità d'uso e la navigabilità dell'applicazione.

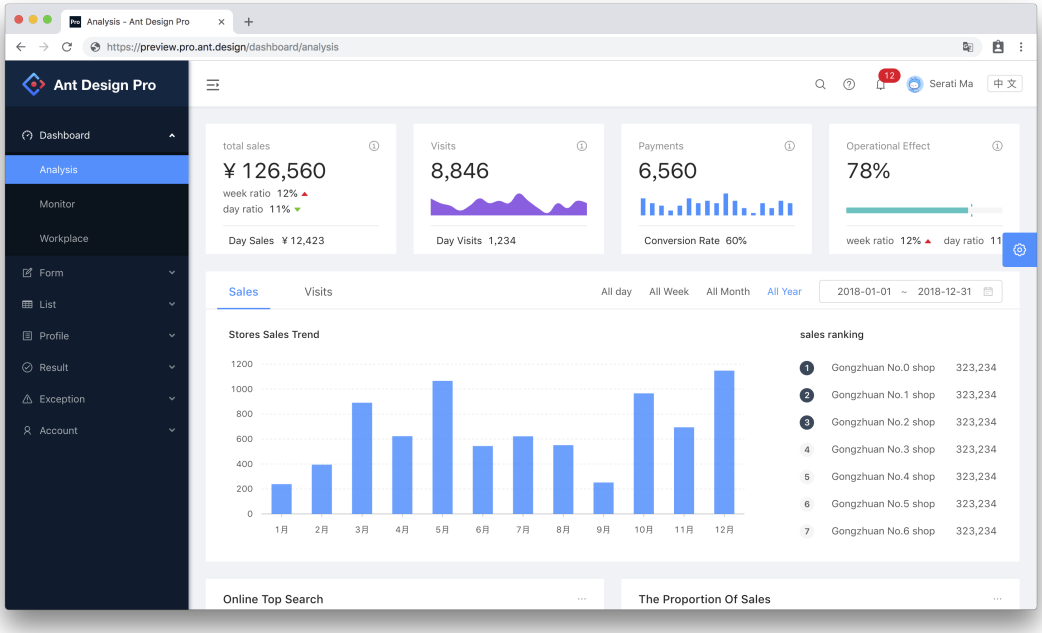

Figure 3.4: Esempio di pagina web realizzata con Ant Design
`https://github.com/ant-design/ant-design-pro`

- **Interviste con gli utenti:** Le interviste con gli utenti sono incontri diretti in cui gli sviluppatori possono fare domande specifiche agli utenti riguardo alle loro esperienze, preferenze e bisogni. Questi incontri sono preziosi per ottenere approfondimenti qualitativi che possono non emergere dai test di usabilità.

- **Feedback continuo:** La raccolta di feedback continuo tramite sondaggi online, questionari o strumenti di feedback integrati nell'applicazione può fornire informazioni sulle opinioni degli utenti anche dopo il lancio ufficiale dell'applicazione. Questo feedback continuo aiuta a mantenere l'interfaccia utente in linea con le aspettative degli utenti in evoluzione.

3.7 Ant Design

Ant Design è una libreria UI basata su React che si distingue per l'attenzione ai dettagli e per la sua capacità di offrire un'esperienza utente coerente ed elegante [12]. Guidata dai principi del design e sviluppata dalla comunità di React, Ant Design è diventata una delle librerie UI più apprezzate e utilizzate dagli sviluppatori

web di tutto il mondo. La libreria offre diverse comodità e semplificazioni:

1. **Componenti pre-stilizzati:** Ant Design offre una vasta gamma di componenti pre-stilizzati pronti per l'uso, tra cui bottoni, form, tabelle, modali e molto altro. Questi componenti sono progettati per essere esteticamente piacevoli e coerenti nel design, consentendo agli sviluppatori di risparmiare tempo prezioso durante lo sviluppo, senza dover costruire ciascun componente da zero.

2. **Semplicità e facilità d'Uso:** Una delle caratteristiche distintive di Ant Design è la sua semplicità e facilità d'uso. L'interfaccia utente intuitiva della libreria rende la creazione di applicazioni web complesse un processo fluido e accessibile anche per i programmatori meno esperti. La documentazione chiara e dettagliata di Ant Design facilita ulteriormente l'apprendimento e l'implementazione dei componenti.

3. **Customizzazione flessibile:** Nonostante la sua vasta gamma di componenti pre-costruiti, Ant Design permette una customizzazione estensiva. Gli sviluppatori possono facilmente adattare i componenti alle esigenze specifiche del progetto, cambiando colori, stili e layout. Questa flessibilità creativa consente di mantenere l'originalità del progetto senza compromettere la coerenza del design.

4. **Coerenza del design:** Ant Design è progettato per garantire una coerenza visiva e comportamentale in tutte le applicazioni che utilizzano la libreria. Questa coerenza è fondamentale per creare un'esperienza utente uniforme, indipendentemente dalla complessità dell'applicazione. La libreria fornisce linee guida rigorose e stili predefiniti per mantenere la coerenza del design in tutte le interazioni dell'utente.

Tra i vantaggi offerti da Ant Design troviamo sicuramente:

- **Risparmio di tempo:** Grazie ai componenti pre-stilizzati, Ant Design consente agli sviluppatori di risparmiare tempo prezioso durante lo sviluppo, riducendo la necessità di scrivere CSS personalizzato per ogni componente.

- **Coerenza e professionalità:** L'uso di Ant Design garantisce un aspetto coerente e professionale alle applicazioni, migliorando la percezione del prodotto da parte degli utenti.

- **Aggiornamenti continui:** Essendo supportato attivamente dalla comunità, Ant Design beneficia di aggiornamenti continui e miglioramenti, garantendo che gli sviluppatori possano usufruire delle ultime funzionalità e correzioni di bug.

Capitolo 4

Sfide e soluzioni

Nel capitolo dedicato alla fase di progettazione del front-end della cartella clinica, ci concentreremo sull'essenziale processo di creazione e ottimizzazione dell'interfaccia utente. Questa fase cruciale richiede un'attenzione particolare a tre pilastri fondamentali: la validazione accurata dei dati inseriti, l'autenticazione sicura degli utenti e un'efficace interazione con il server.

La validazione dei dati riveste un ruolo critico nell'assicurare che le informazioni inserite dagli utenti siano coerenti, corrette e affidabili. Questo non solo previene errori e incongruenze nella cartella clinica, ma garantisce anche la conformità alle normative e standard di sicurezza.

Parallelamente, l'autenticazione sicura è di vitale importanza per proteggere l'accesso alle informazioni sensibili dei pazienti, assicurando che solo gli utenti autorizzati possano accedere alla piattaforma.

L'interazione con il server, inoltre, richiede una progettazione intelligente e ottimizzata per garantire trasmissioni di dati veloci, affidabili e sicure. Questo capitolo esplorerà i dettagli di queste componenti chiave, fornendo linee guida e best practices per un front-end robusto e altamente funzionale, in grado di supportare l'ambiente complesso di una cartella clinica digitale.

4.1 Interazione con il server

Nel contesto dello sviluppo del nostro sistema per la gestione delle cartelle cliniche, l'interazione con il server rappresenta un aspetto cruciale che richiede una comprensione approfondita delle tecnologie coinvolte. Per garantire un'esperienza utente fluida e una gestione efficiente dei dati, è essenziale stabilire una comunicazione affidabile e sicura tra il front-end e il back-end del sistema.

Prima di esplorare gli aspetti tecnici, è cruciale comprendere le priorità fondamentali quando si tratta di interazione con il server. La disponibilità, l'integrità e

la sicurezza dei dati sono al centro di ogni operazione di comunicazione. Garantire che i dati vengano trasferiti in modo accurato e affidabile, rispettando la privacy e la sicurezza del paziente, è di primaria importanza. Inoltre, l'ottimizzazione delle richieste al server, sia in termini di tempi di risposta che di utilizzo delle risorse, contribuisce significativamente all'efficienza complessiva del sistema.

Per interagire con successo tramite RESTful APIs, è essenziale acquisire familiarità con diversi aspetti chiave. Questi includono la conoscenza degli endpoint API, che rappresentano gli URL specifici a cui inviare richieste per ottenere o inviare dati. Comprendere il formato dei dati scambiati, generalmente in formato JSON, è altrettanto importante, in quanto permette di elaborare e visualizzare le informazioni correttamente. Inoltre, è necessario comprendere il processo di autenticazione richiesto, come l'utilizzo di token JWT (JSON Web Tokens), per garantire che le richieste al server siano autorizzate e sicure.

4.1.1 Formato dei dati

Nel contesto della progettazione del sistema, la scelta del formato dei dati e del sistema di gestione del database riveste un ruolo cruciale. In questo contesto, abbiamo optato per MongoDB, un popolare database NoSQL, per lo storage dei dati della cartella clinica. Questa scelta è stata motivata dalla necessità di gestire dati complessi e variegati in modo flessibile ed efficiente.

I database NoSQL rappresentano una categoria di sistemi di gestione dei dati progettati per gestire modelli di dati diversificati e spesso complessi. A differenza dei database relazionali SQL, che seguono uno schema rigido basato su tabelle, i database NoSQL adottano un'approccio più flessibile. Possono gestire dati semistrutturati o non strutturati, consentendo una maggiore agilità nella gestione delle informazioni. Questo permette:

- **Flessibilità dello schema:** I database NoSQL permettono di modificare lo schema dei dati senza dover modificare l'intera struttura del database. Questo rende più semplice adattarsi ai cambiamenti nei requisiti del progetto,
- **Gestione di dati complessi:** Possono gestire dati complessi come documenti JSON, array o altri formati non tabulari. Questa flessibilità è particolarmente utile quando si trattano dati eterogenei, come quelli presenti nelle cartelle cliniche.
- **Scalabilità orizzontale:** I database NoSQL possono scalare orizzontalmente aggiungendo nuovi nodi al cluster, garantendo così prestazioni elevate anche con grandi volumi di dati.

Abbiamo scelto MongoDB come sistema di gestione del database per diversi motivi. Prima di tutto, la struttura flessibile dei dati in formato Binary JSON

Servizio	Funzionalità
`CCInfo-gateway`	Gestisce il carico delle richieste e le indirizza ai rispettivi servizi
`CCInfo-patients`	Fornisce le API per accedere e modificare dati sui pazienti
`CCInfo-schemas`	Fornisce le API per accedere agli oggetti JSON Schemas
`CCInfo-sso`	Fornisce le API per autenticare i medici
`CCInfo-oauth`	Servizio Keycloak utilizzato per la fase di autenticazione
`CCInfo-db`	Servizio MongoDB utilizzato come database centrale del sistema, viene replicato in 2 nodi aggiuntivi
`CCInfo-config`	Servizio per fornire le configurazioni globali a tutti gli altri servizi

Table 4.1: Architettura del server

(BSON) ci ha consentito di memorizzare dati complessi come documenti JSON nidificati. Questo formato è ideale per la rappresentazione di dati clinici, che possono includere informazioni eterogenee come testi, risultati di esami e visite con parametri diversi tra loro.

Inoltre, MongoDB supporta l'indicizzazione efficiente e le operazioni di query complesse, consentendo un recupero rapido dei dati. L'adozione di MongoDB ci ha anche permesso di implementare un modello di dati più elastico, in cui le strutture dei documenti possono adattarsi dinamicamente alle esigenze senza richiedere modifiche dello schema nel database.

4.1.2 Architettura a microservizi

Nel cuore dell'infrastruttura server della nostra applicazione per la gestione delle cartelle cliniche, adottiamo un'architettura a microservizi [6]. Questo approccio modulare ci consente di suddividere le diverse funzionalità del sistema in servizi distinti, ognuno dei quali è specializzato in un aspetto specifico del sistema sanitario. Questo comporta diversi vantaggi:

- **Scalabilità:** I microservizi possono essere scalati separatamente in base al carico di lavoro, consentendo una crescita flessibile in risposta alla domanda degli utenti.
- **Agilità e manutenibilità:** La modularità dei microservizi semplifica l'implementazione di nuove funzionalità e l'aggiornamento indipendente dei componenti, migliorando l'agilità e facilitando la manutenzione del sistema nel tempo.

- **Sicurezza e controllo:** L'utilizzo di Nginx come gateway ci consente di implementare misure di sicurezza centralizzate, come l'autenticazione, fornendo un maggiore controllo sull'accesso ai servizi.

Ogni microservizio nel nostro sistema è progettato per gestire un aspetto specifico del flusso di lavoro clinico. Ad esempio, abbiamo servizi dedicati alla gestione dei dati dei pazienti, alla gestione dei farmaci, nonché servizi ausiliari che si occupano del bilanciamento delle richieste e del processo di autenticazione. Questo approccio ci consente di mantenere ciascun servizio focalizzato su una singola responsabilità, facilitando la manutenibilità e consentendo l'aggiornamento indipendente di ciascun componente.

Per garantire la coerenza dell'ambiente di sviluppo, test e produzione, utilizziamo Docker per containerizzare ciascun microservizio. Docker ci permette di confezionare ogni servizio, comprese le sue dipendenze, all'interno di un container isolato. Questo approccio consente di eliminare le differenze tra gli ambienti di sviluppo e produzione, semplificando la distribuzione e garantendo una maggiore affidabilità e consistenza.

Per semplificare l'accesso ai nostri microservizi e unificarli sotto un unico endpoint, utilizziamo un servizio gateway basato su Nginx. Questo gateway funge da punto di ingresso unico per l'applicazione, consentendo ai client di accedere a tutti i servizi attraverso un singolo URL. Nginx si occupa del routing delle richieste verso i microservizi corrispondenti, permettendo una gestione centralizzata del traffico e semplificando l'implementazione di politiche di sicurezza come l'autenticazione e l'autorizzazione.

4.2 Autenticazione

Nel nostro sistema di gestione delle cartelle cliniche, la sicurezza delle informazioni è di primaria importanza. Per garantire un accesso controllato e sicuro alle risorse sensibili, abbiamo implementato un servizio di autenticazione tramite JSON Web Tokens (JWT) attraverso la piattaforma Keycloak.

4.2.1 JWT

I token JWT [19] sono stringhe di testo codificate che contengono informazioni sull'utente e altre informazioni relative all'autenticazione. Quando un utente si autentica nel sistema, il server genera un token JWT firmato digitalmente, che viene inviato al client. Il client include il token nelle successive richieste al server per autenticarsi. Il server può quindi decodificare il token, verificare la firma e accedere alle informazioni contenute per autorizzare le richieste dell'utente.

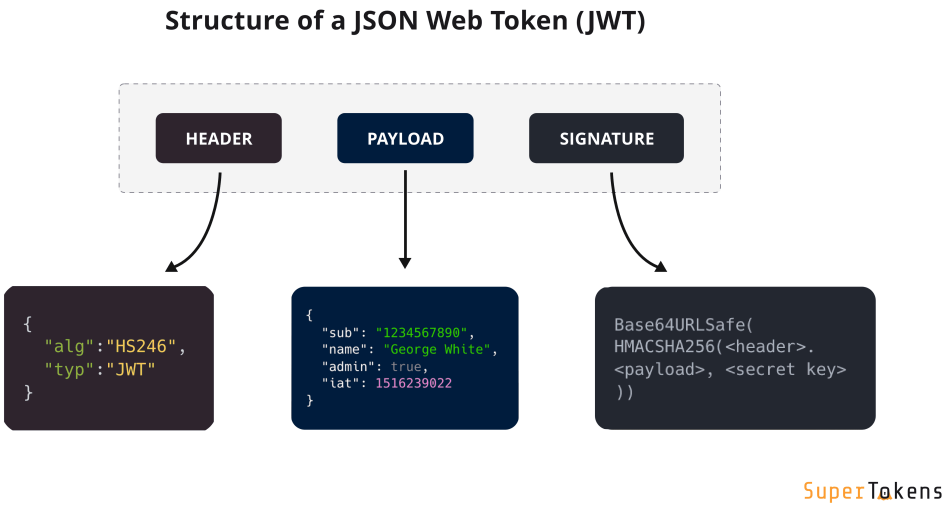

Figure 4.1: Struttura di un Json Web Token

I token JWT possono contenere varie informazioni, come l'ID dell'utente, i ruoli e le autorizzazioni, nonché eventuali dati personalizzati necessari per il sistema. Le informazioni sensibili, come le password, non vengono mai incluse nei token JWT per garantire la sicurezza delle comunicazioni.

Per mantenere attiva la sessione dell'utente senza richiedere l'autenticazione ripetuta, utilizziamo i cosiddetti "Refresh Token". Questi token più lunghi vengono utilizzati per richiedere nuovi token JWT quando quelli iniziali scadono. Questo processo permette agli utenti di rimanere autenticati nel sistema per un periodo prolungato senza dover riautenticarsi frequentemente.

4.2.2 OAuth

OAuth è un protocollo di autorizzazione aperto e standard del settore che consente l'autenticazione sicura e l'autorizzazione di applicazioni di terze parti senza condividere le credenziali sensibili dell'utente [8]. Con OAuth, un'applicazione può ottenere l'accesso autorizzato a determinate risorse da parte di un utente, senza che quest'ultimo debba condividere la propria password con l'applicazione stessa.

Keycloak [20] è una piattaforma open-source per la gestione delle identità e degli accessi, che integra OAuth e altri protocolli di sicurezza. Nel nostro sistema, utilizziamo Keycloak come servizio centralizzato per la gestione degli accessi. Tutti i dati sensibili riguardanti i medici, compresi i loro profili, le autorizzazioni e i dati personali, sono custoditi in modo sicuro all'interno di Keycloak.

Keycloak agisce come un servizio di autorizzazione, generando, validando ed effettuando il refresh dei token OAuth. Quando un utente si autentica nel sistema, Keycloak genera un token OAuth, che viene utilizzato per autenticare e autorizzare le richieste successive. Keycloak si occupa anche di verificare la validità dei token, garantendo che solo le richieste autorizzate siano elaborate dal server. Utilizzare Keycloak come sistema di autenticazione porta diversi benefici:

- **Centralizzazione e controllo:** L'utilizzo di Keycloak centralizza la gestione degli accessi e fornisce un controllo granulare sugli utenti e le loro autorizzazioni, consentendo una gestione sicura e scalabile delle identità.
- **Facilità di integrazione:** Keycloak offre API e integrazioni semplici, facilitando l'implementazione di OAuth nel nostro sistema senza richiedere sviluppo da zero.
- **Sicurezza dei dati:** I dati sensibili dei medici sono mantenuti in modo sicuro all'interno di Keycloak, riducendo il rischio di accessi non autorizzati e garantendo la privacy di medici e operatori sanitari.

4.3 Validazione dei dati

Nel contesto della gestione delle informazioni sanitarie, garantire la validità e la coerenza dei dati è essenziale per garantire la precisione e l'affidabilità del sistema. Dati incoerenti o inaccurati non solo compromettono la qualità delle informazioni cliniche, ma possono anche generare problemi significativi in futuro, influendo sulla diagnosi, la terapia e la sicurezza dei pazienti. Per affrontare questa sfida, abbiamo adottato un approccio che combina la flessibilità del formato JSON con la precisione delle regole di validazione: l'utilizzo di JSON Schemas.

Il formato JSON offre una struttura flessibile e dinamica per memorizzare dati complessi, come le informazioni cliniche. Tuttavia, per garantire la consistenza dei dati e l'applicazione di regole specifiche, è cruciale definire criteri di validazione chiari e coerenti. Questi criteri devono essere flessibili abbastanza da adattarsi alle varie informazioni sanitarie, ma anche precisi al punto da prevenire errori e incoerenze.

4.3.1 JSON Schema

Per soddisfare la nostra esigenza di una struttura flessibile che consentisse la memorizzazione dei dati in formato JSON, ma che permettesse anche una definizione agevole e precisa delle regole di validazione, abbiamo adottato JSON Schemas [18]. Questi schemi forniscono un modo standardizzato per definire le strutture

dati consentite e le regole di validazione associate. Ciò ci consente di garantire che i dati inseriti nel sistema rispettino i criteri specifici, fornendo dati coerenti e affidabili per l'analisi e il trattamento clinico. I JSON schemas sono a loro volta degli oggetti in formato JSON che presentano le seguenti caratteristiche:

1. **Struttura dichiarativa:** JSON Schema è dichiarativo, il che significa che definisce regole di validazione senza richiedere procedure o logica di controllo esplicita nel codice dell'applicazione. Questo semplifica la gestione delle regole di validazione.

2. **Flessibilità:** JSON Schema offre una vasta gamma di tipi di dati, consentendo la validazione di oggetti, array, stringhe, numeri e altro ancora. Questa flessibilità ci permette di adattare le regole di validazione alle diverse strutture dei dati nel nostro sistema.

3. **Validazione ricorsiva:** JSON Schema supporta la validazione ricorsiva, consentendo di definire regole che si applicano non solo all'oggetto principale, ma anche ai suoi campi annidati. Ciò assicura una validazione completa e accurata dei dati complessi.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "nome": {
      "type": "string",
      "minLength": 1
    },
    "cognome": {
      "type": "string",
      "minLength": 1
    },
    "eta": {
      "type": "integer",
      "minimum": 0
    },
    "email": {
      "type": "string",
      "format": "email"
    },
    "numero_di_telefono": {
      "type": "string",
      "pattern": "^\\d{10}$"
    }
  },
  "required": ["nome", "cognome", "email"]
}
```

In questo esempio di JSON Schema, definiamo uno schema che richiede che l'oggetto utente abbia almeno un nome, un cognome e un indirizzo email validi. L'età deve essere un numero intero positivo, e il numero di telefono deve seguire un formato specifico.

4.3.2 Dinamicità degli schemas

Nel nostro sistema, abbiamo implementato un servizio dedicato chiamato 'schemas', che svolge un ruolo cruciale nella validazione dei dati. Questo servizio fornisce ai client gli schemi necessari per validare le informazioni prima dell'inserimento nel sistema. Una delle caratteristiche distintive di questo servizio è la sua flessibilità. Gli schemi di validazione possono essere facilmente modificati e aggiornati direttamente dal database, consentendo una gestione dinamica e adattabile delle regole di validazione.

La struttura degli schemi è stata progettata in modo intelligente per permettere la generazione dinamica di elementi. Ad esempio, se un nuovo campo deve essere introdotto nel sistema, è possibile aggiungere dinamicamente un nuovo attributo allo schema corrispondente. Questo approccio consente di adattare la validazione dei dati alle esigenze specifiche senza richiedere modifiche complesse nel codice dell'applicazione. Inoltre, poiché gli schemi possono contenere regole di validazione per campi diversi, compresi quelli annidati o complessi, possiamo garantire una validazione accurata e completa per varie strutture di dati nel sistema.

L'approccio dinamico del servizio 'schemas' non solo semplifica la gestione delle regole di validazione, ma consente anche un adattamento rapido alle nuove esigenze del sistema sanitario. Questo livello di flessibilità ci consente di mantenere il nostro sistema all'avanguardia, consentendo l'aggiunta di nuovi campi e strutture dati senza interrompere il flusso operativo. In ultima analisi, questo servizio contribuisce a garantire che i dati nel nostro sistema siano sempre accurati, coerenti e conformi agli standard richiesti.

4.4 Percorsi diagnostici

I percorsi diagnostici rappresentano un elemento fondamentale nel panorama medico, delineando una sequenza strutturata di procedure, test e decisioni cliniche necessarie per giungere a una diagnosi accurata. Nel nostro sistema di gestione delle cartelle cliniche, questi percorsi giocano un ruolo cruciale nel guidare i medici attraverso il complesso labirinto delle decisioni diagnostiche, garantendo una diagnosi tempestiva, accurata e ben ponderata.

I percorsi diagnostici sono progettati con cura per ciascuna condizione medica, tenendo conto delle ultime evidenze scientifiche e delle migliori pratiche cliniche.

Figure 4.2: Esempio di percorso diagnostico
`http://www.vene-linfatici.it/percorso-diagnostico-terapeutico/`

Ogni passaggio nel percorso è attentamente studiato, consentendo ai medici di seguire una serie di azioni logiche e specifiche per valutare i sintomi del paziente, eseguire i test appropriati e formulare una diagnosi basata su dati concreti. Questa struttura dettagliata garantisce uniformità e coerenza nel processo decisionale, indirizzando i medici verso le migliori opzioni diagnostiche disponibili.

4.4.1 Flowchart

La richiesta di implementare una sezione dedicata ai percorsi diagnostici e alla loro rappresentazione tramite flowchart è emersa direttamente dalla comunità medica dell'ospedale. I medici hanno espresso l'importanza di avere una guida visuale chiara e dettagliata che delinei i passaggi da seguire per giungere a una diagnosi accurata. Questa esigenza non solo migliora la coerenza nelle procedure diagnostiche ma è essenziale per garantire la sicurezza dei pazienti e la precisione nei trattamenti.

Per rispondere a questa richiesta, abbiamo deciso di integrare la generazione dei percorsi diagnostici nello stesso flusso di lavoro con cui creiamo le pagine e i form relativi agli esami clinici. Questa integrazione è stata progettata per offrire ai medici un'esperienza senza soluzione di continuità all'interno del sistema. Quando un medico accede alla cartella clinica di un paziente per aggiungere o consultare i risultati di un esame, può immediatamente visualizzare il percorso diagnostico corrispondente attraverso una flowchart intuitiva e dettagliata.

Questo sistema porta a diversi benefici, tra cui:

- **Efficienza operativa:** Integrare la generazione dei percorsi diagnostici consente ai medici di accedere alle informazioni diagnostiche in un'unica piattaforma, eliminando la necessità di consultare documenti esterni o risorse aggiuntive.

- **Coerenza nella documentazione:** Assicurando che i percorsi diagnostici siano creati nello stesso ambiente in cui vengono inseriti i dati relativi agli esami, garantiamo la coerenza tra la teoria e la pratica, migliorando la qualità della documentazione medica.

- **Risposta tempestiva:** I medici possono adattare rapidamente i percorsi diagnostici in base alle nuove informazioni sui pazienti, garantendo una risposta tempestiva alle variazioni nel quadro clinico.

- **Facilità di utilizzo:** Gli strumenti di generazione delle flowchart sono progettati per essere intuitivi, consentendo ai medici di creare e modificare i percorsi diagnostici con facilità, anche senza competenze tecniche avanzate.

Capitolo 5

Implementazione

Durante i due incontri cruciali con il team medico, abbiamo intrapreso un processo collaborativo per identificare e comprendere le loro esigenze specifiche.

Nel primo incontro, abbiamo dedicato del tempo per presentare e discutere le loro richieste, con un focus particolare sulla necessità di integrare un flowchart per gestire i percorsi diagnostici. Questo primo incontro è stato fondamentale per ottenere una visione chiara delle funzionalità richieste e delle aspettative degli utenti finali.

Nel secondo incontro, abbiamo sollevato il livello di coinvolgimento presentando un prototipo che ha suscitato apprezzamento da parte dei medici. L'interazione diretta con il prototipo ha permesso loro di visualizzare concretamente le potenzialità dell'applicazione e ha stimolato la definizione di una lista più dettagliata di specifiche. Questo passaggio è stato cruciale per raffinare le funzionalità e adattarle alle effettive esigenze del team medico.

Per ottimizzare il processo e sfruttare al meglio il tempo mentre attendevamo il completamento del foglio di specifiche, abbiamo preso l'iniziativa di concentrarci sulla parte tecnica dell'applicazione. In particolare, abbiamo sviluppato un sistema dinamico che consentisse di rispondere alle richieste del team medico con modifiche minime al codice. Questa strategia mirava a massimizzare l'efficienza dello sviluppo, permettendo al contempo una flessibilità immediata per adattarsi alle esigenze emergenti, dimostrandosi un approccio proattivo e orientato alla soddisfazione delle richieste degli utenti finali.

5.1 Struttura

Nella scelta dell'architettura Multi-Page Application (MPA) per lo sviluppo dell'applicazione web dedicata alla cartella clinica, tre motivazioni fondamentali guidano questa decisione, superando l'approccio tradizionale orientato alla Search

Engine Optimization (SEO). In primo luogo, la necessità di generare dinamicamente contenuti in base ai parametri di navigazione degli utenti ha influenzato la preferenza per un'architettura che agevoli questa operazione. L'architettura MPA consente una gestione flessibile e dinamica dei contenuti, rendendo possibile l'adattamento delle pagine in risposta a variabili specifiche di navigazione, un requisito essenziale per un'applicazione di gestione sanitaria.

Il secondo motivo chiave è la gestione controllata dell'accesso definito dai ruoli alle singole pagine. Con un'architettura MPA, è più agevole implementare un sistema di autorizzazioni granulare, permettendo un controllo più preciso sull'accesso alle diverse sezioni dell'applicazione. Questa caratteristica risulta particolarmente rilevante in un contesto in cui la privacy delle informazioni mediche è prioritaria, garantendo che solo gli utenti autorizzati abbiano accesso a determinati dati e funzionalità.

Infine, la possibilità di implementare il Server Side Rendering (SSR) sottolinea il terzo vantaggio dell'architettura MPA in questa specifica situazione. Dato il numero significativo di client connessi e la varietà di dispositivi utilizzati, l'ottimizzazione attraverso SSR diventa una considerazione chiave. Renderizzare le pagine lato server consente di alleggerire il carico di lavoro del client e offre una soluzione più efficiente, particolarmente benefica per dispositivi meno potenti, come le macchine presenti nell'ambiente ospedaliero.

5.1.1 Routing

Il sistema di routing gioca un ruolo cruciale nell'indirizzare l'utente verso le diverse sezioni e pagine dell'interfaccia. Abbiamo adottato la libreria React Router [26] per gestire il routing dell'applicazione, scegliendola per la sua capacità di mantenere lo stato dei componenti durante la navigazione. Questo aspetto è risultato fondamentale poiché ci ha consentito di preservare i dati e lo stato dei componenti anche attraverso diversi redirect, garantendo una transizione fluida e senza perdita di informazioni durante la navigazione dell'utente.

Un aspetto rilevante del nostro sistema di routing è l'uso di route dinamiche, evidenziate nella tabella di routing (Tabella 5.1) attraverso l'utilizzo del carattere ":" negli URL. Le route dinamiche consentono la generazione dinamica di pagine basate su parametri specifici, fornendo un modo flessibile per gestire varie visualizzazioni.

Un'eccezione notevole riguarda le route `/pazienti/:id/percorsi` e `/pazienti/:id/:tabName`, in cui "percorsi" è definito prima. Questa scelta è dettata dalla natura di "percorsi", che rappresenta una pagina generata non dinamicamente, richiedendo un componente React dedicato.

Una caratteristica interessante del nostro sistema di routing è la possibilità di definire elementi comuni per molteplici pagine, rendendo l'applicazione più coer-

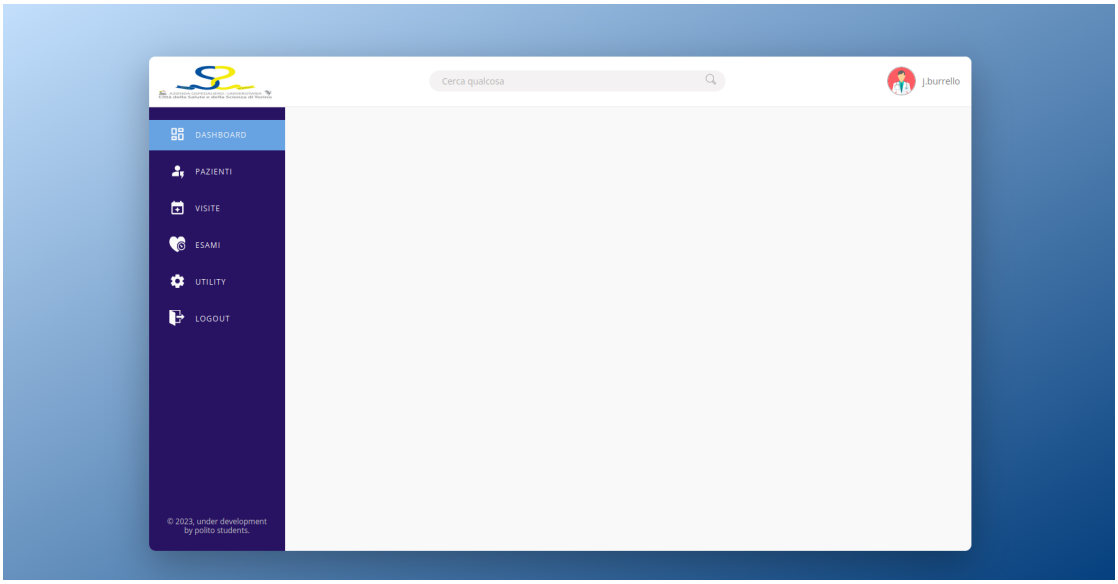

Figure 5.1: Menu di navigazione dell'applicazione

ente e user-friendly. Questo risultato è ottenuto attraverso l'uso del componente `Outlet` di React Router. Ad esempio, tutte le pagine, ad eccezione di login e logout, presentano lo stesso menu (Figura 5.1), migliorando la coerenza visiva e semplificando la navigazione.

Un aspetto potente del nostro sistema di routing è la capacità di definire script personalizzati per la gestione delle pagine. Un esempio è la route `/`, che esegue uno script di reindirizzamento in base allo stato dell'utente. Se l'utente è autenticato, verrà reindirizzato a una pagina specifica; altrimenti, sarà indirizzato verso una diversa pagina di accesso.

5.2 Pagine

Nella progettazione del front-end dell'applicazione, un focus primario è stato posto sull'offrire agli utenti un'esperienza intuitiva e altamente funzionale. Le pagine implementate riflettono questo impegno, presentando non solo un design accattivante ma anche una struttura che si adatta dinamicamente alle esigenze degli utenti.

Gran parte di queste pagine presenta elementi generati dinamicamente, un aspetto che sarà ampiamente esplorato nelle sezioni successive. Questa caratteristica non solo consente una maggiore flessibilità nell'adattarsi a varie situazioni, ma contribuisce anche a una navigazione più fluida e personalizzata.

Route	**Contenuto**
`/`	Se autenticato redirect a `/dashboard`, altrimenti redirect a `/login`
`/login`	Schermata di inserimento username e password
`/logout`	Callback per effettuare il processo di logout
`/dashboard`	Pagina momentaneamente vuota
`/pazienti`	Tabella con elenco di tutti i pazienti a carico del medico
`/pazienti/:id`	Elemento dinamico che mostra una barra laterale cui si accede ai dati del paziente
`/pazienti/:id/percorsi`	Tab non dinamico che mostra i percorsi diagnostici e i rispettivi flowchart
`/pazienti/:id/:tabName`	Elemento dinamico che mostra una sotto-sezione dei dati del paziente
`/visite`	Tabella con elenco di tutte le visite effettuate dai pazienti del medico
`/visite/nuovo`	Form per inserimento dei dati relativi ad una visita effettuata
`/visite/per/:id`	Form dinamico per registrare i dati di una nuova visita per un paziente
`/esami`	Tabella con elenco di tutti gli esami effettuate dai pazienti del medico
`/esami/nuovo`	Form per inserimento dei dati relativi ad un esame effettuato
`/esami/per/:id`	Form dinamico per registrare i dati di un nuovo esame per un paziente

Table 5.1: Routing dell'applicazione

5.2.1 Login e Layout generale

La pagina di login rappresenta il punto di ingresso principale nell'applicazione per gli utenti non autenticati. Presenta un'interfaccia semplice con un form di accesso, in cui gli utenti possono inserire le proprie credenziali: username e password.

Una volta completato il processo di autenticazione con successo, l'utente viene reindirizzato alla pagina `/dashboard`. Qui prende forma il layout generale dell'applicazione, che si sviluppa attorno a un design intuitivo e funzionale sostenuto dal componente `Outlet` di React Router.

Il layout principale è strutturato con una barra laterale posizionata a sinistra, che presenta le diverse sezioni dell'applicazione come dashboard, pazienti, visite, ed esami. Al momento, le pagine dedicate ai pazienti, alle visite e agli esami sono state implementate, garantendo una navigazione coerente e accessibile.

Nella parte superiore a destra, è visibile l'username dell'utente autenticato, fornendo un riferimento immediato per l'identificazione. Nel centro del layout è collocata una barra di ricerca globale, attualmente configurata per la ricerca di pazienti tramite il loro nome. Tuttavia, questa funzionalità può essere facilmente ampliata per includere altre opzioni di ricerca, come farmaci, terapie o medici, per adattarsi alle diverse esigenze degli utenti.

5.2.2 Pazienti

La pagina dedicata ai pazienti rappresenta il fulcro dell'applicazione, progettata per essere il punto focale a cui ogni medico accede regolarmente. Durante il suo sviluppo, è stata posta particolare attenzione a ottimizzare l'esperienza utente e garantire un accesso efficiente alle informazioni.

Al momento del caricamento della pagina pazienti, viene effettuata una query al server per recuperare i dati relativi ai pazienti. Questi dati sono successivamente memorizzati nella cache dell'applicazione per garantire un rapido accesso alle informazioni in future interazioni.

Il componente principale di questa pagina è una tabella dinamica che presenta le informazioni essenziali di ciascun paziente. Il tasto "Apri Cartella" permette l'accesso immediato ai dati completi di un singolo paziente.

Una volta aperta la cartella, compare una barra di navigazione sulla destra, offrendo accesso alle diverse sezioni relative ai dati del paziente. La sezione "Anagrafe" viene automaticamente aperta, mostrando i dettagli anagrafici del paziente, mentre le sezioni "Dati Clinici" presentano in modo strutturato terapie, esami e visite.

Particolare attenzione è stata dedicata ai "Percorsi Diagnostici", un aspetto approfondito in seguito, che unifica il processo diagnostico tra i medici attraverso una rappresentazione chiara e uniformata delle procedure diagnostiche.

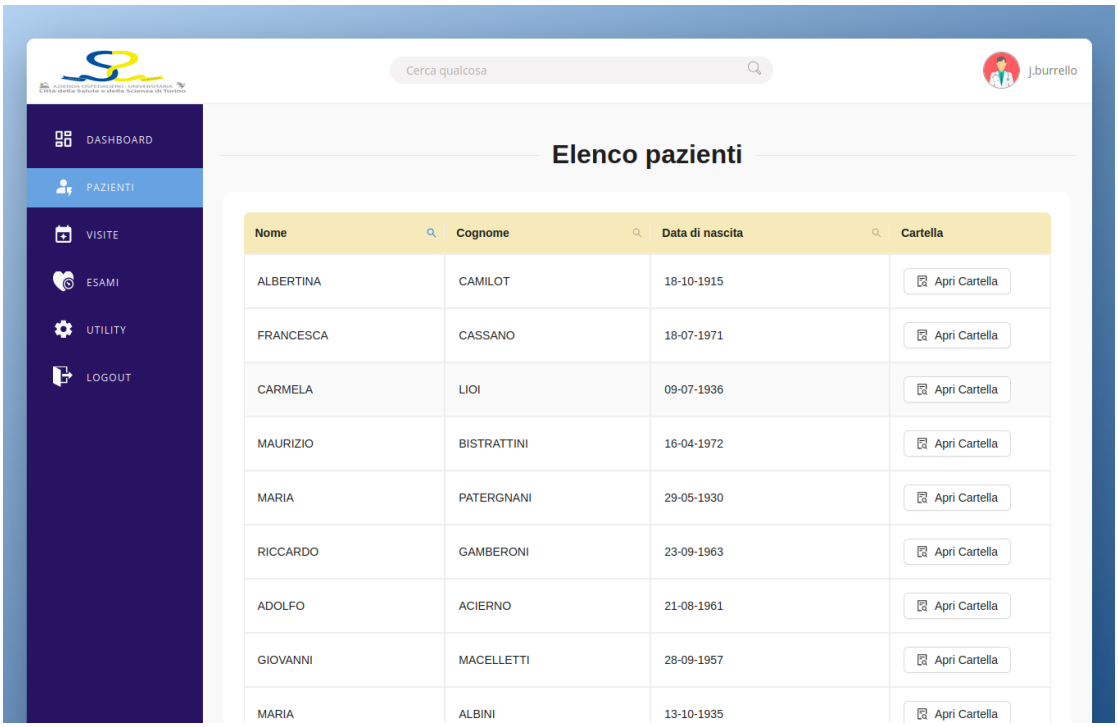

Figure 5.2: Pagina dei pazienti

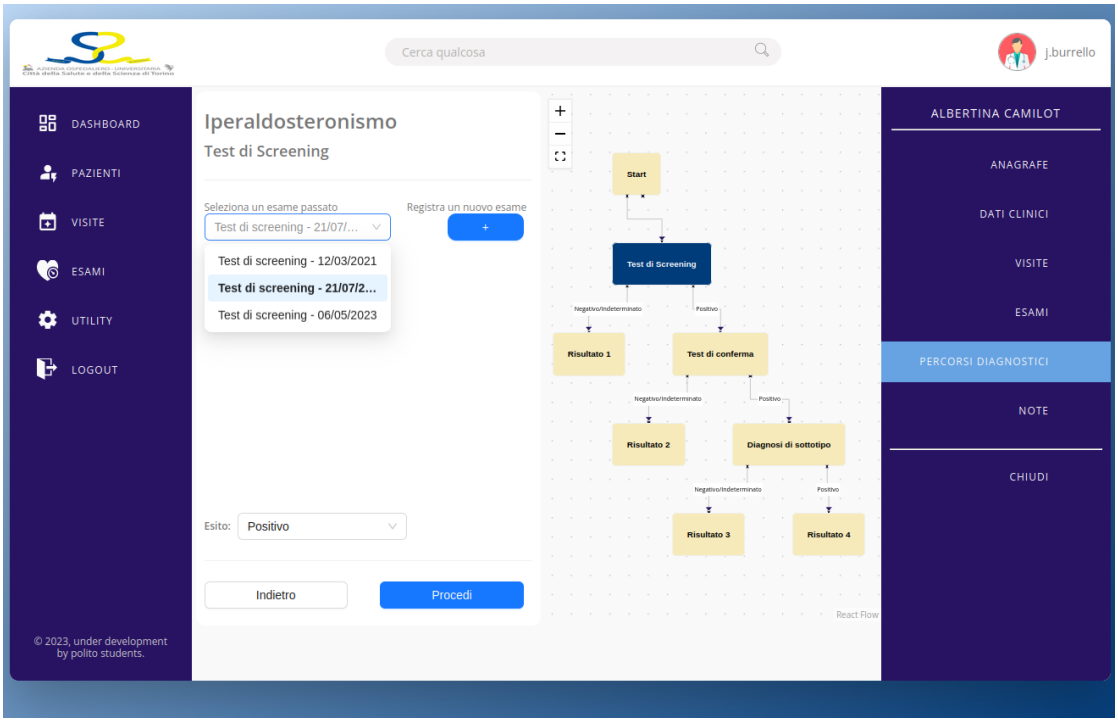

Figure 5.3: Selezione di un esame per un nodo del percorso diagnostico

Infine, la sezione "Chiudi" permette di tornare alla tabella principale dei pazienti, offrendo una navigazione fluida tra le varie sezioni senza perdere il contesto generale delle informazioni mediche disponibili.

5.2.3 Percorsi

Nella sezione dedicata ai percorsi diagnostici, agli utenti viene presentata una lista esaustiva dei vari percorsi che possono essere intrapresi, evidenziando lo stato di ciascuno (da iniziare, in corso o terminato) insieme al relativo esito. Dall'altra parte della schermata, è presente un elenco dei percorsi già completati, mostrando anche il loro esito.

L'interazione con questa sezione differisce da altre pagine dell'applicazione, in quanto la visualizzazione del contenuto è gestita senza reindirizzamenti. Se un percorso da iniziare o in corso viene selezionato, la vista cambia dinamicamente.

Una volta selezionato un percorso, sulla destra viene generato e visualizzato il flowchart corrispondente, evidenziando chiaramente lo step attuale nel percorso diagnostico. Nello stesso momento, sulla sinistra si apre un nuovo menu che indica il tipo di esame necessario per procedere ulteriormente.

A questo punto, il medico può scegliere di utilizzare un esame precedentemente

effettuato, selezionandolo dal menu a tendina, oppure di registrarne uno nuovo tramite l'apposito pulsante. Una volta scelto l'esame, al medico viene richiesto di assegnare un esito, solitamente tra positivo, negativo o indeterminato, completando così il percorso diagnostico corrente.

Una volta effettuata la scelta, il tasto "avanti" diventa interagibile e consente al medico di procedere con un semplice clic. In seguito, l'arco selezionato viene attraversato e si passa al nodo successivo del percorso diagnostico.

Questo sistema consente ai medici di navigare avanti e indietro nel grafo del percorso diagnostico, modificando gli esiti degli esami in caso di necessità. Tale flessibilità permette di adattare il percorso, consentendo a ciascun medico di visualizzare il prossimo esame o la successiva terapia necessaria per completare il percorso diagnostico e raggiungere una diagnosi conclusiva.

5.2.4 Esami e Visite

La sezione esami e visite si basano sulla stessa struttura e fanno uso degli stessi componenti, evidenziando la robustezza e la flessibilità della dinamicità dell'intera piattaforma. Consideriamo ad esempio la sezione esami, la quale presenta una tabella contenente l'elenco di tutti gli esami prescritti e registrati dal medico per un particolare paziente.

Ogni voce all'interno di questa tabella offre un tasto interattivo che, al click, apre un modal contenente i dettagli specifici dell'esame, senza obbligare il medico a navigare verso altre pagine. Questa modalità di presentazione dei dettagli consente un'esperienza utente più fluida e diretta.

Inoltre, in alto a destra, è disponibile un pulsante per registrare un nuovo esame. Cliccando su questo tasto, si apre un modal con una selezione delle tipologie d'esame disponibili. Una volta scelta la tipologia desiderata, il medico viene indirizzato verso una nuova pagina dove viene presentato un form con i campi dati necessari per inserire le informazioni relative all'esame prescelto. Tale approccio mira a semplificare il processo di inserimento dei dati, garantendo un'esperienza efficiente e intuitiva per il medico.

Qualora il medico necessiti di controllare tutti gli esami effettuati su tutti i suoi pazienti, può farlo attraverso la pagina "Esami", appositamente strutturata in maniera simile alla pagina dedicata ai singoli pazienti. Questa sezione offre un'ampia panoramica di tutti gli esami prescritti e registrati per ciascun paziente a cui il medico è associato.

In aggiunta, il medico ha la possibilità di registrare un nuovo esame attraverso lo stesso procedimento precedentemente descritto. Tuttavia, in questo caso, deve specificare per quale paziente viene registrato l'esame, consentendo così di associare l'informazione all'anagrafe corretta. Questa funzionalità offre al medico una

visione completa delle proprie attività e dei dati clinici dei pazienti, garantendo un controllo e una gestione efficace delle informazioni mediche.

5.3 Autenticazione

Per garantire un solido sistema di autenticazione per l'applicazione della cartella clinica, abbiamo fatto una scelta strategica affidandoci al servizio esterno Keycloak. Questa decisione è stata guidata dalla sua comprovata affidabilità e dalle robuste funzionalità di gestione delle identità e degli accessi. Tuttavia, riconoscendo l'importanza della flessibilità e la necessità di adattarsi a eventuali requisiti specifici dell'ospedale, abbiamo strutturato l'architettura per consentire la sostituzione di Keycloak con qualsiasi sistema raccomandato dal responsabile della sicurezza informatica dell'ospedale. Questa progettazione modulare ci ha fornito la possibilità di rispondere in modo agile a eventuali cambiamenti o aggiornamenti dei protocolli di sicurezza.

Grazie all'utilizzo di JSON Web Tokens (JWT), siamo in grado di garantire che tutte le richieste verso il server siano autenticate in modo sicuro. Questi token contengono informazioni cruciali che consentono di verificare l'autenticità dell'utente. La struttura dei JWT utilizzati è stata attentamente progettata per fornire una solida sicurezza e allo stesso tempo contenere le informazioni necessarie per l'autenticazione e l'autorizzazione. All'interno di ciascun token, sono inclusi dati specifici come l'identificativo dell'utente, i ruoli assegnati e altri attributi necessari per consentire un accesso controllato e sicuro alle risorse dell'applicazione. Questo approccio garantisce un'autenticazione robusta e la gestione accurata degli accessi, contribuendo alla creazione di un ambiente sicuro e protetto per la gestione delle informazioni mediche sensibili.

5.3.1 AuthContext

Tutte le informazioni e i metodi relativi al processo di autenticazione sono centralizzati nel context denominato `AuthContext`. Questo contesto svolge un ruolo chiave nell'assicurare che le funzionalità di login e logout siano accessibili in modo coerente in tutta l'applicazione.

Un elemento distintivo del `AuthContext` è l'uso di una funzione reducer, un concetto centrale in React. Il reducer definisce le operazioni da eseguire durante le fasi di login e logout, consentendo una gestione chiara e controllata dello stato di autenticazione.

Il cuore di questo contesto risiede nel componente root dell'applicazione, dove viene definito uno stato globale di autenticazione denominato `state`. Questo stato contiene informazioni cruciali come l'username dell'utente e i ruoli assegnati. In-

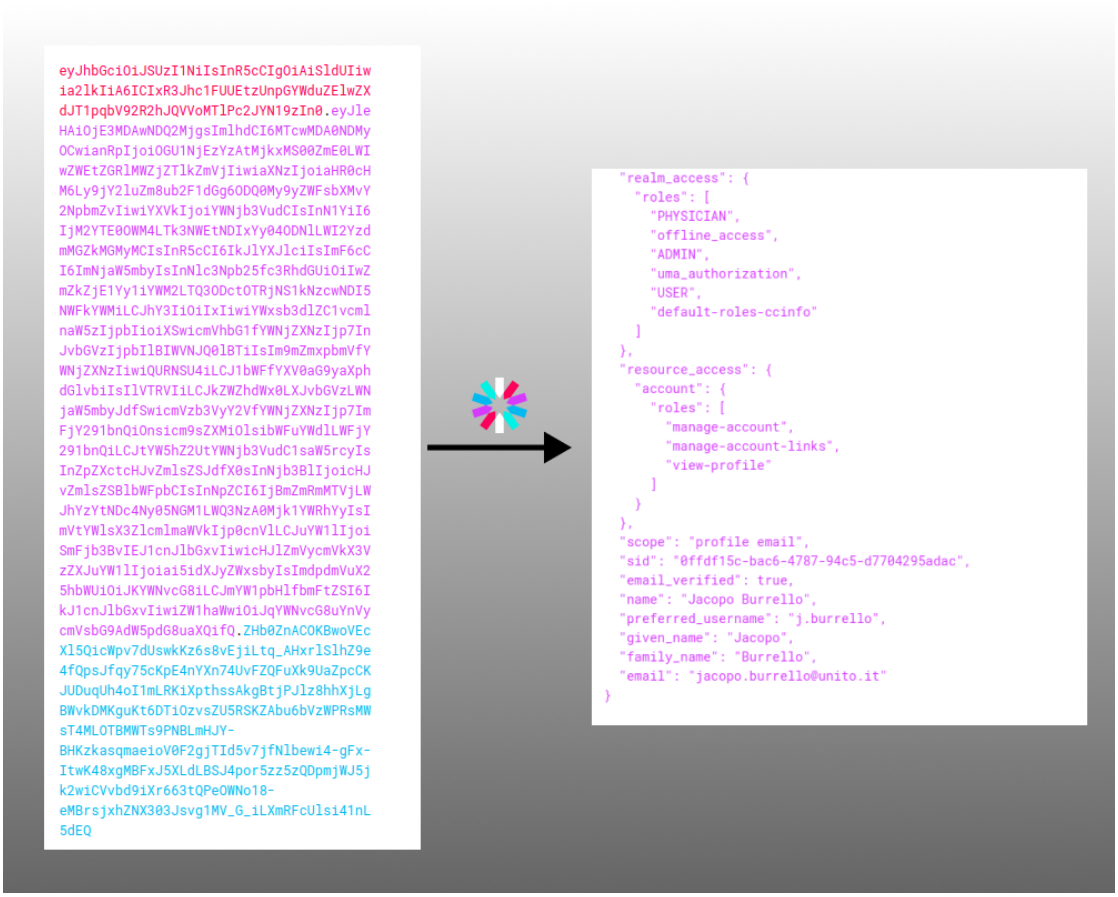

Figure 5.4: Decodifica del payload di un JWT

oltre, viene implementata la funzione `dispatch` attraverso l'hook `useReducer()`, che consente di effettuare modifiche controllate allo stato.

Lo stato e la funzione di `dispatch` vengono quindi inseriti in un oggetto di tipo `AuthContext`, creando un contesto che offre accesso a queste informazioni in qualsiasi parte dell'applicazione. L'`AuthContext.Provider`, che incapsula tutti gli altri componenti, è definito per garantire la disponibilità coerente del contesto a livello globale.

Per semplificare ulteriormente l'accesso a state e dispatch, è stato creato un nuovo hook denominato `useAuth()`. Questo hook agisce come un alias per `useContext(AuthContext)`, agevolando l'accesso ai parametri critici in modo chiaro e conciso in qualsiasi parte dell'applicazione.

5.3.2 Processo di autenticazione

Il sistema di autenticazione implementato nell'applicazione segue un flusso ben definito per garantire un accesso sicuro e controllato agli utenti autorizzati. Di seguito sono descritti i passaggi chiave di questo processo:

1. **Richiesta di Autenticazione:** Per avviare il processo di autenticazione, il client invia una richiesta HTTP di tipo POST all'endpoint `/api/sso/login` del server. Il corpo di questa richiesta contiene un oggetto JSON con le credenziali dell'utente, ovvero "username" e "password".

2. **Reindirizzamento al Servizio OAuth:** La richiesta di autenticazione viene instradata attraverso il gateway, che reindirizza la richiesta al servizio OAuth implementato tramite Keycloak.

3. **Autenticazione con Keycloak:** Keycloak verifica le credenziali fornite e, se corrette, risponde con un oggetto contenente vari attributi, tra cui gli importanti `access_token` e `refresh_token`. Entrambi sono formati come JSON Web Token (JWT).

4. **Salvataggio dei Token:** Il client salva in modo sicuro i token ricevuti (access e refresh) nel local storage del browser. Questi token saranno utilizzati per le successive richieste al server e per mantenere l'autenticazione durante la sessione dell'utente.

5. **Decodifica dell'Access Token:** L'access token viene decodificato per ottenere le informazioni cruciali, come l'username e i ruoli assegnati all'utente. Questo processo consente al sistema di identificare l'utente autenticato e definirne i privilegi.

6. **Aggiornamento dello Stato Globale:** Utilizzando l'hook `useAuth()`, l'applicazione ottiene il metodo dispatch, che consente di inviare azioni per aggiornare lo stato globale. Un oggetto JSON con le informazioni di login, contenente {`"type": "LOGIN", "username": username, "roles": roles`}, viene passato al metodo `dispatch`. Questo processo aggiorna lo stato globale dell'applicazione con le informazioni dell'utente autenticato.

5.4 Middleware

La necessità di definire un middleware per il frontend si è rivelata essenziale per ottimizzare il processo di gestione delle richieste e delle risposte dal server nell'applicazione della cartella clinica. Questo middleware funge da intermediario tra il frontend e il backend, facilitando la comunicazione e automatizzando diverse operazioni, rendendo così il flusso di dati più efficiente e gestibile.

Un componente cruciale di questo middleware è l'implementazione di una cache, progettata per evitare sovraccarichi al server attraverso la memorizzazione temporanea delle risposte alle richieste più frequenti. Questa cache si è rivelata fondamentale per migliorare le prestazioni complessive dell'applicazione, riducendo il tempo di risposta e ottimizzando l'uso delle risorse.

L'implementazione di queste caratteristiche è stata resa possibile grazie alle librerie Axios e React-Query. Axios è stata utilizzata per semplificare la gestione delle richieste HTTP, fornendo un'interfaccia pulita e intuitiva per effettuare chiamate al server.

D'altra parte, React-Query ha fornito un framework potente per la gestione dello stato globale e delle query ai dati. La sua capacità di gestire automaticamente la cache delle query, insieme a una serie di funzionalità avanzate per gestire lo stato dei dati, ha semplificato notevolmente l'implementazione del middleware e ha migliorato l'esperienza complessiva degli utenti.

5.4.1 Axios

La libreria Axios [13] rappresenta un potente strumento nel contesto dello sviluppo front-end, offrendo una serie di vantaggi significativi e una flessibilità che la rende una scelta popolare per la gestione delle richieste HTTP in applicazioni React.

Axios semplifica notevolmente la gestione delle richieste e delle risposte HTTP, fornendo un'interfaccia chiara e intuitiva basata su Promises. Questo facilita la scrittura di codice asincrono pulito e comprensibile, fondamentale nelle moderne applicazioni web.

Tra i vantaggi chiave di Axios, spicca la possibilità di intercettare richieste e risposte. Questo meccanismo consente di intervenire in fasi specifiche della co-

municazione HTTP, fornendo un controllo dettagliato e consentendo, ad esempio, l'aggiunta di header personalizzati o la gestione degli errori in modo centralizzato.

Un aspetto cruciale di Axios è la sua configurabilità. L'utente può personalizzare facilmente le impostazioni globali, come le intestazioni predefinite per tutte le richieste, attraverso l'uso di un oggetto di configurazione. Questo consente un adattamento preciso alle esigenze specifiche dell'applicazione, garantendo coerenza e facilitando eventuali aggiornamenti o modifiche.

Inoltre, Axios supporta in modo nativo la gestione dei cookie e fornisce un'ampia compatibilità con i browser, il che lo rende ideale per applicazioni web complesse che richiedono una comunicazione affidabile con il server.

5.4.2 Interceptors

L'autenticazione delle richieste in uscita è gestita in modo efficace attraverso l'uso degli interceptors forniti da Axios. Questi costrutti consentono la manipolazione flessibile delle richieste in uscita e delle risposte in entrata, offrendo una soluzione chiara e scalabile per garantire l'autenticazione delle comunicazioni con il server.

In primo luogo, viene configurato un nuovo componente Axios denominato axiosAuth, sfruttando il metodo axios.create. Questo componente rappresenta una versione personalizzata di Axios, preconfigurata per gestire specifiche esigenze di autenticazione.

Un passo cruciale nella configurazione di axiosAuth è la definizione di un interceptor sulle richieste in uscita. Questo interceptor svolge un ruolo chiave nell'assicurare che ogni richiesta inviata al server sia opportunamente autenticata. La sua funzione principale è accedere al token JWT memorizzato in loco, comunemente nel local storage del browser, e inserirlo nell'header della richiesta con il campo "Authorization" avente il formato "Bearer ${token}".

Questa modifica garantisce che ogni richiesta in uscita sia debitamente autenticata, consentendo al server di riconoscere l'identità dell'utente e processare la richiesta di conseguenza. Grazie a questo approccio, il flusso delle richieste e delle risposte può avvenire senza intoppi, evitando blocchi non autorizzati.

5.4.3 Refresh automatico

Durante l'esecuzione di una richiesta protetta, se l'access token è scaduto, il client può utilizzare il token di refresh per richiedere un nuovo access token senza richiedere nuove credenziali all'utente. Questo processo è trasparente per l'utente finale, garantendo un'esperienza senza interruzioni.

La richiesta di refresh token viene inviata al servizio OAuth, che verifica la validità del token di refresh associato. Se il token di refresh è valido, il servizio OAuth risponde con un nuovo access token e un nuovo token di refresh, entrambi

con nuovi tempi di scadenza. Grazie ad axios è possibile rendere questo processo automatico.

Quando una richiesta riceve uno status di risposta 401, indicativo di un accesso non autorizzato, e il campo `_retry` della richiesta originale è impostato su false, il sistema identifica la necessità di rinnovare il token. Il campo `_retry` viene quindi impostato su `true`, evitando potenziali loop infiniti di richieste in seguito all'aggiornamento.

A questo punto, entra in gioco un'istanza di Axios non autenticata, creata ad hoc per gestire il processo di refresh. Viene inviata una richiesta di tipo POST all'URL di refresh, di solito `/api/sso/refresh`, contenente il refresh token recuperato dalla memoria del browser.

Dopo aver ottenuto i nuovi access token e refresh token dalla risposta, questi vengono salvati in memoria per gli utilizzi successivi. Successivamente, viene effettuata una nuova richiesta al server, identica a quella originale, ma questa volta con il token di accesso aggiornato.

Questa strategia assicura che l'autenticazione dell'utente rimanga sempre valida, anche in presenza di token scaduti. Inoltre, l'approccio di interceptors offre una soluzione elegante e scalabile, mantenendo il flusso delle richieste e delle risposte senza interruzioni per l'utente, e garantendo al contempo un elevato livello di sicurezza nell'intero processo di autenticazione.

5.4.4 React Query

React Query [25] è una libreria che semplifica notevolmente il processo di fetch dei dati in React, eliminando la necessità di scrivere manualmente logiche complesse per gestire le richieste API. Questa libreria fornisce un'API intuitiva, consentendo agli sviluppatori di concentrarsi sulla logica dell'applicazione senza doversi preoccupare dei dettagli della gestione delle richieste di rete.

Con React Query, le chiamate API possono essere eseguite senza l'utilizzo esplicito di `useEffect` e `useState`. L'hook `useQuery` semplifica l'interazione con le API, gestendo automaticamente il ciclo di vita delle richieste e fornendo informazioni chiave come `data`, `isLoading` e `isError`. Grazie all'hook il codice mostrato in precedenza viene notevolmente semplificato:

```
1 const fetchData = async () => {
2   const response = await fetch('https://api.example.com/data');
3   const data = await response.json();
4   return data;
5 };
6
7 const ApiExample = () => {
8   const { data, isLoading, isError } = useQuery('data',
      fetchData);
```

```
 9
10    return (
11      <div>
12        {isLoading && <p>Caricamento...</p>}
13        {isError && <p>Errore nel caricamento dei dati.</p>}
14        {data && <p>Dati: {data}</p>}
15      </div>
16    );
17  };
```

Oltre a semplificare il processo di fetch dei dati, React Query offre funzionalità aggiuntive come la cache automatica dei dati. Le chiamate API vengono automaticamente memorizzate in cache, migliorando le prestazioni e garantendo una gestione ottimizzata delle richieste di rete. È possibile specificare il tempo di durata dei dati in cache, gestire automaticamente il rinnovo della cache e controllare il polling automatico per aggiornare i dati a intervalli regolari.

5.5 Elementi dinamici

L'implementazione di elementi dinamici è la parte della fase di sviluppo a cui è stata prestata più attenzione. Questa flessibilità consente di generare e visualizzare contenuti senza la necessità di apportare modifiche sostanziali sul server. Grazie a questa caratteristica, è possibile adattarsi rapidamente alle mutevoli esigenze mediche e agli aggiornamenti dei protocolli clinici. La capacità di modificare e generare dinamicamente dati e layout a partire da modifiche minime nel backend consente di mantenere l'applicazione allineata alle ultime specifiche e pratiche mediche senza interventi complessi o ricorsivi sul codice sorgente, garantendo un'aggiornamento rapido e continuo delle funzionalità.

5.5.1 Dynamic page

Il componente React in questione rappresenta un elemento chiave nell'architettura dell'applicazione, poiché è in grado di generare dinamicamente il proprio layout, adattandosi a diverse esigenze di visualizzazione dei dati. Questo componente funge da wrapper per generare il contenuto che reindirizza agli specifici sottocontenuti dell'applicazione.

Al fine di costruire il componente, si utilizza un oggetto di tipo "page" ottenuto da un servizio di layout, contenente le informazioni essenziali per la sua creazione. I soli props necessari per il suo funzionamento sono il titolo e l'endpoint da cui ottenere l'oggetto layout corrispondente.

L'oggetto layout contiene diversi parametri fondamentali:

- `name`: una stringa che identifica l'elemento;

- `dataSchema:` un oggetto che fornisce informazioni sul formato e sui dati da visualizzare, ad esempio il formato della data o la formattazione del codice fiscale;
- `type:` una stringa che definisce la struttura dell'elemento della pagina, come una tabella, una lista o un form;
- `dataURL:` una stringa che specifica l'endpoint da cui effettuare richieste per ottenere i dati necessari per popolare il componente;
- `options:` un oggetto che contiene opzioni aggiuntive di configurazione, se necessarie.

In questo modo, il componente riceve le specifiche dal servizio di layout tramite l'oggetto "page", permettendo la generazione dinamica del layout e l'accesso ai dati da fonti esterne attraverso richieste HTTP agli endpoint specificati, adattandosi così alle esigenze e al tipo di contenuto da mostrare.

5.5.2 Dynamic tab

Il componente "Dynamic Tab" si distingue dal suo analogo "Dynamic Page" poiché la sua generazione avviene non attraverso l'uso dei props, bensì basandosi sui parametri passati. Questi parametri cruciali sono "id" e "tabName".

Il parametro "id" ha il compito di identificare una singola risorsa all'interno del sistema, ad esempio, potrebbe rappresentare un paziente specifico o una visita. D'altra parte, "tabName" funge da indicatore per sottolineare una sotto-categoria specifica di dati a cui si desidera accedere all'interno della risorsa identificata.

Analogamente a "Dynamic Page", anche "Dynamic Tab" si avvale degli stessi tipi di oggetti per la generazione del contenuto. Tuttavia, la sua progettazione è finalizzata a offrire una visione più specifica e dettagliata della sotto-sezione dei dati, rispetto alla pagina dinamica di cui potrebbe essere una componente o un'interfaccia di navigazione ristretta a una singola area tematica o operativa.

5.5.3 Dynamic Element

Il componente "Dynamic Element" riveste il ruolo di un involucro generico atto a creare una varietà di elementi basandosi sul tipo specificato all'interno dello schema JSON.

Per poter generare correttamente gli elementi desiderati, il "Dynamic Element" viene istanziato all'interno di componenti più ampi come "Dynamic Page" o "Dynamic Tab". Questi componenti di livello superiore forniscono a "Dynamic Element" una serie di parametri essenziali per la sua creazione:

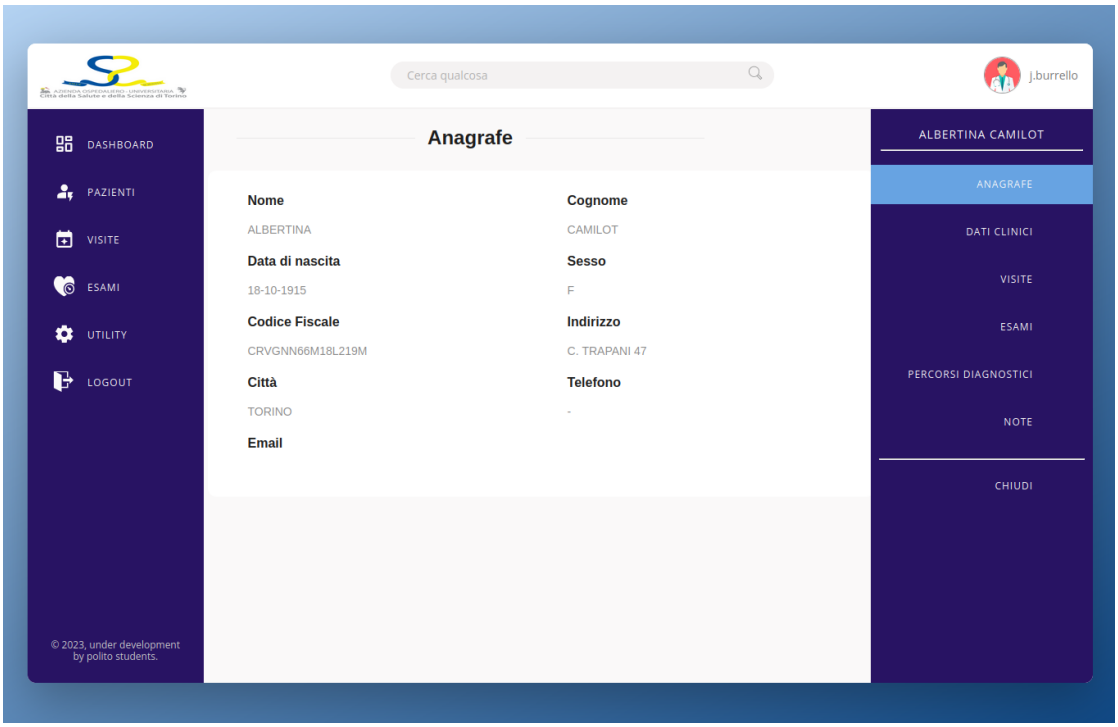

Figure 5.5: DynamicTab generato per la sezione anagrafe di un paziente

- `queryKey`: un array di stringhe che viene trasmesso a React Query, consentendo così di memorizzare nella cache i dati che verranno utilizzati per popolare l'elemento.

- `dataURL`: l'endpoint a cui il componente dovrà effettuare una richiesta di tipo GET per ottenere i dati necessari.

- `page`: l'oggetto dello schema JSON che descrive il layout specifico della pagina che si intende generare.

A questo punto, il componente "Dynamic Element" esaminerà attentamente la proprietà `type` all'interno dell'oggetto "page" per determinare il tipo di elemento che dovrà istanziare. In seguito, una volta determinato il tipo, avverrà la chiamata all'endpoint specifico per ottenere i dati utilizzando React Query, che verranno successivamente utilizzati per popolare l'elemento dinamico creato.

Attualmente, l'architettura supporta la creazione di quattro tipi distinti di elementi dinamici:

- **LIST**: Implementato tramite l'elemento "Dynamic List", progettato per mostrare diverse proprietà di un singolo oggetto, come ad esempio la sezione "anagrafe" di un paziente (Vedi fig. 5.5). Offre flessibilità nella disposizione delle proprietà grazie alla definizione di un formato di griglia specifico.

- **LINKS TABLE**: Realizzato attraverso l'elemento "Dynamic Table", finalizzato a mostrare un numero limitato di proprietà di ciascun oggetto all'interno di una collezione. Viene fornito un link che reindirizza ad una pagina dedicata con maggiori dettagli sul singolo oggetto.

- **INFO TABLE**: Anch'esso creato mediante l'elemento "Dynamic Table". Simile al "Links Table", ma invece di un link, visualizza un modal con le informazioni complete sull'oggetto selezionato. Questa soluzione è utile per la consultazione di diverse voci senza la necessità di essere reindirizzati.

- **FORM**: Costruito utilizzando l'elemento "Dynamic Form". Consente la generazione dinamica di un form basato su uno schema JSON, permettendo la validazione dei dati inseriti rispetto allo schema stesso. Fornisce la possibilità di effettuare operazioni di tipo POST verso un endpoint specifico, consentendo l'invio dei dati raccolti dal form.

In aggiunta, le Dynamic Table permettono di operazioni di filtraggio e ricerca su ogni attributo presente nell'intestazione della tabella.

5.6 Form

I form in HTML rappresentano uno strumento essenziale per raccogliere dati dagli utenti. Sono elementi interattivi che permettono agli utenti di inserire informazioni in un'applicazione web. Tuttavia, gestire i form può essere una sfida continua. La necessità di validare i dati inseriti, sia lato client che lato server, per garantirne la coerenza e la sicurezza, rappresenta una delle principali difficoltà. La validazione lato client è essenziale per fornire un feedback immediato agli utenti, mentre la validazione lato server è cruciale per proteggere l'integrità dei dati prima di salvarli permanentemente nel database.

Qui entra in gioco JSON Schema: fornisce una soluzione strutturata e affidabile per gestire la definizione dei campi di un form e, allo stesso tempo, offre uno strumento per la loro validazione. Definendo uno schema tramite JSON Schema, è possibile stabilire in modo chiaro e dettagliato quali dati devono essere inseriti in ogni campo del form e quali regole devono rispettare. Questo schema non solo fornisce una guida precisa per la struttura dei dati, ma può anche essere utilizzato per convalidare i dati inseriti, sia lato client che lato server, assicurandosi che rispettino le specifiche stabilite nello schema. Grazie a questa soluzione, è possibile gestire in maniera più robusta e affidabile la raccolta e la validazione dei dati dei form.

5.6.1 React JSONSchema Form

La libreria "react-jsonschema-form" [24] si è rivelata un'opzione potente per implementare la generazione dinamica dei form. Questa libreria offre un componente altamente personalizzabile chiamato Form, che richiede solamente pochi parametri fondamentali per funzionare in modo efficiente all'interno dell'applicazione.

Il primo parametro cruciale è lo schema, che rappresenta l'oggetto JSON Schema. Questo schema definisce non solo la struttura dei dati, ma anche le regole di validazione che devono essere applicate ai dati inseriti nell'applicazione. Definendo lo schema in maniera dettagliata, è possibile specificare i tipi di dati accettati, i vincoli di formato, i campi obbligatori e molte altre regole di validazione.

Per la validazione dei dati rispettando le specifiche dello schema, react-jsonschema-form richiede un secondo parametro, il validator. In questo caso, abbiamo scelto `ajv` [11] come libreria di validazione. `ajv` è ampiamente utilizzato nell'ecosistema JavaScript, incluso in molte applicazioni basate su Node.js, offrendo una solida e comprovata affidabilità nella validazione dei dati.

Infine, il componente Form mette a disposizione una funzione di callback chiamata onSubmit. Questa callback permette di accedere a tutti i dati inseriti nel form una volta che l'utente ha completato l'inserimento delle informazioni richieste. Questo livello di personalizzazione e controllo offerto dalla libreria react-

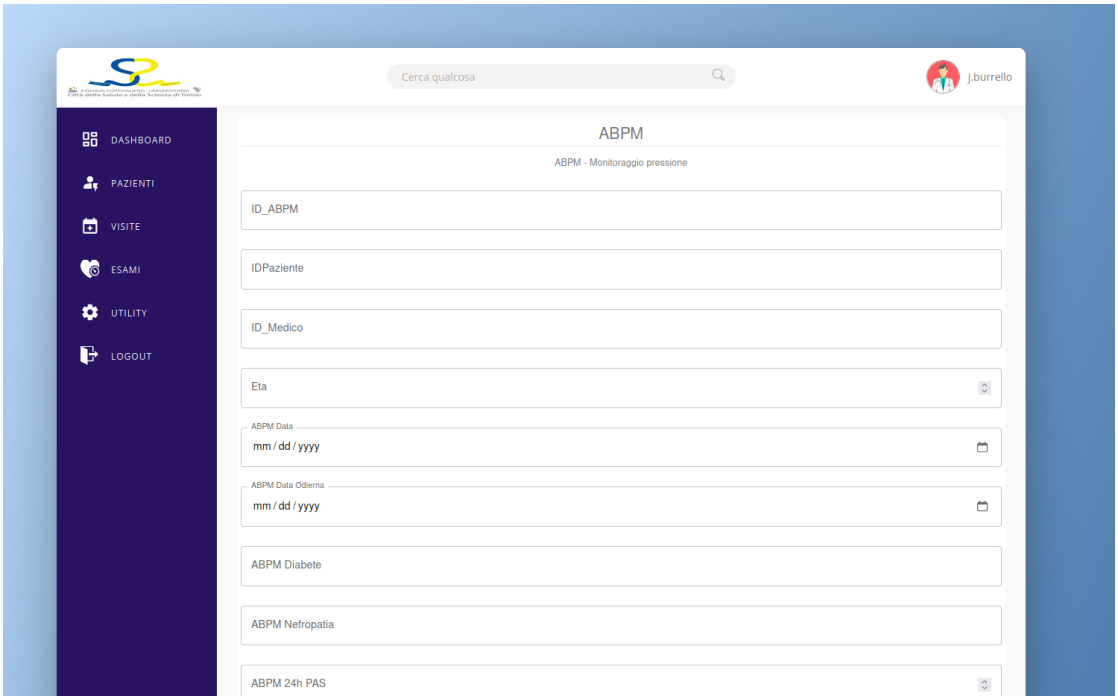

Figure 5.6: Form di un esame ABPM generato in maniera dinamica

jsonschema-form si è rivelato fondamentale per la gestione e la validazione efficace dei dati all'interno dell'applicazione.

5.7 Flowchart

La generazione dei flowchart per i percorsi diagnostici è stata una fase cruciale nello sviluppo dell'applicazione. L'obiettivo era ottenere un sistema dinamico e funzionale che rispondesse a specifiche precise:

Innanzitutto, dovevano essere generati dinamicamente, sfruttando gli stessi oggetti JSON utilizzati per la generazione delle pagine web. Questa integrazione garantiva uniformità di informazioni tra i percorsi diagnostici e i dati visualizzati agli utenti.

Una caratteristica chiave era la disposizione automatica e intelligente dei nodi e degli archi. Era fondamentale che la disposizione fosse ottimale, evitando sovrapposizioni e congestioni sullo schermo, garantendo così una chiara comprensione del percorso diagnostico.

Infine, l'aspetto visivo dei flowchart doveva rispettare il tema grafico dell'intera applicazione. Ciò richiedeva l'implementazione di una libreria aggiornata e altamente customizzabile, in modo da garantire coerenza estetica e funzionale con il

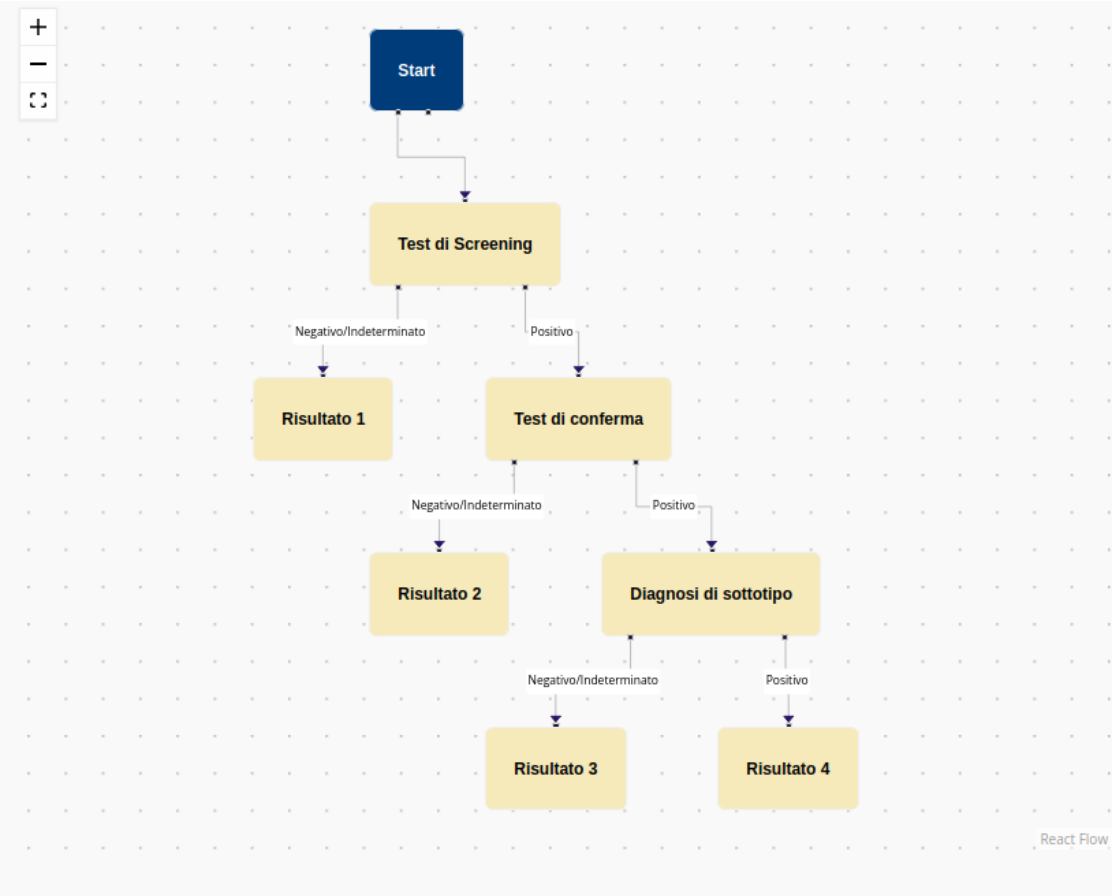

Figure 5.7: Esempio di flowchart generato dinamicamente

resto dell'applicazione.

5.7.1 React Flow

La generazione dei flowchart è stata resa possibile grazie all'implementazione della libreria React Flow [23]. Questa libreria offre un componente chiamato `Flow` che agevola la creazione dei nodi e degli archi all'interno del flowchart stesso.

Per definire la struttura del flowchart, si utilizzano oggetti JSON che rappresentano i nodi e gli archi. Un oggetto "nodo" contiene informazioni come il nome dell'esame, un id numerico univoco e il tipo di nodo, che può essere di inizio (senza archi in entrata ma con uno in uscita), di procedura (con almeno un arco in entrata e uno o più in uscita) o di diagnosi (finale, senza archi in uscita).

Gli oggetti "arco" invece contengono informazioni riguardanti l'id del nodo di partenza, l'id del nodo di destinazione e il nome della connessione. Quest'ultima

corrisponde all'esito dell'esame ed è fondamentale per automatizzare il percorso diagnostico.

La posizione dei nodi viene calcolata dinamicamente utilizzando la libreria "dagre" [14], totalmente indipendente da React Flow. Si utilizza la funzione `Graph()` che restituisce un oggetto grafo a cui vengono aggiunti i dati relativi ai nodi e agli archi del percorso diagnostico tramite le funzioni `setNode` e `setEdge`.

Infine, chiamando la funzione `Dagre.layout(graph)`, si assegna agli oggetti "nodo" del grafo la proprietà posizione, espressa in coordinate x e y. Queste informazioni consentono a React Flow di visualizzare e posizionare correttamente i nodi all'interno del flowchart.

5.8 Deployment

Per integrare il front-end in modo ottimale nel sistema complessivo dell'ospedale, si è scelto di utilizzare Docker. Sfruttando Docker, si garantisce un ambiente uniforme tra diversi sistemi e si semplifica il deployment, incapsulando l'applicazione e le sue dipendenze in contenitori virtuali. Questo approccio non solo agevola il deployment, ma assicura anche scalabilità ed efficienza nella gestione dell'applicazione all'interno dell'infrastruttura più ampia dell'ospedale.

5.8.1 Docker

Docker [15] è una piattaforma che semplifica la creazione, la distribuzione e l'esecuzione di applicazioni all'interno di contenitori. Un contenitore è un'istanza leggera e isolata di un'applicazione, che include tutto il necessario per l'esecuzione: codice, librerie, dipendenze e configurazioni. Rispetto alle macchine virtuali, i container offrono un'efficienza superiore in quanto condividono il kernel del sistema operativo ospite, evitando sovraccarichi di risorse e avviandosi più rapidamente.

Docker consente di gestire facilmente i container, fornendo un ambiente standardizzato per lo sviluppo, il test e il deployment delle applicazioni. Inoltre, permette l'orchestrazione di più container attraverso strumenti come Kubernetes. Kubernetes offre funzionalità avanzate di orchestrazione, consentendo di gestire, scalare e monitorare automaticamente i container in un ambiente distribuito.

Integrare l'applicazione nell'ecosistema degli altri servizi attraverso Docker significa garantire coerenza e facilità di gestione. I container Docker possono essere facilmente distribuiti e replicati, fornendo una soluzione scalabile e affidabile per l'integrazione delle applicazioni in un ambiente più ampio, come quello di un ospedale, consentendo una gestione semplificata e una maggiore flessibilità nell'implementazione e nell'esecuzione di servizi.

Capitolo 6

Conclusioni

Attualmente, il progetto ha raggiunto uno stato di notevole versatilità e completezza, con risultati tangibili che testimoniano l'efficacia delle scelte progettuali e degli strumenti implementati. Abbiamo sviluppato un sistema estremamente versatile, caratterizzato dalla capacità di adattarsi a nuove funzionalità con modifiche minime, grazie all'implementazione di un'architettura modulare basata su JSON schemas presenti sul server.

Uno dei successi principali è rappresentato da un sistema di autenticazione automatico e flessibile, indipendente dal servizio scelto, purché supporti Oauth e JWT per l'autenticazione. Questo approccio garantisce una gestione sicura e personalizzabile dell'accesso all'applicazione, consentendo al contempo un facile adattamento a eventuali evoluzioni o cambiamenti nei protocolli di autenticazione.

Un'altra caratteristica chiave è la possibilità di definire e visualizzare in modo automatico nuovi percorsi diagnostici. Questa funzionalità, derivante dalle richieste dei medici, si è rivelata essenziale per uniformare e semplificare il processo diagnostico, migliorando l'efficienza complessiva dell'applicazione.

La gestione automatica della cache è stata implementata con successo, garantendo una risposta più veloce alle richieste frequenti e contribuendo all'ottimizzazione delle prestazioni dell'applicazione. Inoltre, il deployment del servizio è reso efficiente e scalabile grazie all'utilizzo di Docker, semplificando la distribuzione su diversi ambienti.

In conclusione, il progetto ha raggiunto un livello avanzato di funzionalità, fornendo una solida base per la gestione delle informazioni mediche nella cartella clinica. Gli elementi chiave, come la versatilità del sistema, l'autenticazione flessibile, la definizione automatica dei percorsi diagnostici, la gestione automatica della cache e il deployment tramite Docker, testimoniano il successo delle scelte progettuali e l'efficacia delle implementazioni.

6.1 Sviluppi futuri

Guardando al futuro, il progetto presenta diverse prospettive di sviluppo per migliorare ulteriormente le prestazioni e l'accessibilità. Una delle prossime tappe prevede l'implementazione del Server-Side Rendering (SSR), un passo significativo per ottimizzare i tempi di caricamento delle pagine. L'introduzione del SSR permetterà di pre-caricare le pagine lato server, migliorando l'esperienza utente attraverso tempi di risposta più rapidi e una navigazione più fluida.

Un altro aspetto cruciale è l'espansione dell'applicazione per dispositivi mobili. Questo implica la definizione di un'interfaccia responsive, adattando il layout e il design per garantire un'esperienza utente ottimale su diverse dimensioni di schermo. L'estensione a dispositivi mobili non solo amplia il raggio di utilizzo dell'applicazione, ma risponde anche alla crescente tendenza di accesso alle informazioni mediche da parte degli operatori sanitari attraverso dispositivi portatili.

Parallelamente, è prevista l'implementazione di funzionalità per trasformare l'applicazione in una Progressive Web App (PWA). Questo permetterà agli utenti di installare l'applicazione direttamente sul proprio dispositivo e accedervi come un'applicazione nativa, migliorando l'usabilità e consentendo un accesso più rapido e immediato alle funzionalità dell'app anche in assenza di connessione internet.

In sintesi, gli sviluppi futuri del progetto si concentreranno sull'implementazione del SSR per ottimizzare i tempi di caricamento, sull'adattamento dell'interfaccia per dispositivi mobili per migliorare l'accessibilità, e sull'evoluzione verso una Progressive Web App per offrire un'esperienza utente ancora più avanzata e flessibile. Queste prospettive mirano a mantenere il progetto all'avanguardia, soddisfacendo al meglio le esigenze degli utenti e adattandosi alle tendenze emergenti nel campo delle applicazioni web.

Bibliografia

[1] S. Allamaraju and S. Allamaraju. *RESTful Web Services Cookbook*. O'Reilly Series. O'Reilly Media, 2010. ISBN: 9780596801687. URL: https://books.google.it/books?id=ed5ml0T3zyIC.

[2] J.S. Beasley and P. Nilkaew. *Networking Essentials: A CompTIA Network+ N10-006 Textbook*. Pearson Education, 2015. ISBN: 9780134299747. URL: https://books.google.it/books?id=TVdCCwAAQBAJ.

[3] D. Crockford. *Javascript: The Good Parts*. Shroff Publishers & Distributors. ISBN: 9788184045222. URL: https://books.google.it/books?id=u435jwEACAAJ.

[4] J. Gillies and R. Cailliau. *How the Web was Born: The Story of the World Wide Web*. Oxford paperback reference. Oxford University Press, 2000. ISBN: 9780192862075. URL: https://books.google.it/books?id=pIH-JijUNS0C.

[5] C. Henderson. *Building Scalable Web Sites*. O'Reilly Series. O'Reilly Media, Incorporated, 2006. ISBN: 9780596102357. URL: https://books.google.it/books?id=hzybAgAAQBAJ.

[6] S. Newman. *Building Microservices*. O'Reilly Media, 2021. ISBN: 9781492033998. URL: https://books.google.it/books?id=aPM5EAAAQBAJ.

[7] D.A. Norman. *The Design of Everyday Things*. The MIT Press. MIT Press, 2013. ISBN: 9780262525671. URL: https://books.google.it/books?id=heCtnQEACAAJ.

[8] J. Richer and A. Sanso. *OAuth 2 in Action*. Manning, 2017. ISBN: 9781638352280. URL: https://books.google.it/books?id=jTozEAAAQBAJ.

[9] J.N. Robbins. *Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics*. Oreilly and Associate Series. O'Reilly Media, Incorporated, 2012. ISBN: 9781449319274. URL: https://books.google.it/books?id=FJkVxtXr7n0C.

[10] L. Shklar and R. Rosen. *Web Application Architecture: Principles, Protocols and Practices*. Wiley, 2009. ISBN: 9780470518601. URL: https://books.google.it/books?id=gcHJQwAACAAJ.

Sitografia

[11] *Ajv JSON schema validator*. URL: https://ajv.js.org/.

[12] *Ant Design*. URL: https://ant.design/docs/react/introduce.

[13] *Axios*. URL: https://axios-http.com/.

[14] *Dagre*. URL: https://www.npmjs.com/package/@dagrejs/dagre.

[15] *Docker*. URL: https://www.docker.com/.

[16] *ECMAScript*. URL: https://ecma-international.org/publications-and-standards/standards/ecma-262/.

[17] *Fondazione LINKS*. URL: https://linksfoundation.com.

[18] *JSON Schemas*. URL: https://json-schema.org/.

[19] *JWT*. URL: https://jwt.io/.

[20] *Keycloak*. URL: https://www.keycloak.org/documentation.html.

[21] *PWA*. URL: https://web.dev/explore/progressive-web-apps.

[22] *React*. URL: https://react.dev/.

[23] *React Flow*. URL: https://reactflow.dev/.

[24] *React JSON Schema Form*. URL: https://github.com/rjsf-team/react-jsonschema-form.

[25] *React Query*. URL: https://tanstack.com/query/v3/docs/react/overview.

[26] *React Router*. URL: https://reactrouter.com/en/main.

[27] *Web Sockets*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket.