Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Informatica
A.a. 2022/2023
Sessione di Laurea Luglio 2023

Interfaccia front-end per il sistema di monitoraggio del benessere degli assistiti medici: Design e usabilità

Relatore:
Prof. Maurizio MORISIO

Candidato:
Dott. Alberto CIPOLLINA

SOMMARIO

ELENCO DELLE FIGURE

I) INTRODUZIONE

1. HealthApp

HealthApp è un progetto che punta al miglioramento dello stato di salute dei pazienti attraverso consigli relativi alla dieta e all'attività fisica. Questa iniziativa è frutto di una collaborazione tra i medici dell'Azienda Ospedaliera di Verona e il Dipartimento di Automatica e Informatica (DAUIN) del Politecnico di Torino, con l'obiettivo di sviluppare un sistema software nel campo medico per il monitoraggio dei parametri salutari dei pazienti e per supportare ulteriori ricerche nel settore del miglioramento della salute.

L'applicazione HealthApp offre diverse modalità di interazione. In primo luogo, è disponibile un'interfaccia mobile dedicata ai pazienti, che rappresentano il principale target utente. Attraverso questa piattaforma, i pazienti possono consultare i propri parametri salutari e ricevere raccomandazioni personalizzate. Inoltre, è disponibile un'interfaccia web che costituisce lo strumento principale utilizzato dai medici per monitorare in tempo reale lo stato dei pazienti e che rappresenta l'oggetto di studio di questa tesi. Un back-end appositamente sviluppato in una fase precedente elabora le richieste provenienti dalle applicazioni mobile e web. Un elemento fondamentale per la ricerca all'interno del progetto HealthApp è stato l'impiego del dispositivo Fitbit, noto per il tracciamento accurato dei dati relativi al sonno, ai passi e ai battiti cardiaci. Al momento, HealthApp si integra esclusivamente con Fitbit, ma è prevista l'integrazione di altri braccialetti tracker in futuro. Ciò consentirà non solo di supportare un'ampia gamma di dispositivi, ma anche di sfruttare tecnologie più avanzate come la rilevazione della pressione sanguigna o altri dati che Fitbit da solo non raccoglie. I dati raccolti da Fitbit rappresentano il punto di partenza per avviare l'indagine sulla salute dei pazienti. Tuttavia, i parametri biometrici da soli non sono sufficienti per identificare complessivamente lo stato di salute di un paziente. Sono necessari ulteriori dati, come quelli riguardanti le abitudini alimentari e, più in generale, la dieta che i pazienti affrontano nella loro quotidianità. Per questo motivo, è stata implementata nel sistema la funzionalità che consente ai medici di creare questionari personalizzati o di utilizzare altre scale di valutazione derivate direttamente dalla letteratura scientifica sul benessere. L'integrazione di dati provenienti da Fitbit, insieme alle informazioni sulla dieta e alle valutazioni personalizzate dei medici, consente a HealthApp di fornire una visione completa e dettagliata dello stato di salute di un paziente. Questo approccio basato su dati multipli e personalizzati è fondamentale per offrire raccomandazioni mirate e supporto medico adeguato, promuovendo così un miglioramento effettivo del benessere complessivo dei pazienti.

2. Obbiettivo della tesi

L’obbiettivo principale di questa tesi è stato impostare un'architettura efficace per lo sviluppo di un'applicazione web destinata ai medici per il supporto nell'esperimento di HealthApp.

Durante la definizione degli obiettivi, è stata posta particolare attenzione alle esigenze degli utilizzatori principali, i medici, al fine di garantire un'interfaccia utente intuitiva e facilmente fruibile. L'obiettivo centrale è stato quello di creare un'applicazione web che fosse facilmente estendibile, scalabile e mantenibile nel lungo periodo. Questo permette di offrire un'esperienza utente ottimale e agevolare il lavoro dei medici nel monitoraggio e nella consulenza dei pazienti. La scelta di utilizzare la libreria Material-UI per React e Redux per la gestione dello stato dell'applicazione ha giocato un ruolo fondamentale nel raggiungimento di tali obiettivi. L'utilizzo di Material-UI ha consentito di realizzare un'interfaccia utente moderna e coerente, offrendo componenti predefiniti ben progettati e personalizzabili. Questo ha semplificato il processo di sviluppo e ha permesso ai medici di apprendere rapidamente le interazioni con l'applicazione front-end, riducendo la curva di apprendimento. Inoltre, l’utilizzo di Redux per la gestione dello stato dell'applicazione ha fornito un meccanismo efficace per la condivisione e l'aggiornamento dei dati tra i diversi componenti dell'applicazione. Questo ha contribuito a garantire coerenza e coesione all'interno dell'applicazione, facilitando anche la gestione delle interazioni con il back-end.

L'organizzazione del workspace, l'utilizzo delle librerie per la gestione dello stato dell'applicazione e l'interfaccia utente utilizzate in HealthApp possono essere adottati come best practice per lo sviluppo di futuri progetti simili.

3. Architettura dell'applicazione

Lo schema architetturale dell'applicazione di sistema HealthApp segue il tradizionale schema a tre livelli (*three-tier*) [1] (**Figura 1**). Questo tipo di architettura consente di separare l'applicazione in diversi livelli fisici e logici, offrendo così maggiore manutenibilità e flessibilità. Inoltre, ha il vantaggio di permettere la sostituzione di uno dei livelli senza dover modificare l'intera struttura dell'applicazione.

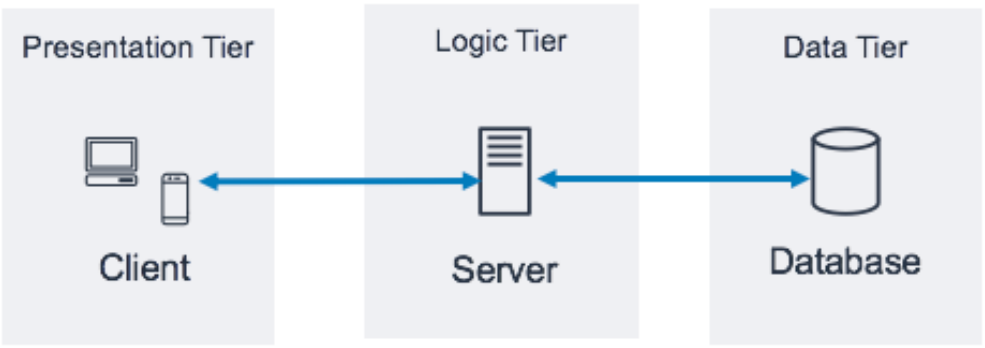

Figura 1 - Architettura a tre livelli [2].

I tre livelli principali sono i seguenti:

1) *Data tier*: che gestisce la persistenza e la lettura dei dati a supporto dell'applicazione;
2) *Logic tier*: in cui risiede l'applicativo back-end, che fa da intermediario tra le richieste provenienti dal *presentation tier* e il livello sottostante o altri servizi esterni;
3) *Presentation tier*: che contiene l'interfaccia utente, in cui è possibile visualizzare i dati e interagire con il sistema attraverso opportune azioni.

Tuttavia, se osserviamo il deployment diagram (**Figura 2**), notiamo che il client web non è l'unico front-end del sistema, poiché è presente anche un'interfaccia mobile all'interno del *Presentation tier*. Entrambe le interfacce hanno alcuni fattori in comune, come la visualizzazione dei dati relativi ai parametri di salute, e offrono caratteristiche specifiche per gli utilizzatori.

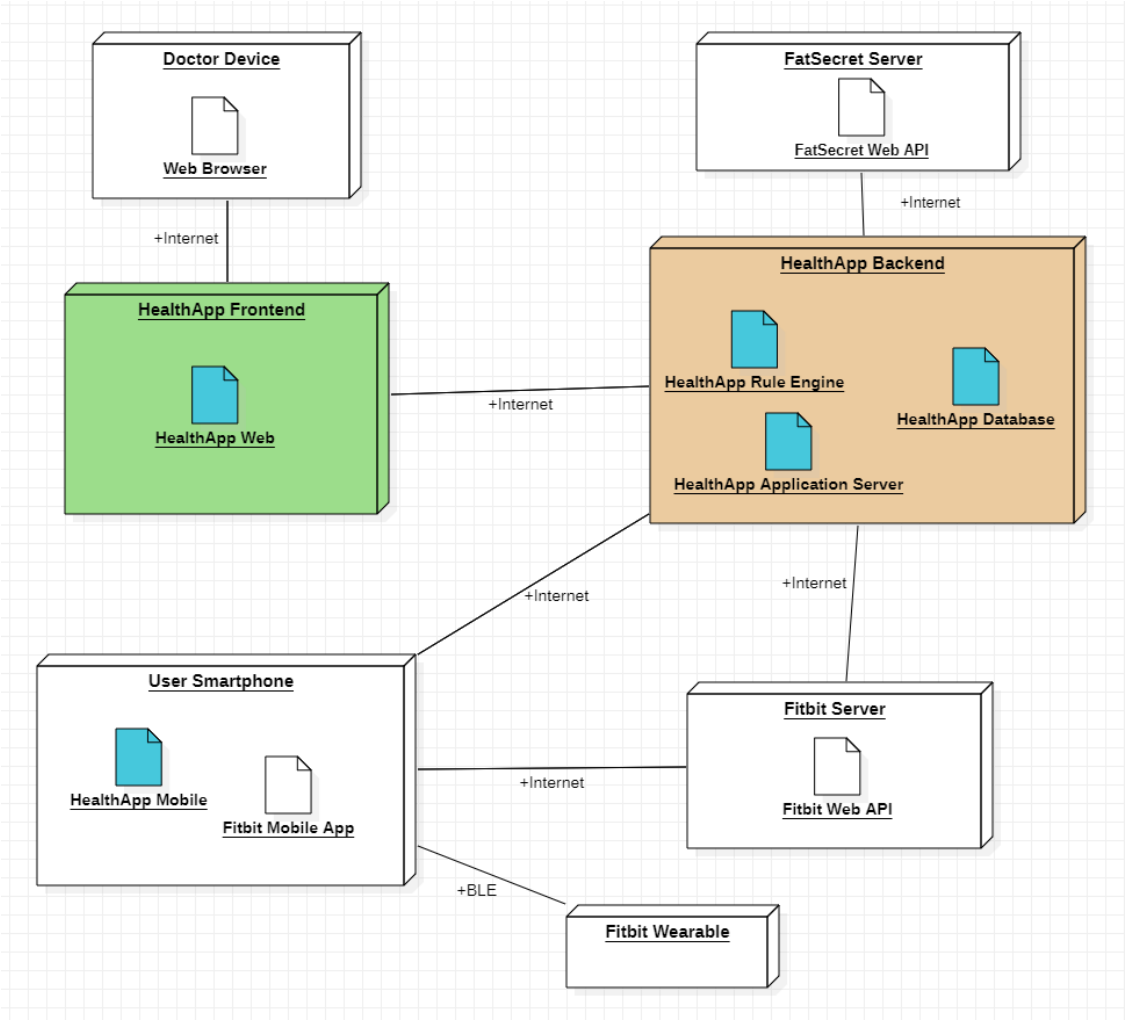

Figura 2 – Deployment Diagram.

3.1 Back-end

Nel progetto di HealthApp, il back-end (BE) è stato sviluppato utilizzando Spring Boot [3], un framework applicativo che, in questo caso, è stato basato sul linguaggio di programmazione Kotlin. Il motivo per cui è stato scelto Spring Boot dipende principalmente da due fattori: la sua robustezza e la rapidità nella consegna dell'applicazione.

Spring Boot è un framework estremamente popolare per lo sviluppo di applicazioni BE. Una delle caratteristiche principali di Spring Boot è quella di fornire agli sviluppatori un set completo di strumenti per creare moduli di sicurezza, gestire l'interazione con vari tipi di database [4], gestire code come Kafka [5], gestire l'autenticazione dei servizi [6], e molte altre funzionalità.

Nel corso degli anni, Spring Boot si è evoluto diventando sempre più efficiente in termini di consumo di risorse. Questa efficienza è particolarmente desiderabile per applicazioni che devono gestire un carico di lavoro di dimensioni ragionevolmente elevate. Altri server, come ad esempio quelli basati su Node.js con l'uso di Express.js [7], potrebbero sembrare soluzioni alternative valide. La differenza fondamentale tra questi due approcci risiede, non solo nel linguaggio di programmazione utilizzato, quale Java/Kotlin per Spring Boot e JavaScript/TypeScript per Node.js, ma anche nel modello sottostante per l'implementazione del server. Node.js, ad esempio, si basa su uno schema "single-threaded" e le richieste sono gestite attraverso un modello di event loop non bloccante [8]. Questo modello funziona bene per applicazioni con carichi di lavoro leggeri, ma può avere limitazioni quando si tratta di applicazioni con carichi di lavoro pesanti. Al contrario, Spring Boot utilizza un modello "multi-threaded" che permette di sfruttare più risorse assegnando un thread dove necessario. Questo lo rende ideale per applicazioni che devono gestire un numero elevato di richieste simultaneamente.

Il back-end di HealthApp, oltre ad implementare i servizi REST esposti alle interfacce mobile e web, ingloba Drools [9], un business rule engine il cui obiettivo nel contesto del progetto HealthApp è quello di elaborare consigli sulla base dei dati raccolti dal tracker e derivati dai questionari.

3.2 Front-end

Come anticipato nel paragrafo relativo allo schema architetturale, il front-end è ulteriormente suddiviso in due parti: Mobile Application e Web App Application. Entrambe hanno una matrice in comune, ovvero quelle di essere sviluppate con la libreria React.

3.2.1 Mobile Application

L'applicazione Mobile è interamente destinata ai pazienti che aderiscono all'esperimento di HealthApp ed è stata sviluppata utilizzando React Native. React Native è un framework per lo sviluppo di applicazioni mobile che consente di creare applicazioni native per IOS e Android utilizzando la libreria React basata sui componenti [10]. Pertanto, il vantaggio nell'utilizzo di questo framework di sviluppo cross-platform, è quello di avere una singola code base che permette di abbattere i tempi di sviluppo e di messa in produzione.

3.2.2 Web Application

Il client web di questo progetto è stato sviluppato utilizzando il framework React [11]. Questa libreria JavaScript (JS) ha riscosso un grande successo nel mondo dello sviluppo front-end grazie alla sua praticità nell'implementazione di applicazioni web complesse, posizionandosi tra i principali strumenti utilizzati per la creazione di interfacce utente.

Una delle caratteristiche fondamentali di React è il suo modello basato su componenti. I componenti rappresentano le unità fondamentali dell'applicazione, incorporando sia la logica visuale per la presentazione dei contenuti, sia le interazioni attraverso le quali gli utenti interagiscono con il sistema. Questo approccio consente una gestione più ordinata e intuitiva del codice, favorendo il riuso dei componenti in diverse parti dell'applicazione web e promuovendo una manutenibilità del codice più efficiente grazie alla sua struttura modulare. La struttura a componenti non solo facilita la manutenzione e la leggibilità del codice, ma contribuisce anche a rendere l'interfaccia utente più robusta e testabile (**Figura 3**).

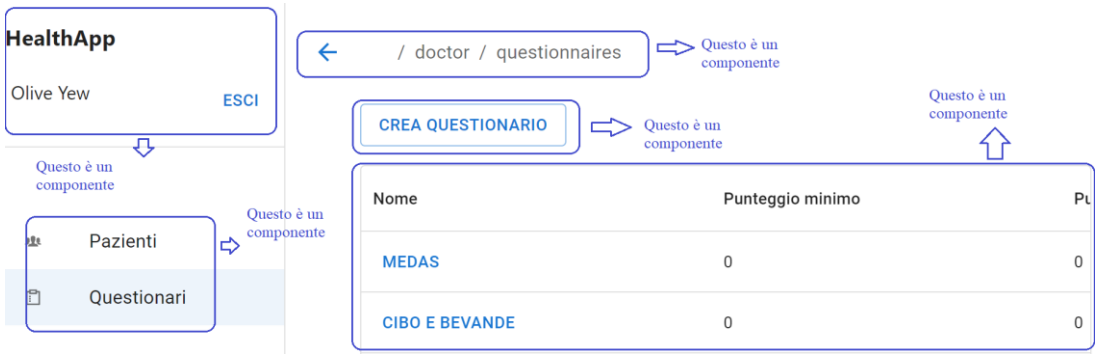

Figura 3 – Esempio di componenti React utilizzati nello sviluppo di una pagina in HealthApp.

In React ci sono due tipi di componenti: “functional component” (**Figura 4**) e “class component” (**Figura 5**), rispettivamente caratterizzati da una funzione e da una classe JS che estende la classe React.Component [12].

```
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
      </header>
    </div>
  );
}
```

Figura 4 – Esempio del functional component principale di un'applicazione React [12].

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Figura 5 – Esempio di class componenent [12].

Entrambi gestiscono le "props" ovvero le proprietà attraverso cui è possibile trasferire dati al componente e che vengono impiegate nella visualizzazione nell'interfaccia utente. Le "props" sono "read-only", cioè non è possibile modificarle all'interno del componente a cui vengono passate [13]. Se l'intento invece è quello di mantenere le informazioni nel componente e modificarle attraverso le interazioni con la UI occorre utilizzare il concetto di "stato". Per i class component è possibile gestirlo tramite il metodo "setState()" a cui va passata la copia dello stato corrente con i soli campi che si intendono modificare [14] (**Figura 6**).

```
import React, { Component } from 'react'

class App extends Component {
constructor(props){
	super(props)

	// Set initial state
	this.state = {greeting :
		'Click the button to receive greetings'}

	// Binding this keyword
	this.updateState = this.updateState.bind(this)
}
updateState(){
	// Changing state
	this.setState({greeting :
			'GeeksForGeeks welcomes you !!'})
}
render(){
	return (
	<div>
	<h2>Greetings Portal</h2>
	<p>{this.state.greeting}</p>

		{/* Set click handler */}
		<button onClick={this.updateState}>
		Click me!
		</button>
	</div>
	)
}
}
```

Figura 6 – Esempio di utilizzo del metodo setState() per un class component in React [15].

Nel caso dei "functional component" l'approccio è diverso e richiede l'uso dell'hook "useState" [14] (**Figura 7**). Questo restituisce in output un array di due elementi che rispettivamente rappresentano lo stato e una funzione per aggiornarlo [16].

```
function AnalisiMediche(props) {
…
  const [bloodAnalysisLoading, setBloodAnalysisLoading] = useState(true);
  const [patientWeightLoading, setPatientWeightLoading] = useState(true);
  const [bloodAnalysisRangeLoading, setBloodAnalysisRangeLoading] = useState(
    true);
…
  useEffect(() => {
    if (patientWeightLoading) {
      setPatientWeightLoading(false);
      dispatch(getPatientWeights(patientId));
    }
    if (bloodAnalysisLoading) {
      setBloodAnalysisLoading(false);
      dispatch(getPatientBloodAnalysis(patientId));
    }
    if (bloodAnalysisRangeLoading) {
      setBloodAnalysisRangeLoading(false);
      dispatch(getBloodAnalysisRange());
    }

},[patientWeightLoading,bloodAnalysisLoading,bloodAnalysisRangeLoading,dispa
tch,patientId]);
```

Figura 7 – Esempio di utilizzo del hook useState() di React in HealthApp per indicare il termine del caricamento degli elementi nella pagina.

È possibile specificare molteplici hook, in quanto un componente potrebbe gestire più informazioni. La loro dichiarazione deve essere posta prima del "return" in cui viene specificato il codice per la resa grafica del componente [16]. Un'altra differenza tra le due

tipologie di componenti deriva dal fatto che i “class components” espongono dei metodi per la gestione del ciclo di vita del componente stesso che sono:

1) *componentDidMount*: questo metodo viene chiamato la prima volta che il componente viene montato sul DOM e viene impiegato per configurare eventuali event listener o timer [17];
2) *componentDidUpdate*: questo metodo viene utilizzato ogni volta che il componente viene aggiornato in relazione al cambiamento di stato, tramite la funzione setState [17].

Nel contesto di HealthApp sono stati utilizzati solo i “functional component” per la loro semplicità, in quanto non sono risultate necessarie le altre funzionalità relative ai “class component”.

II) ASPETTI FUNZIONALI DELLA WEB APPLICATION

1. Funzionalità dell'applicazione

In questo paragrafo verranno analizzate le principali funzionalità dell'applicazione che sono messe a disposizione dei medici, gli utilizzatori principali del client web. In particolare, si approfondiranno gli step necessari per portare a termine i task:

1) Registrazione del paziente;
2) Creazione dei template dei questionari;
3) Compilazione dei questionari;
4) Visualizzazione dei dati biometrici;
5) Inserimento e visualizzazione delle analisi del sangue;
6) Inserimento e visualizzazione del peso;
7) Inserimento dei valori soglia.

1.1 Registrazione del paziente

L'implementazione della registrazione dei pazienti rappresenta la prima fondamentale operazione da eseguire all'interno dell'applicazione HealthApp. Questo processo implica la raccolta di dati fondamentali per l'identificazione dei pazienti, tra cui nome, cognome, data di nascita, sesso, codice fiscale e altezza (**Figura 8**). Oltre ai dati anagrafici va specificata la tipologia, cioè, se si tratta di paziente "sperimentale" o di "controllo". Nel primo caso il paziente aderisce all'esperimento; nel secondo invece il medico decide di sfruttare HealthApp per monitorare semplicemente i dati biometrici e quelli derivati dai questionari del paziente senza alcuna finalità nel contesto del progetto di HealthApp. Infine, vengono richiesti i dati relativi ad username e password che il paziente dovrà fornire e immettere personalmente durante la visita con il medico. Nonostante questa procedura venga svolta dal medico in presenza del paziente, quest'ultimo in un secondo momento può effettuare il cambio password direttamente dall'interfaccia mobile.

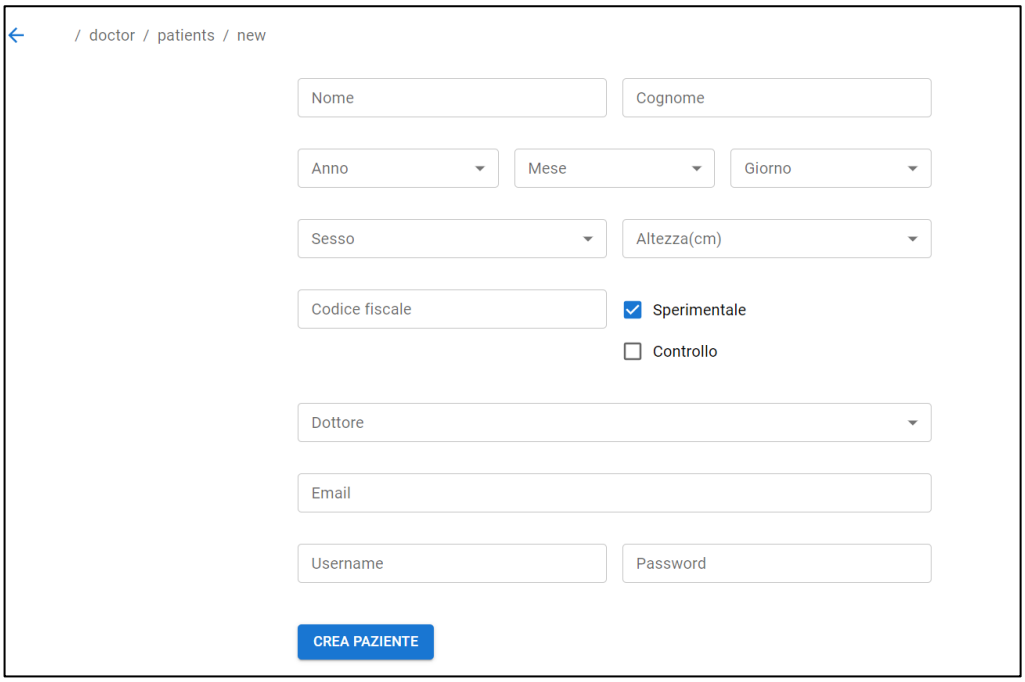

Figura 8 – Form di registrazione del paziente in HealthApp.

Un elemento di particolare importanza è il codice identificativo che viene generato dopo la sottomissione del form di registrazione. Questo viene mostrato nell'interfaccia grafica nel formato "P"<id>, dove "id" è il valore della chiave primaria memorizzata nel database (**Figura 9**). L'impiego di un codice identificativo per riconoscere i pazienti all'interno di HealthApp non risponde unicamente a esigenze tecniche, come l'adozione di una forma rappresentativa più concisa. In realtà, si tratta di una scelta ponderata, fortemente influenzata da questioni legate alla privacy. Durante le consultazioni con i medici, è emersa l'esigenza di utilizzare un identificativo per sostituire nome e cognome. Questo perché, nelle future implementazioni della versione web di HealthApp, sarà possibile estrarre la lista dei pazienti e le relative informazioni correlate, come i parametri salutari e i dati derivanti dai questionari. Il formato dei file estratti potrà essere .txt, .xlsx o .csv; tuttavia, il punto cruciale risiede nel fatto che tali file potrebbero essere elaborati da strumenti di analisi statistica di terze parti. Pertanto, per proteggere i dati sensibili come nome e cognome, si è deciso di optare per l'uso di un codice. È importante sottolineare che, all'interno di HealthApp, i dati possono essere consultati in chiaro, tranne le password, in quanto solo i medici sono autorizzati ad accedere ai dati sensibili dei pazienti.

Figura 9 – Esempio di visualizzazione dei dati del paziente identificato dal codice alfanumerico P7 in HealthApp.

La registrazione del paziente viene considerata completata, dopo che il medico ha effettuato con successo l'operazione di registrazione sul client web, e successivamente, il paziente ha effettuato il primo accesso utilizzando le stesse credenziali sul client web. Durante il primo accesso, al paziente sarà richiesto di effettuare, inoltre, il login su Fitbit facendo clic sul pulsante "procedi" (**Figura 10**). Successivamente, sarà necessario accettare i consensi relativi alla raccolta dei dati biometrici da parte dell'applicazione HealthApp. Infine, verrà visualizzato un messaggio sull'interfaccia per confermare che la registrazione è stata completata (**Figura 11**), consentendo al paziente di utilizzare l'applicazione mobile.

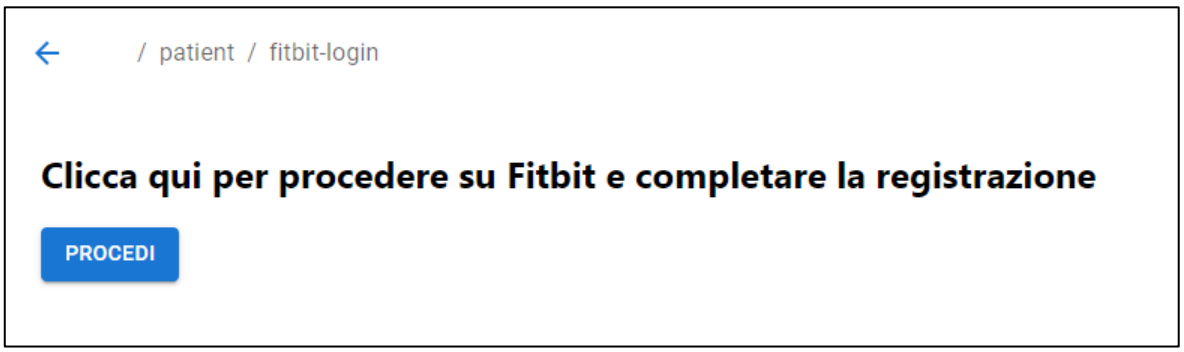

Figura 10 – Componente che mostra all'utente l'azione di proseguire su Fitbit.

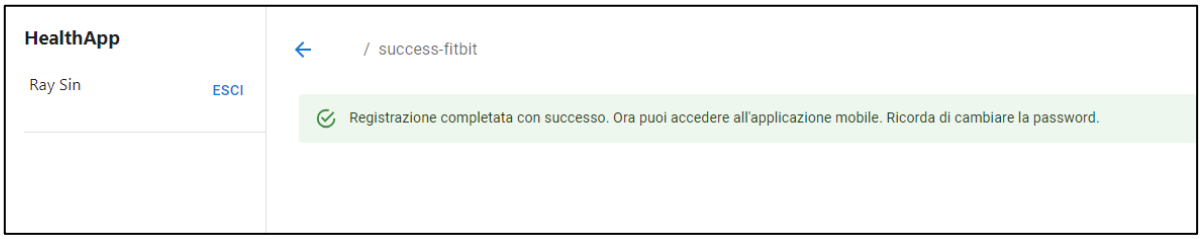

Figura 11 – Componente che mostra il completamento della registrazione del paziente su HealthApp.

1.2 Creazione dei template dei questionari

Nonostante la rilevanza della raccolta dei dati riguardanti i parametri di salute, quali sonno, conteggio dei passi e frequenza cardiaca, acquisiti da dispositivi tracker, volta a migliorare la salute dei pazienti coinvolti nel progetto, è stato necessario integrare l'approccio con un metodo più selettivo e focalizzato sulle abitudini dei pazienti. Infatti, vi sono informazioni che non possono essere raccolte direttamente dai dispositivi come Fitbit, ma necessitano di un approccio differente, quale l'utilizzo dei questionari. Nel corso degli incontri con i medici, è emersa la necessità di sfruttare questionari derivati direttamente dalla letteratura scientifica

sul benessere e la salute, come il MEDAS [18]. Questo specifico strumento di indagine viene sottoposto al paziente durante l'esperimento e permette di acquisire dati preziosi sulle abitudini alimentari o di altra natura, essenziali per la valutazione del suo stile di vita e della sua salute.

Tuttavia, i soli questionari ereditati dalla letteratura scientifica possono essere insufficienti o limitati. Non tutti, infatti, possono risultare adeguati al contesto specifico di un esperimento. Per questo motivo, durante gli incontri di pianificazione, è emersa la necessità di sviluppare un questionario personalizzato, prodotto direttamente dai medici coinvolti. Utilizzando il questionario MEDAS come modello di riferimento, è stato sviluppato un template per la creazione di questionari personalizzati, curando particolarmente l'interfaccia utente. L'obiettivo era di replicare, il più fedelmente possibile, l'esperienza di creazione e compilazione di un questionario in formato cartaceo.

La creazione di un questionario inizia con l'inserimento di un titolo (**Figura 12**).

Figura 12 – Campo di testo per l'inserimento del nome del questionario.

Successivamente, si procede alla selezione delle sezioni di domande (**Figura 13**): queste rappresentano un modo per categorizzare le domande, dettaglio che può rivelarsi prezioso durante l'analisi statistica dei dati raccolti.

Figura 13 – Esempio di inserimento di sezioni delle domande.

Nella sezione apposita si definisce la domanda (**Figura 14**) e si procede all'inserimento delle possibili risposte (**Figura 15**), ognuna delle quali può essere associata a una sezione. Dopo l'inserimento delle risposte, il medico deve salvare la domanda e poi sarà reindirizzato alla schermata precedente per l'aggiunta di una nuova domanda, ripetendo il processo fino al completamento del questionario. Ciascuna domanda e risposta può essere modificata in qualsiasi momento. Prima della conclusione e dell'effettiva creazione del questionario, è possibile visualizzare un'anteprima: il medico può così verificare l'accuratezza delle informazioni inserite, apportare eventuali modifiche e confermare solo quando si ritiene soddisfatto del risultato (**Figura 16**).

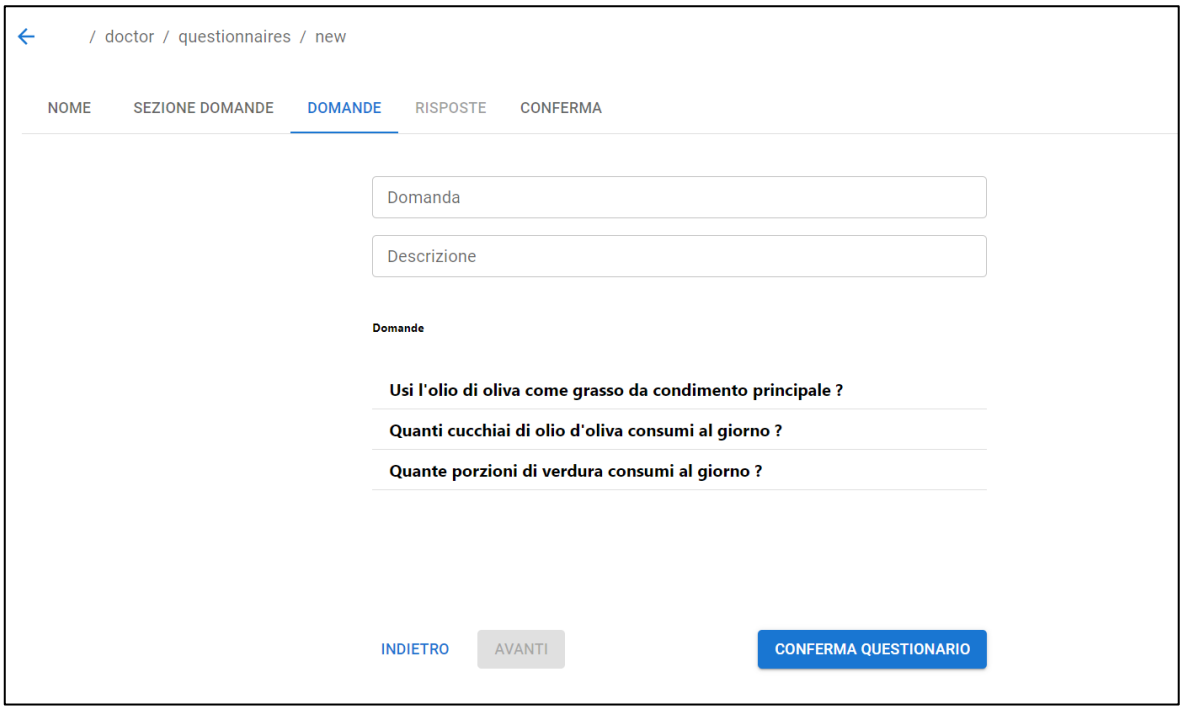

Figura 14 – Esempio di inserimento di una domanda e una sua descrizione.

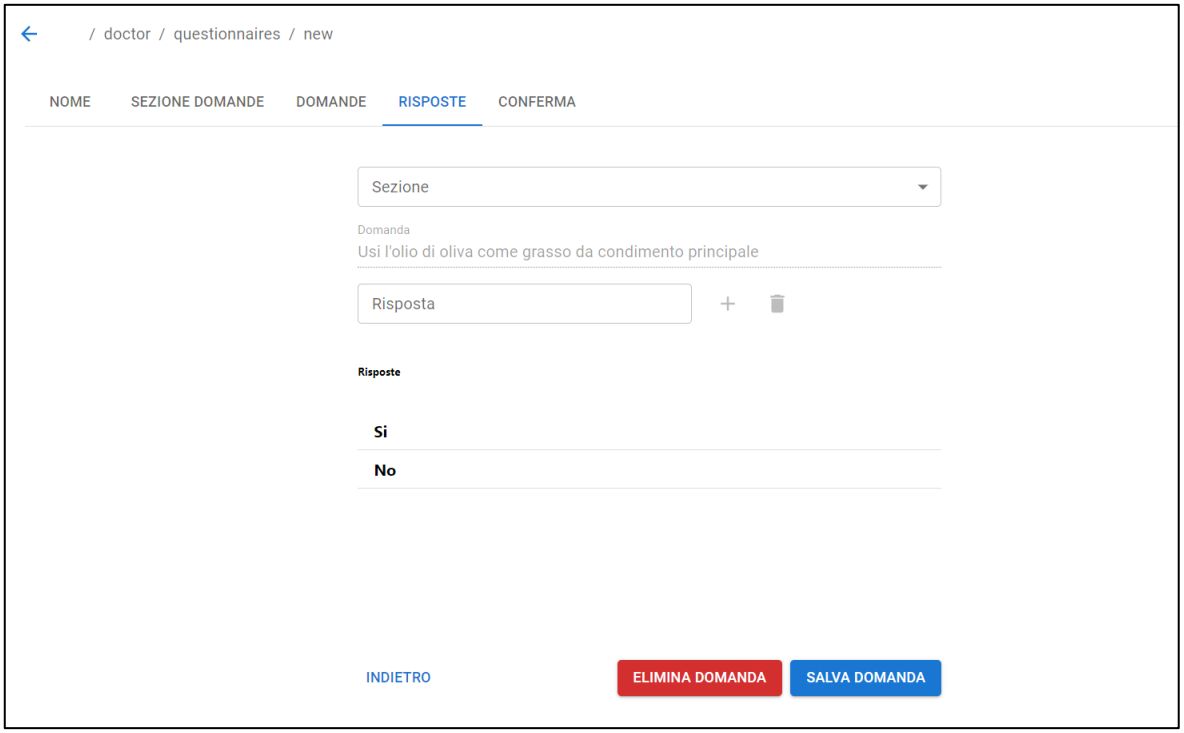

Figura 15 – Esempio di inserimento delle risposte ad una domanda.

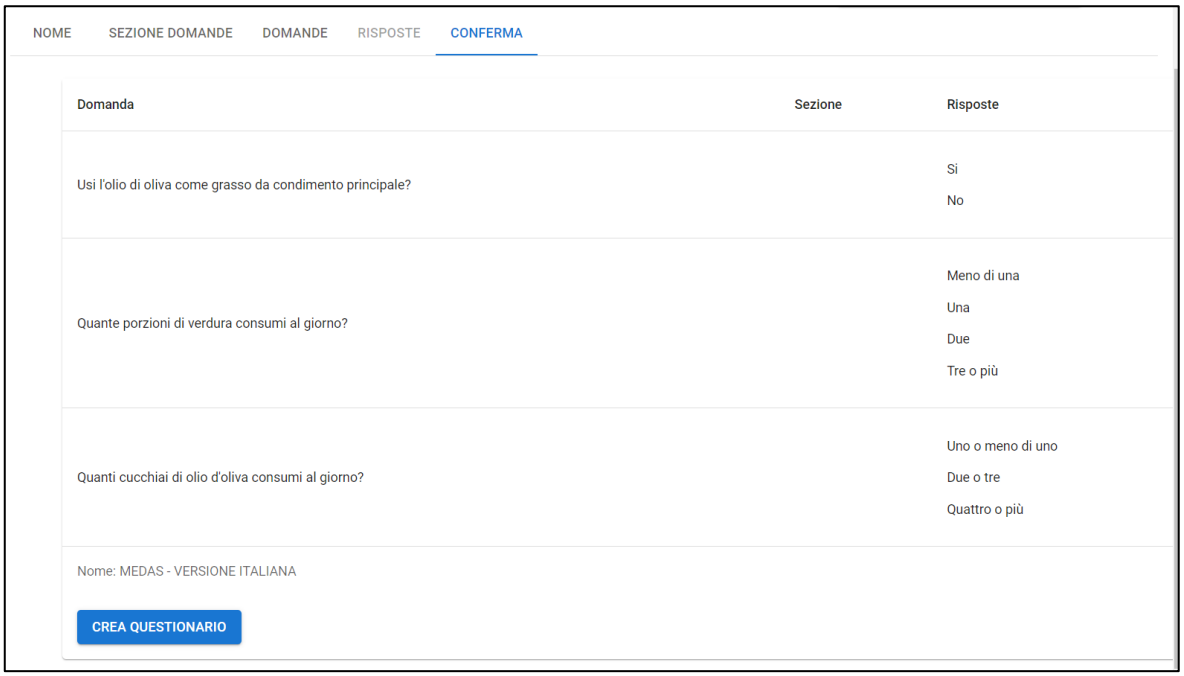

Figura 16 – Esempio di anteprima della creazione di un questionario.

1.3 Compilazione dei questionari

La compilazione dei questionari rappresenta un elemento cruciale nel contesto dell'esperimento, e si è scelto di integrarla direttamente all'interno del client web, oltre che nell'applicazione mobile. Questa decisione deriva dalla natura interattiva di questa specifica attività: la compilazione dei questionari, infatti, avviene solitamente nel corso degli incontri tra medico e paziente, e richiede una comunicazione e una comprensione reciproca che sarebbe difficile replicare in un contesto individuale tramite lo smartphone. La scelta di ricorrere ai questionari nasce dall'indispensabilità di acquisire dati qualitativi che non possono essere rilevati da un dispositivo tracker come Fitbit. Nonostante l'importanza dei dati biometrici che un tale dispositivo può fornire, esistono aspetti del benessere e delle abitudini dei pazienti che necessitano di un'indagine più approfondita e qualitativa, realizzabile attraverso questionari dedicati. L'obiettivo è stato quello di sviluppare un'interfaccia utente che replicasse il più fedelmente possibile l'esperienza reale di compilazione di un questionario a crocette in formato cartaceo. Tale approccio permette di ridurre la barriera tra utente e sistema, facilitando l'adozione e l'uso della piattaforma.

Prima di iniziare la compilazione del questionario, occorre selezionare dalla home page del medico il paziente per cui si vuole effettuare la compilazione del questionario. Successivamente, è necessario selezionare il template appropriato (**Figura 17**). Questi

possono essere standard, derivati dalla letteratura scientifica esistente, oppure personalizzati, creati ad hoc dai medici per rispondere a specifiche necessità di ricerca. La selezione avviene attraverso un campo di multi-selezione, progettato per essere intuitivo e facile da usare. Una volta selezionato il template, il medico può procedere con la compilazione del questionario.

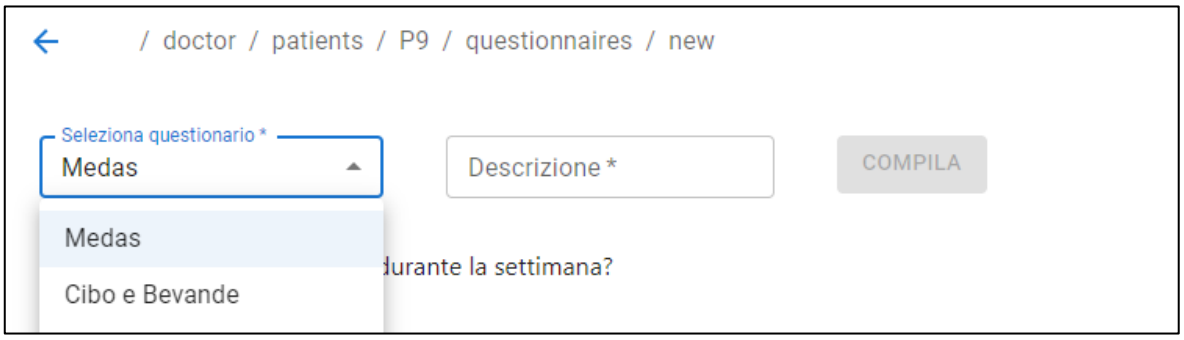

Figura 17 – Campo selezione per la scelta di un questionario.

Il questionario si presenta come un elenco di domande accompagnate da una serie di "checkbox" (**Figura 18**), che l'utente può selezionare in base alle proprie risposte. Attualmente, il sistema richiede che a tutte le domande venga data una e una sola risposta per poter completare il questionario. Tuttavia, è probabile che in futuro si adottino questionari in cui le risposte alle domande potrebbero essere opzionali. Pertanto, si può considerare la possibilità di rendere questo aspetto più flessibile nelle future integrazioni di HealthApp, per migliorare l'esperienza dell'utente e facilitare il processo di compilazione.

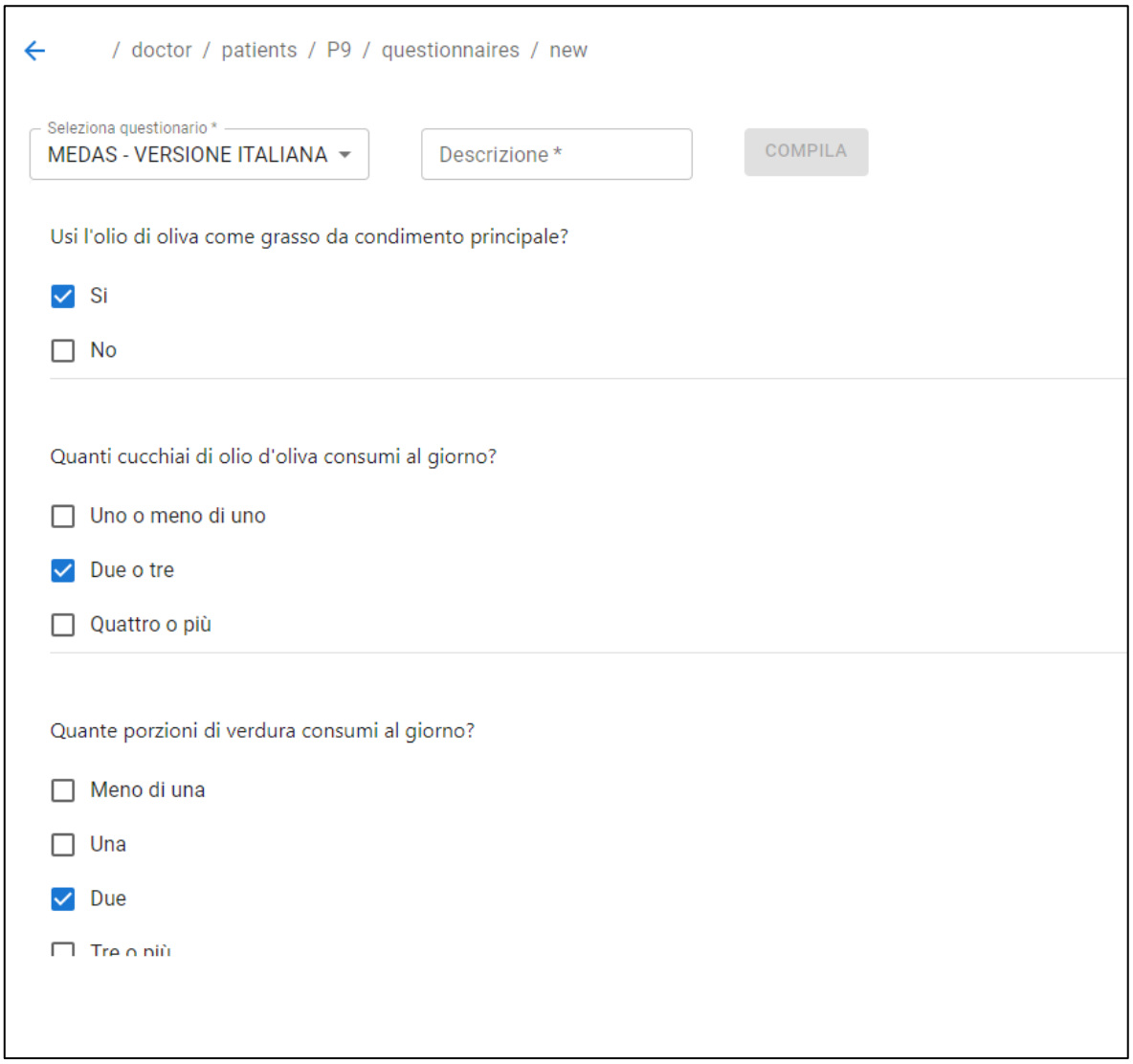

Figura 18 – Esempio di compilazione del questionario MEDAS versione italiana.

Il questionario compilato può essere consultato nella sezione del dettaglio del paziente, permettendo ai medici di avere una panoramica completa dei dati raccolti (**Figura 19, 20**). Un aspetto particolarmente importante in HealthApp è la possibilità di compilare lo stesso questionario più volte. Questa caratteristica permette ai medici di monitorare e valutare la costanza e l'impegno dei pazienti nel tempo, fornendo una visione dinamica del loro progresso e del loro percorso verso il benessere.

Figura 19 – Elenco questionari compilati nel dettaglio paziente.

Figura 20 - Estratto questionario compilato.

1.4 Visualizzazione dei parametri biometrici

La visualizzazione di parametri di salute raccolti dal Fitbit rappresenta un aspetto cruciale per i medici. Questi dati, visualizzabili anche attraverso l'applicazione mobile, permettono di effettuare osservazioni, consultazioni e analisi che facilitano la generazione di consigli volti a migliorare il benessere dei pazienti. Non si tratta di una duplicazione dei dati, ma di una rappresentazione alternativa, ottimizzata per l'utilizzo medico. I dati vengono visualizzati attraverso una finestra temporale, regolabile su una base giornaliera o settimanale (ultimi sette giorni) (**Figura 21**). L'utente medico può muoversi avanti o indietro di un giorno o di una settimana tramite delle frecce, o selezionare direttamente una data nel

calendario. Ogni volta che l'intervallo temporale viene modificato vengono recuperati i dati dal back-end relativi a ciascun parametro biometrico rilevato.

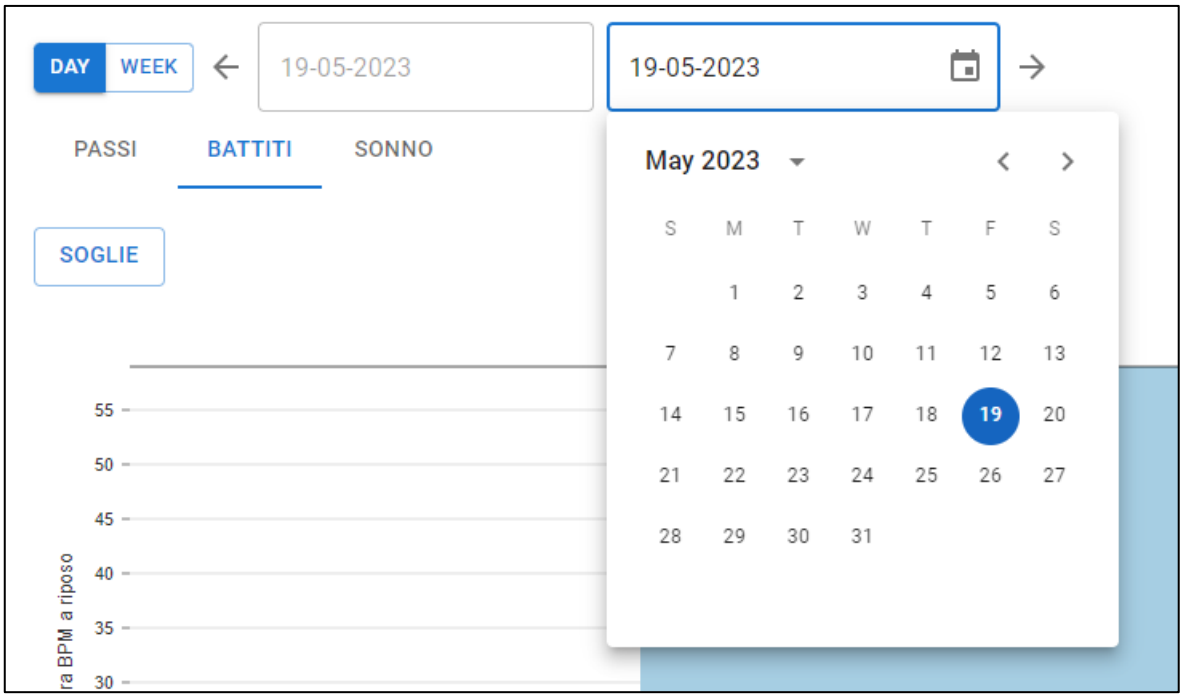

Figura 21 – Componente relativo alla selezione dell'intervallo di tempo.

I dati biometrici sono visualizzati attraverso grafici a barre i cui valori sull'asse delle ordinate rappresentano le misure specifiche relative a sonno, battiti o passi, e sull'asse delle ascisse l'arco temporale (**Figura 22-24**).

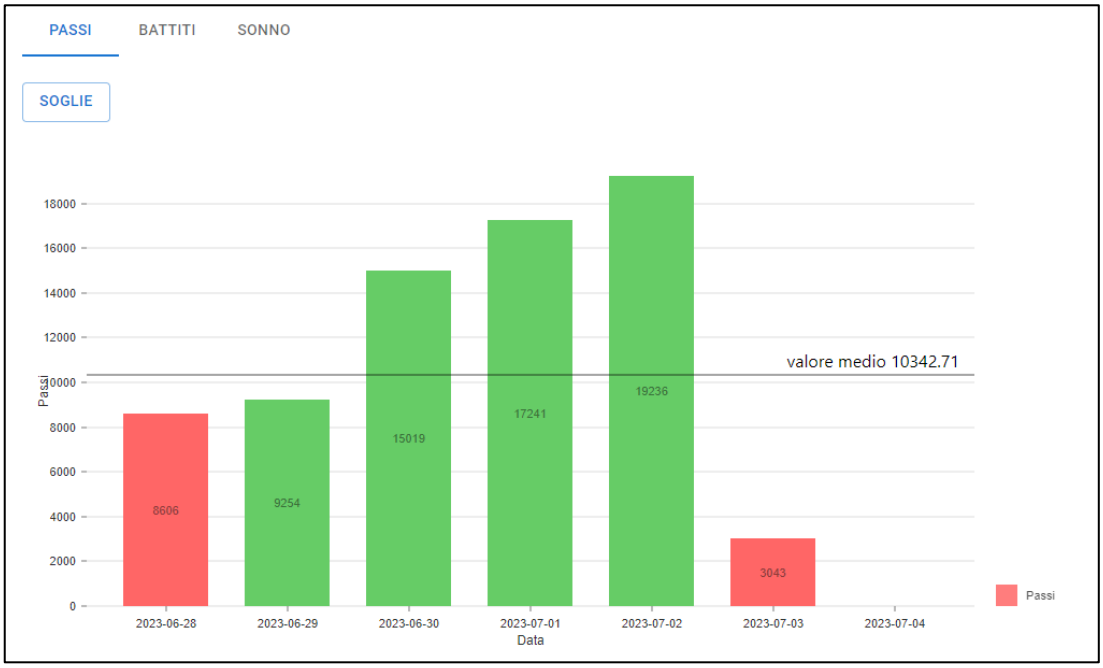

Figura 22 – Esempio di grafico per la visualizzazione dei dati relativi ai passi.

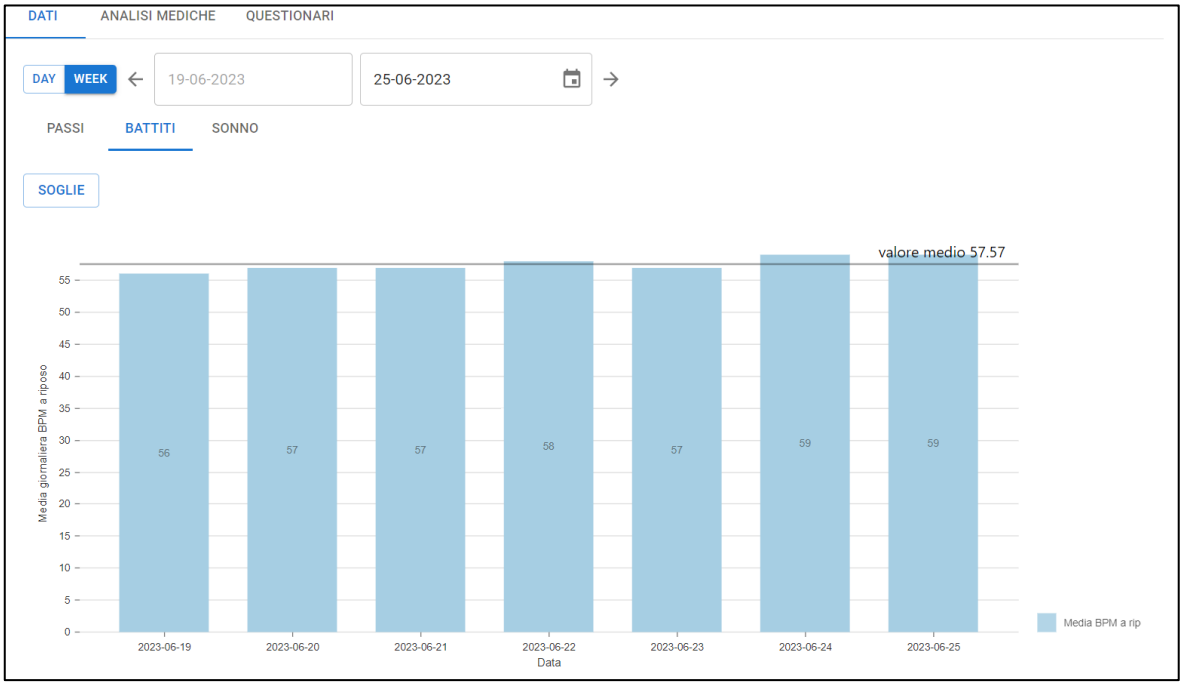

Figura 23 – Esempio di grafico per la visualizzazione dei dati relativi ai battiti medi a riposo.

Figura 24 – Esempio di grafico per la visualizzazione dei dati relativi al sonno.

In particolare, nel caso del monitoraggio di sonno, passi e battiti, la modalità di visualizzazione può essere impostata su “DAY”, con valori relativi al singolo giorno o su “WEEK”, con valori relativi agli ultimi sette giorni. Per ogni misurazione viene riportato un

valore medio, utile al medico per identificare trend o anomalie significative nel periodo selezionato.

Per il monitoraggio dei battiti cardiaci, invece, è stato utilizzato un approccio differente, al fine di fornire un'analisi più dettagliata. In particolare, i battiti cardiaci sono suddivisi in base a tre distinti livelli di attività, corrispondenti a diverse fasce di frequenza cardiaca (**Figura 25**):

1) *Fase di Riposo*: quando i battiti sono inferiori al 60% del valore massimo registrato. Questa fascia rappresenta periodi di bassa attività cardiaca e generalmente associati a momenti di riposo o di attività molto lieve;
2) *Fase di Attività Moderata*: quando i battiti superano il 60% ma sono inferiori all'80% del valore massimo. Questo intervallo indica un'attività cardiaca moderata, tipica di attività fisiche di intensità medio-bassa o situazioni di moderato stress;
3) *Fase di Attività Intensa*: quando i battiti superano l'80% del valore massimo. Questa fascia segnala momenti di intensa attività cardiaca, come durante l'esercizio fisico intenso o situazioni di stress acuto.

In questo caso nei grafici viene riportata la durata in minuti nell'arco di una giornata in cui il valore dei battiti cardiaci corrisponde a una delle tre fasce.

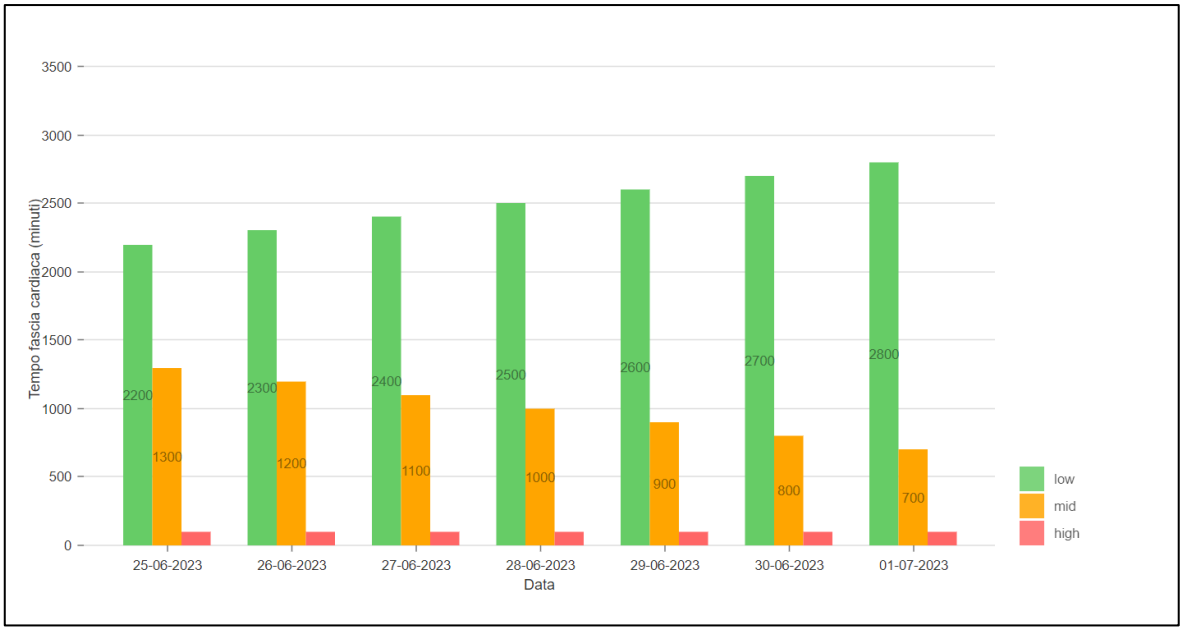

Figura 25 – Grafico delle fasce di frequenza cardiaca.

Pur presentando un metodo di rappresentazione leggermente differente per i battiti cardiaci, il design e l'usabilità rimangono coerenti con la visualizzazione dei restanti dati biometrici. Questo permette al medico di avere una visione più completa dell'attività cardiaca del paziente, senza compromettere l'intuitività e la facilità di interpretazione dell'interfaccia.

1.5 Inserimento e consultazione delle analisi del sangue

Durante gli incontri con i medici, è stata valutata la possibilità di permettere loro di inserire i risultati delle analisi mediche, in particolare del sangue, per conto dei pazienti. Le analisi del sangue rappresentano una risorsa aggiuntiva di dati per il medico, fondamentale per calibrare le decisioni più adeguate al fine di promuovere il miglioramento dello stile di vita e quindi del benessere del paziente. Questo potrebbe avvenire durante una visita di routine con i pazienti coinvolti nell'esperimento, consentendo così al medico di consultare e inserire contestualmente le analisi del sangue nel sistema. L'inserimento dei dati relativi alle analisi mediche nel client web avviene in maniera semplice e diretta: mediante una finestra di "dialog" [19] (**Figura 26**) vengono visualizzati diversi campi di testo, ciascuno corrispondente a un parametro delle analisi del sangue. È previsto che il medico effettui questa operazione in presenza del paziente, il quale avrà con sé il referto delle analisi del sangue necessarie per l'inserimento tramite HealthApp. Poiché gli esami in questione sono stati effettuati in passato, il medico deve selezionare la data tramite il componente del calendario posto nella parte superiore della pagina e poi procedere con l'inserimento. In questo caso d'uso la modalità "WEEK" viene disabilitata (**Figura 27**) e la data selezionata dal componente del calendario corrisponde alla data presunta delle analisi mediche.

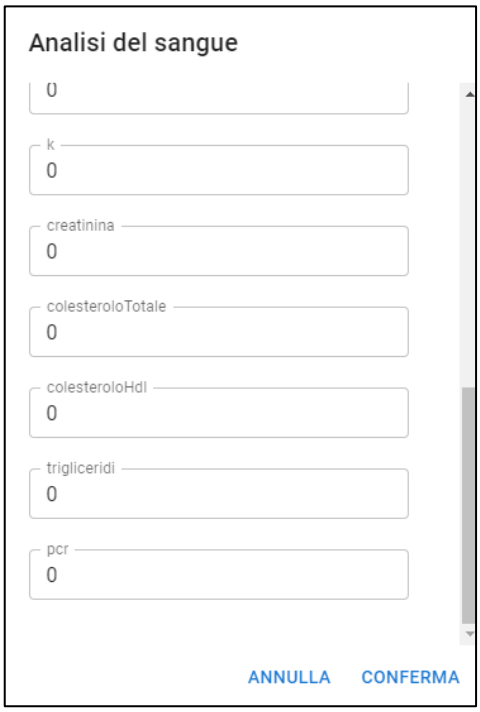

Figura 26 – Finestra per l'inserimento dei valori dell'analisi del sangue.

Figura 27 – Modalità visualizzazione "WEEK" disabilitata nella sezione Analisi del sangue.

Per quanto riguarda la visualizzazione dei dati, questa è destinata esclusivamente al medico. Al fine di facilitare la consultazione, per ogni parametro dell'analisi (**Figura 28**), che generalmente viene recuperato da un'apposita API dal back-end, vengono riportati anche i valori di riferimento, nel formato minimo e massimo.

Tipologia campione-esame	Risultato	Unità di misura	Valore di riferimento minimo
Eritrociti	0	x 10^12/L	4
Emoglobina	0	g/dL	12
MCV	0	fL	80

Figura 28 – Estratto della tabella contente i valori delle analisi del sangue.

Come accennato in precedenza la visualizzazione delle analisi del sangue avviene con la modalità "DAY" in quanto è previsto che il paziente effettui al più una volta in un giorno i prelievi del sangue. La resa grafica del referto è realizzata tramite una tabella che occupa l'intera pagina. Se il medico intende consultare altri esami clinici meno recenti e presenti su HealthApp può aprire il calendario e notare la presenza o meno di pallini rossi posti su un giorno del mese. In questo modo si facilita la ricerca da parte del medico e si velocizza il recupero dei dati (**Figura 29**).

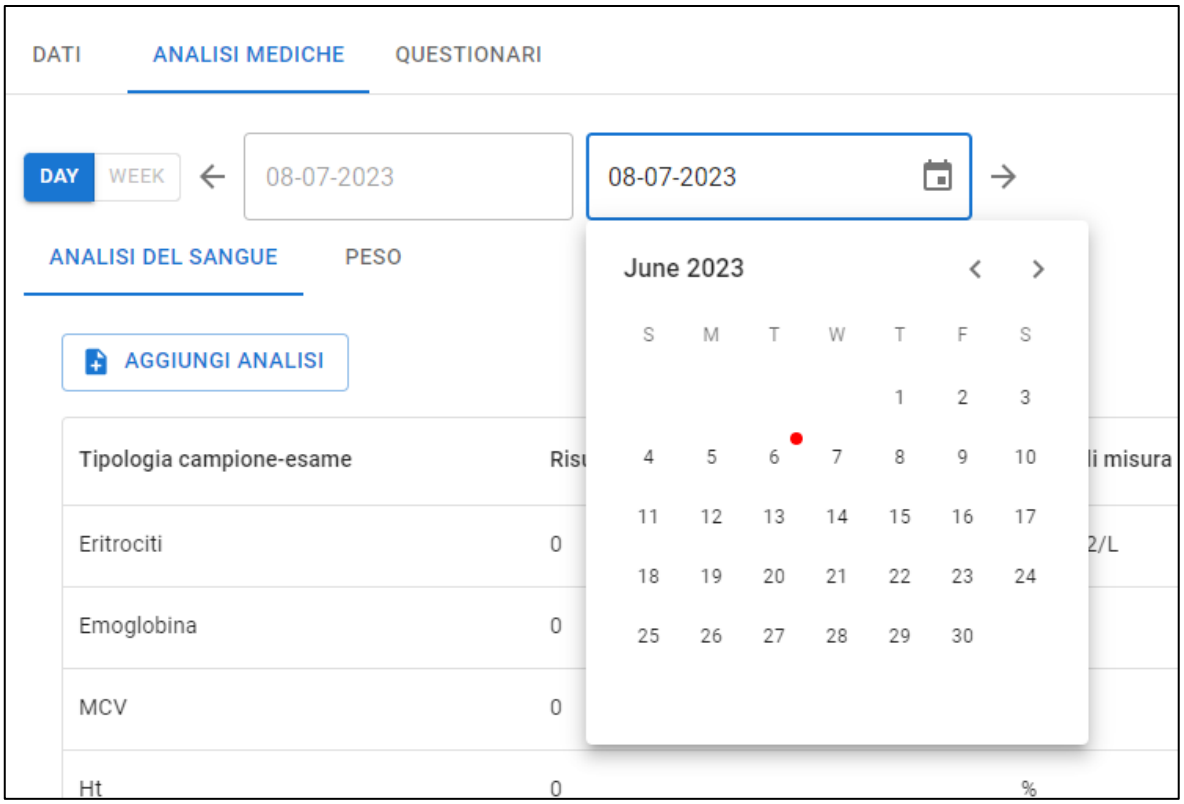

Figura 29 – Esempio di visualizzazione delle date in cui sono state effettuate le analisi del sangue.

1.6 Inserimento e consultazione del peso corporeo

L'inclusione della funzionalità per l'inserimento del peso è una caratteristica disponibile sia nell'interfaccia mobile che nel client web. Quest'ultimo è stato considerato di particolare utilità anche nell'applicazione web poiché, durante le visite di routine, i medici possono misurare il peso dei pazienti e registrare direttamente nel sistema il valore ottenuto. Per questo tipo di operazione, non è necessario selezionare una data specifica. Infatti, il servizio fornito dal back-end permette solo di inserire la nuova misurazione del peso, mentre la data, corrispondente al giorno corrente, è gestita automaticamente dalla logica del back-end. Per quanto concerne il processo di inserimento, il medico deve prima selezionare il paziente dalla sua home e poi navigare la sezione "Analisi mediche" e poi dirigersi nella sottosezione "Peso". Dopo aver cliccato sul pulsante "aggiungi misura" si apre una finestra simile a quella utilizzata per le analisi mediche (**Figura 30**). Questa interfaccia presenta un singolo campo di testo. L'unità di misura, in questo caso i chilogrammi (Kg), è chiaramente indicata. Dopo l'inserimento del dato, basta confermare la modifica. Durante l'interazione con il back-end, simile a quanto avviene per le analisi mediche, verrà mostrato uno stato di caricamento. Al suo completamento, sarà possibile visualizzare il risultato: in caso di errore, un messaggio di errore sarà mostrato in alto al centro della pagina. In caso di successo, invece, l'utente sarà reindirizzato alla schermata che mostra il grafico di andamento del peso (**Figura 31**).

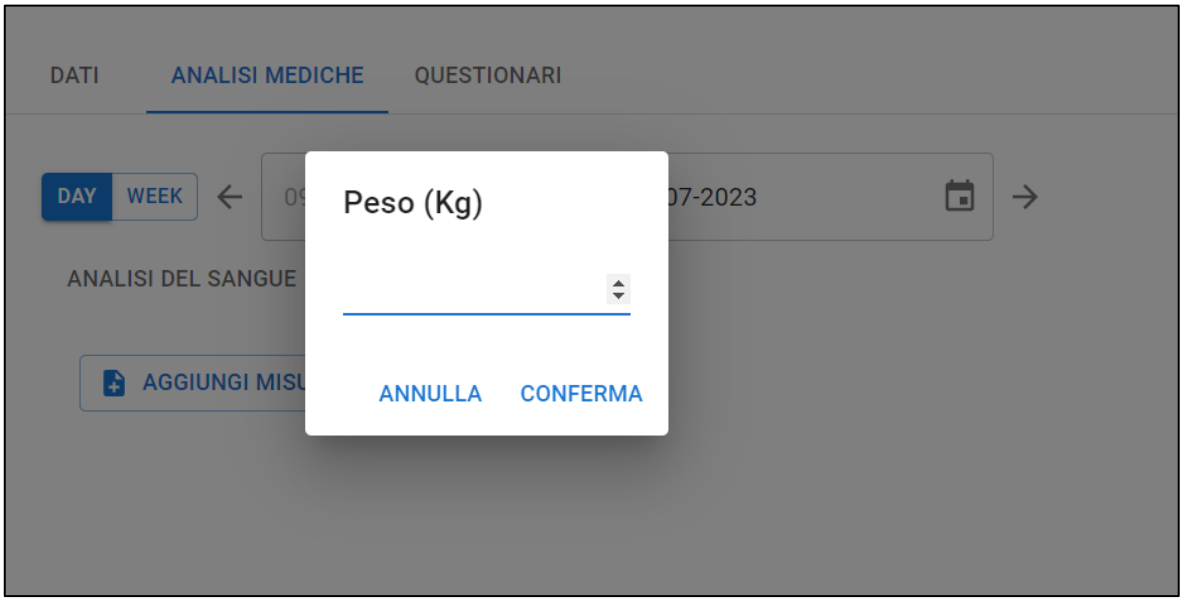

Figura 30 – Finestra per l'inserimento del peso.

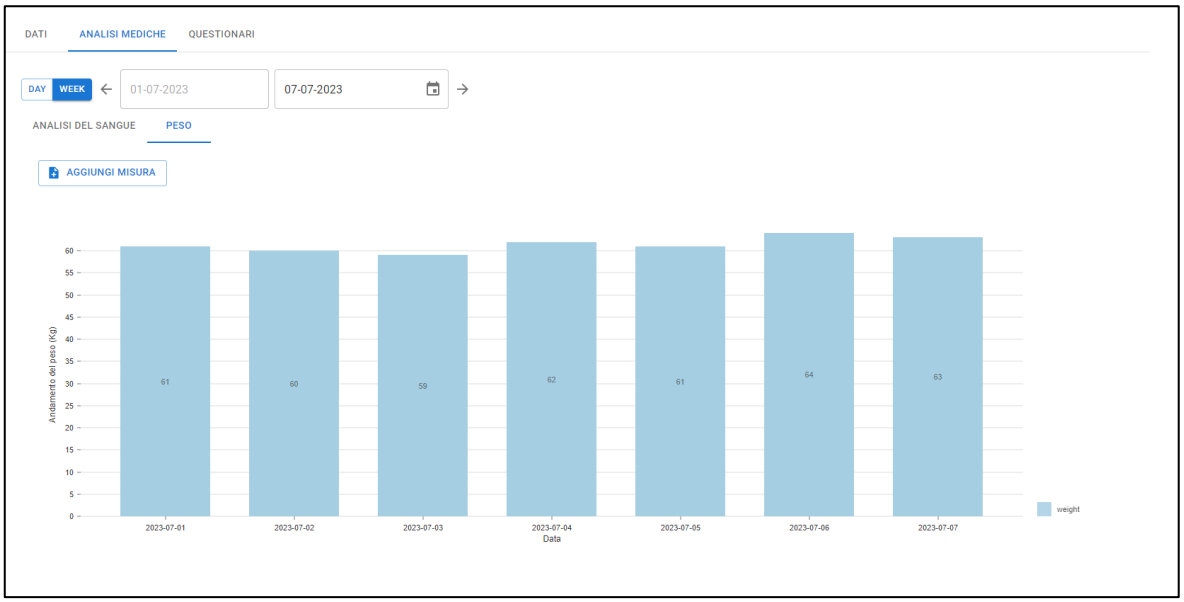

Figura 31 – Grafico andamento del peso.

1.7 Inserimento dei valori soglia

L'operazione chiave per il successo dell'esperimento è la creazione di consigli personalizzati per ciascun paziente, basati su valori soglia appropriati per diverse categorie di dati quali biometrici e derivanti dalle analisi mediche. Queste informazioni vengono poi sfruttate dal “recommender system” integrato nel back-end. Questo rule engine applica specifiche regole ai dati raccolti, permettendo così di definire lo stato di salute del paziente. L'interfaccia, progettata per essere intuitiva e user-friendly, guida il medico nell'inserimento dei valori soglia. Il processo inizia con la selezione del paziente dalla homepage del medico, seguita dalla navigazione verso la sezione "Dati" o “Analisi mediche”. Nella sezione “Dati”, è possibile accedere a tre categorie di dati biometrici: passi, battiti cardiaci e sonno. In questa sezione, oltre alla visualizzazione dei grafici, è possibile impostare valori soglia personalizzati per il paziente selezionato cliccando sul pulsante "soglie". Questi valori definiscono il range entro cui dovrebbero oscillare i parametri di salute del paziente. A seconda del tipo di dato, il formato dell'input richiesto può variare (**Figure 32-34**). Per i passi, è richiesto un valore numerico intero superiore a zero. Per i battiti cardiaci, è necessario inserire un intervallo di valori, in formato stringa come suggerito dal "placeholder" nel campo di testo. Per quanto riguarda il sonno, è richiesto l'inserimento della durata in ore. Va notato che l'API corrispondente richiede la durata del sonno in millisecondi. Pertanto, prima di inviare la richiesta, il modulo responsabile della comunicazione con il back-end esegue la conversione in millisecondi.

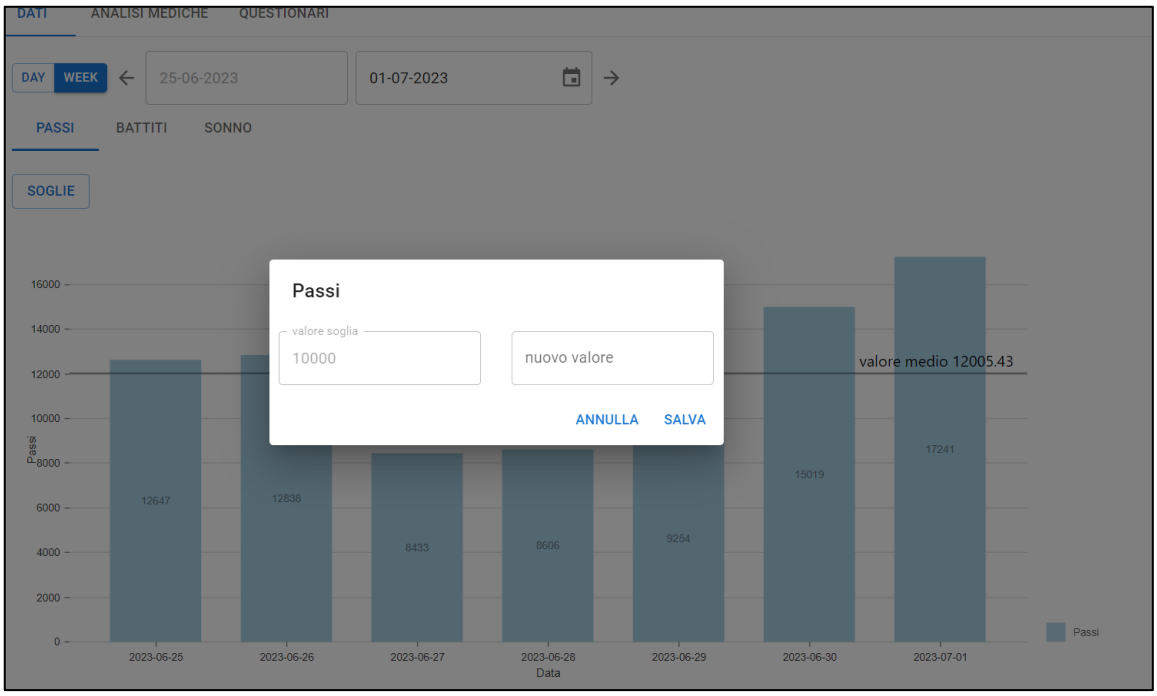

Figura 32 – Finestra per inserimento valore soglia dei passi.

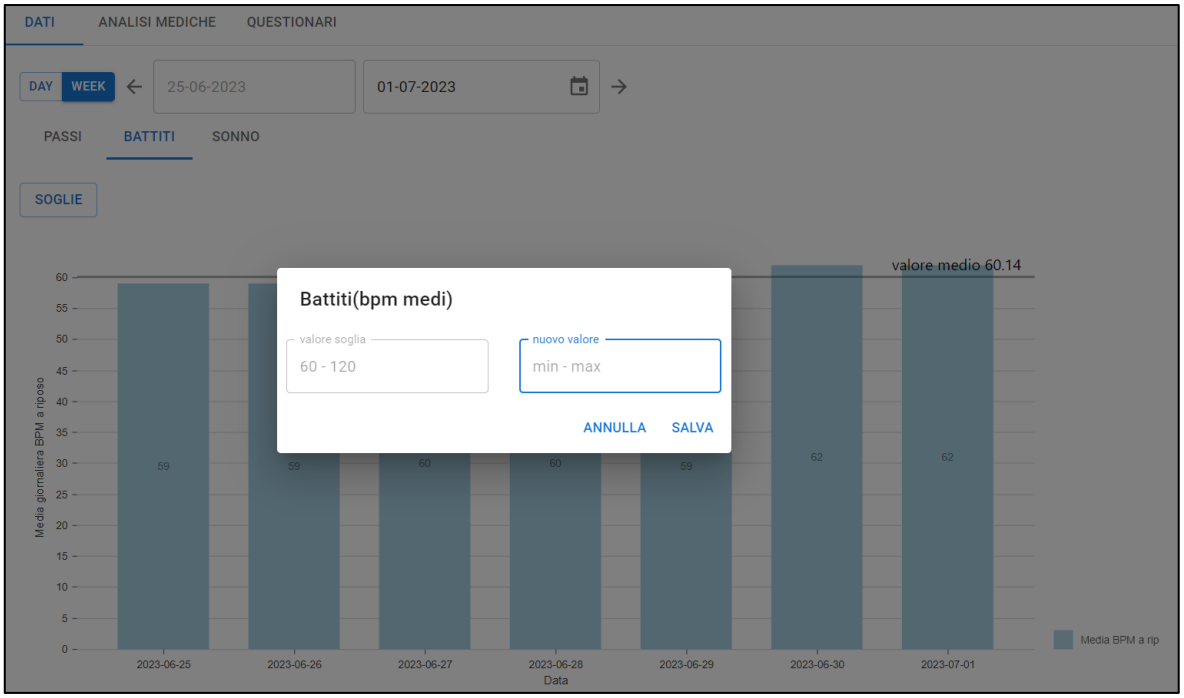

Figura 33 - Finestra per inserimento valore soglia dei battiti.

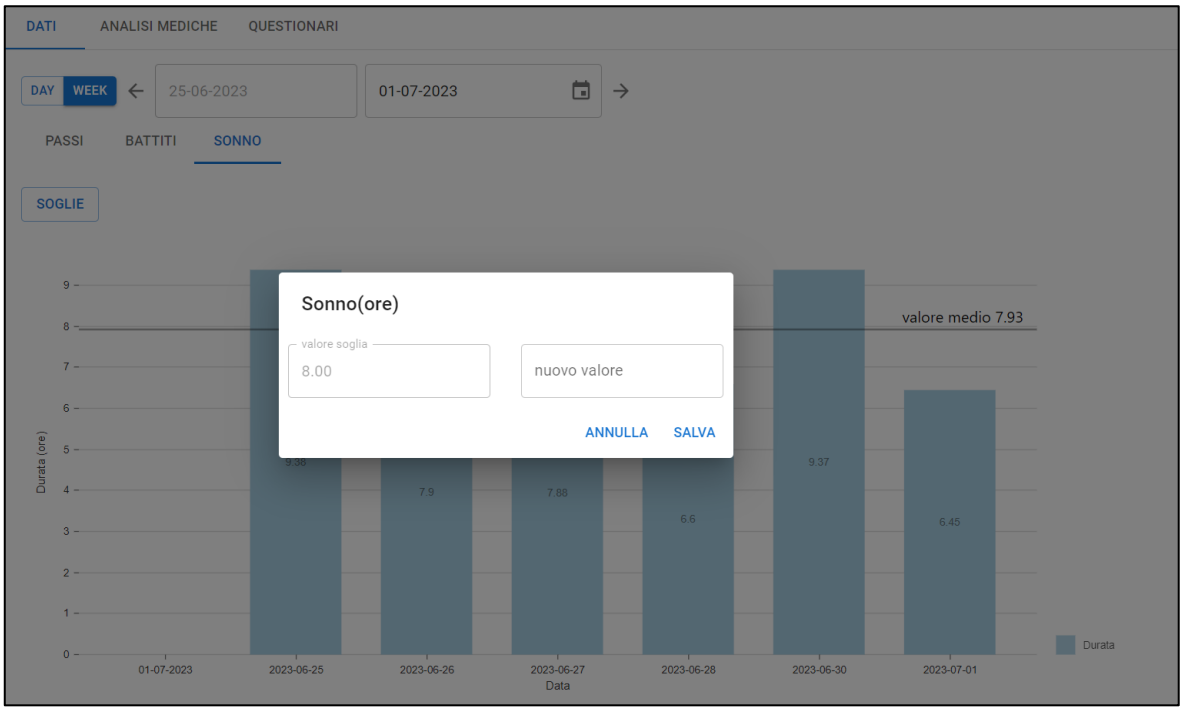

Figura 34 - Finestra per inserimento valore soglia del sonno.

L'implementazione dei valori soglia si estende anche alle analisi mediche. Le regole applicate nel recommender system per valutare lo stato di salute del paziente in questo caso sono differenti, richiedendo l'inserimento di valori sia minimi che massimi. Per specificare questi valori soglia, è necessario navigare alla sezione appropriata e cliccare sul pulsante "soglie". In seguito, si aprirà una finestra di dialogo (**Figura 35)** simile a quella utilizzata per l'inserimento delle analisi mediche. Tuttavia, in questa finestra appariranno due campi di testo per ciascun parametro, indicando rispettivamente il valore minimo e massimo.

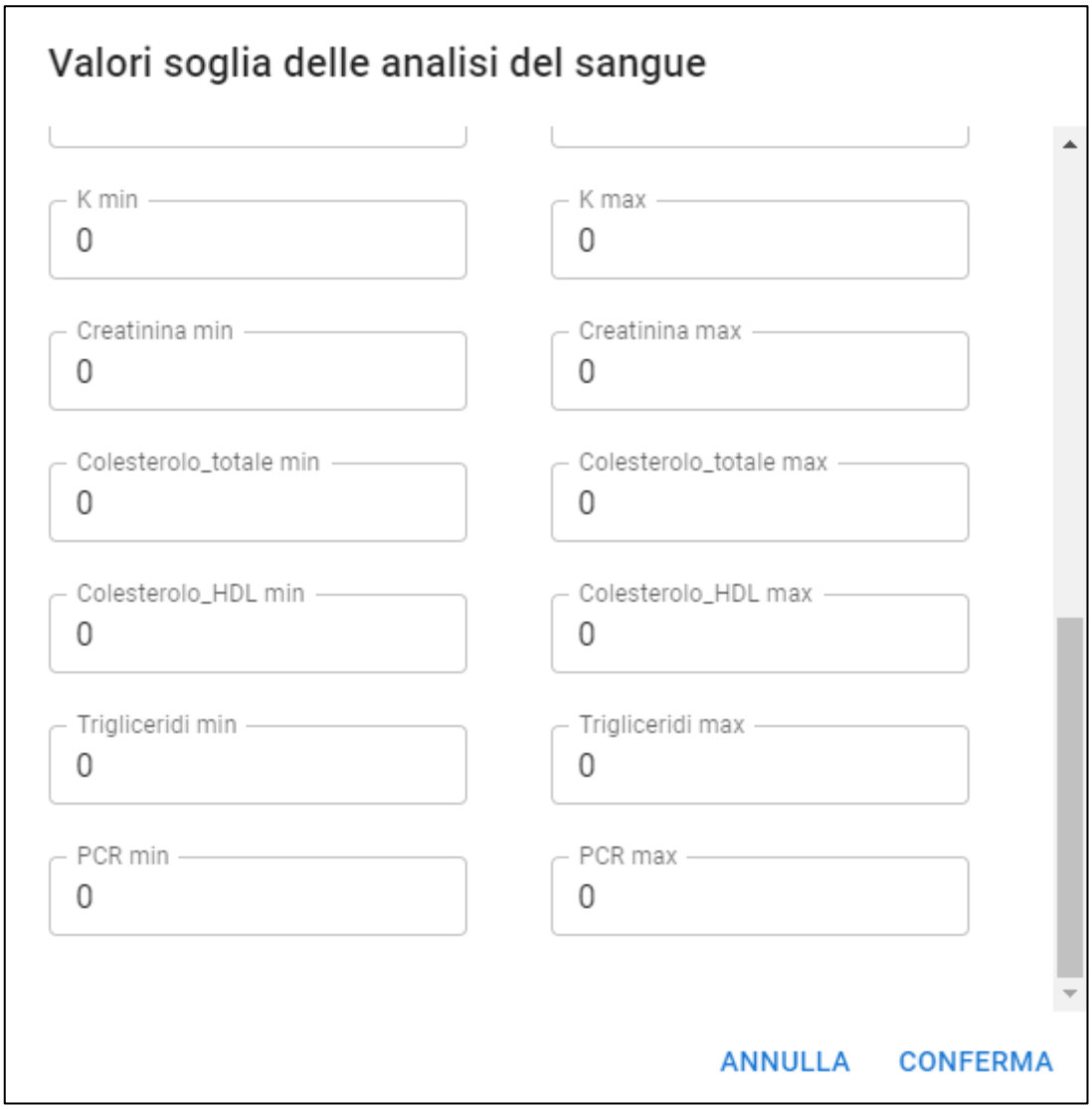

Figura 35 – Finestra per l'inserimento dei valori soglia delle analisi mediche.

Tali valori sono resi disponibili in fase di consultazione delle analisi mediche come descritto nel paragrafo precedente.

III) TECNOLOGIE UTILIZZATE PER LA WEB APPLICATION

Nel seguente capitolo, l'obiettivo è affrontare una discussione esaustiva riguardo alle librerie principali che sono state utilizzate e che sono risultate imprescindibili per la realizzazione del client web quali:

1) Redux: utilizzata per la gestione dello stato dell'applicazione;
2) Material-UI: che ha permesso la creazione di un'interfaccia utente intuitiva ed esteticamente piacevole;
3) Axios: che ha facilitato le operazioni di richiesta HTTP, fondamentali per l'interazione con il back-end dell'applicazione.

1. React Redux

Nel contesto di questa applicazione è stato utilizzato un approccio differente per la gestione dello stato, ricorrendo a Redux, un gestore di stato esterno. Questo strumento fornisce un "stato globale" che avvolge l'intera applicazione, piuttosto che limitarsi a gestire lo stato a livello di singolo componente. Questo approccio offre vantaggi significativi in termini di prevedibilità e tracciabilità dello stato dell'applicazione, consentendo una migliore gestione dell'interazione tra componenti e il flusso dei dati all'interno dell'applicazione. L'utente interagisce con l'interfaccia utente ed esegue operazioni che modificano lo stato che nel contesto di Redux vengono chiamate "actions". Le azioni sono poi intercettate dai "reducers" che implementano la logica di come lo store deve essere aggiornato. Successivamente l'interfaccia riflette le informazioni presenti nello store e il processo si ripete (**Figura 36**).

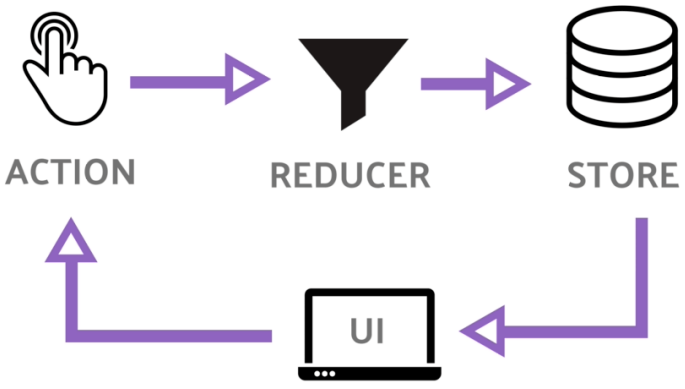

Figura 36 – Schema del funzionamento di Redux.

1.1 Store

Uno dei componenti chiave di Redux è lo store, cioè un oggetto che ospita l'intero stato dell'applicazione [20]. Lo store è unico nel contesto di un'applicazione React Redux. In generale questo viene associato ad un unico stato globale di root, ma la libreria Redux permette di scomporlo e assegnare a più reducers la sua gestione, agendo su porzioni dello stato di root. Questa gestione è particolarmente adatta nel contesto di applicazioni di grandi dimensioni. Nel caso di HealthApp, essendo un'applicazione più semplice in termini di dimensioni, si è deciso di utilizzare un singolo stato e una sola funzione reducer.

Lo store è definito utilizzando la funzione "createStore" che accetta come parametri la funzione principale "reducer" e un "middleware" (**Figura 37**) [21]. Generalmente, la logica di creazione risiede in un file JS e nel caso di HealthApp è stato scritto il codice in un file chiamato store.js. Ciò permette di isolarlo rispetto alla gestione dei reducers e delle azioni al fine di mantenere un'organizzazione modulare e pulita.

```
import { createStore, applyMiddleware } from "redux";
import thunk from "redux-thunk";
import rootReducer from "./reducers";

const store = createStore(rootReducer, applyMiddleware(thunk));
export default store;
```

Figura 37 – Codice di creazione dello store in HealthApp.

1.2 Middleware

Il "middleware" in Redux è un potente meccanismo che consente di avvolgere il processo di "dispatch" dell'azione, fornendo un punto in cui si possono inserire operazioni aggiuntive come l'esecuzione di funzioni asincrone, quali chiamate verso il back-end, operazioni di log e molto altro [22]. Esistono vari middleware che si integrano in Redux e possono essere personalizzati o di terze parti.

Nel contesto di HealthApp è stato utilizzato il middleware di terze parti react-thunk [23]. Questo permette di scrivere azioni Redux come funzioni invece di oggetti puri che possono eseguire ulteriori logiche, oltre a specificare il tipo di azione (**Figura 38**). Nel seguente

esempio “getHearthThreshold” è un’azione definita come funzione thunk [23] ed è responsabile della ricezione dei dati relativi ai valori soglia dei battiti cardiaci del paziente identificato dal valore del parametro “patientId”. Prima che l’azione “HR_THRESHOLD” venga eseguita e gestita dal “reducer” competente, si attende che la promise associata alla chiamata “patientService.getHearthThreshold” venga risolta.

```
export const getHearthThreshold = (patientId) => (dispatch) => {
    patientService.getHearthThreshold(patientId).then(result =>
      ...
      dispatch({
        type : "HR_THRESHOLD",
        payload : result.data
      })
    },
    ...
  }
```

Figura 38 – Esempio di chiamata REST nel contesto di un’azione redux in HealthApp.

1.3 Reducers

In Redux, i reducers sono funzioni che prendono lo stato corrente dell'applicazione e un'azione, e restituiscono un nuovo stato. Questi interpretano l'azione e decidono come l'applicazione dovrebbe cambiare in risposta ad essa [24]. Nel contesto di un’applicazione React-Redux, i reducers possono essere molteplici e agire su singole porzioni dello stato globale associato allo store redux, pertanto vengono definiti “slice Reducers” [25].

Questo pattern è noto come “reducer composition" e risulta efficace per applicazioni complesse in cui occorre suddividere lo stato globale di Redux in parti più piccole [25] (**Figura 39**).

```
export default function todosReducer(state = initialState, action) {
  switch (action.type) {
    case 'todos/todoAdded': {
      // Can return just the new todos array - no extra object around it
      return [
        ...state,
        {
          id: nextTodoId(state),
          text: action.payload,
          completed: false
        }
      ]
    }
    case 'todos/todoToggled': {
      return state.map(todo => {
        if (todo.id !== action.payload) {
          return todo
        }

        return {
          ...todo,
          completed: !todo.completed
        }
      })
    }
    default:
      return state
  }
}
```

Figura 39 – Esempio di slice reducer, *todosSlice.js* [26].

Tuttavia, in fase di creazione dello store Redux, la funzione “reducer” che viene usata come parametro è una sola; pertanto, si parla di funzione “root reducer”. Per combinare quindi più slice reducers in un’unica funzione, si utilizza “combineReducers” che accetta come parametro un oggetto JS in cui sono mappate le singole funzioni reducers [20] (**Figura 40**).

```
rootReducer = combineReducers({potato: potatoReducer, tomato: tomatoReducer})
// This would produce the following state object
{
  potato: {
    // ... potatoes, and other state managed by the potatoReducer ...
  },
  tomato: {
    // ... tomatoes, and other state managed by the tomatoReducer, maybe some
nice sauce? ...
  }
}
```

Figura 40 – Esempio di combineReducers [27].

Un pattern comune nell'implementazione dei reducers è l'uso di un'istruzione switch-case per gestire diverse tipologie di azioni [28]. Ciascun case fa riferimento ad un’azione specifica che è caratterizzata da una stringa e descrive come deve essere aggiornato lo stato. Per l’aggiornamento dello stato si utilizza lo spread operator di Javascript che consente di creare una copia dell'oggetto e aggiornare solo le proprietà specifiche che si desidera modificare [29].

Nel caso di HealthApp, come accennato in precedenza, si è utilizzata un’unica funzione reducer che gestisce tutte le azioni volte a modificare lo store globale (**Figura 41**)

```
export default function reducer (state = initialState, action) {
  const {
    type,
    payload,
  } = action;
  switch (type) {
    case 'HR_THRESHOLD':
      return {
        ...state,
        hrThreshold : payload
      }
    case 'PATIENT_ID' :
      return {
        ...state,
        patientId:  payload
      }
    case 'DOCTORS':
      return {
        ...state,
        doctorsList: payload
      };
```

Figura 41 – Estratto della funzione reducer usata in HealthApp.

1.4 Azioni

In Redux, le azioni sono rappresentate come oggetti che incorporano un “type” ed eventuali dati supplementari. Il campo "type" nell’oggetto azione, stabilisce il tipo di azione da intraprendere, mentre le altre proprietà dell'oggetto possono servire a trasportare dati aggiuntivi legati all' azione [30]. Questi oggetti vengono poi trasmessi allo store tramite il metodo “dispatch”.

Nel contesto di HealthApp, si è scelto di utilizzare il campo "payload" per conservare il valore che verrà integrato nello specifico campo dello stato a cui l'azione si riferisce (**Figura 42**).

```
export const getPatients = () =>(dispatch) => {
      authServices.getPatients().then(
        (data) =>{
          debugger;
          try{
            var json = JSON.parse(JSON.stringify(data.data));
            if(!json  || typeof(json) !== 'object'){
              dispatch({
                type: "LOGOUT",
                payload: null
              })
              return;
            }
          }
          catch(e){
            dispatch({
              type: "LOGOUT",
              payload: null
            })
          }
          dispatch({
            type:"PATIENTS",
            payload : data.data
          })
        },
        (error)=>{
          dispatch({type:"ERROR", payload : error.message})
        }
      )
}
```

Figura 42 – Esempio di azione tramite funzione per recuperare la lista dei pazienti in HealthApp.

Come menzionato in precedenza, le azioni oltre ad essere oggetti puri di JavaScript, possono essere trattati come funzioni.

In HealthApp, si è optato per raggruppare le azioni associate alla raccolta dei dati attraverso le REST API come funzioni e, una volta risolta la promise legata alla chiamata HTTP si esegue il "dispatch" dell'azione sotto forma di oggetto [31] (**Figura 42**). In questo caso, nel campo "payload", viene salvato il risultato della chiamata che sarà successivamente gestito dal reducer competente che aggiornerà il campo dello stato legato alla lista dei pazienti. Successivamente, il componente dell'interfaccia utente che si occupa di mostrare l'elenco dei pazienti sarà nuovamente ricaricato ed aggiornato (**Figura 43-45**).

```
const { patientsList,isLoggedIn } = useSelector(
    (state) => state.auth
  );
```

Figura 43 – Recupero dei dati immagazzinati nello stato globale da parte del componente che visualizza la lista dei pazienti in HealthApp.

```
<DataGrid
                  rows={patientsList}
                  columns={columnsNew}
                  pageSize={10}
                  loading={patientsListLoading === true}
                  initialState={{

                    pagination: { paginationModel: { pageSize: 10 } },
                  }}
                  pageSizeOptions={[10, 20, 50]}
                  disableRowSelectionOnClick

                />
```

Figura 44 – Estratto del codice relativo al componente che mantiene la lista dei pazienti.

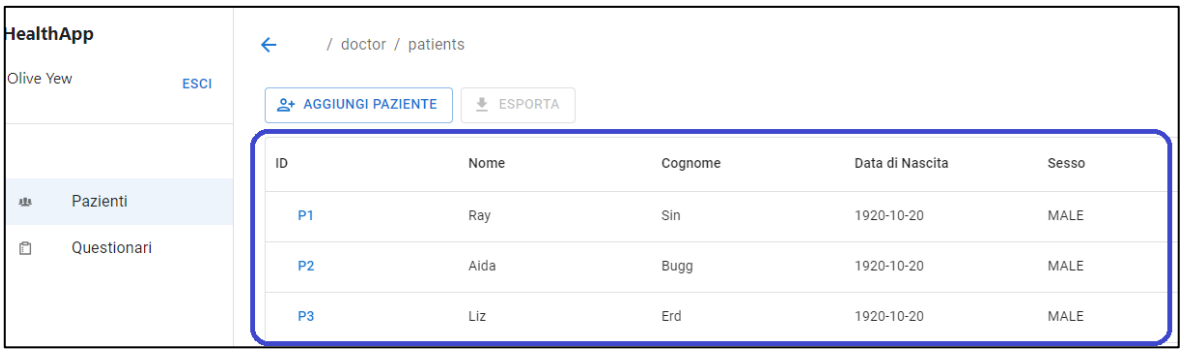

Figura 45 – Resa grafica del componente che mostra l'elenco dei pazienti (dati fittizi).

Nell'applicazione HealthApp, è stato adottato uno schema organizzativo dei file JavaScript per gestire le azioni. Ogni tipo di risorsa su cui occorre effettuare operazioni di modifica o lettura ha un file dedicato che ne definisce ed esporta le funzioni. Ad esempio, per gestire le operazioni relative ai pazienti e tutto ciò che è correlato a loro, è stato creato il file "patient.js" nella cartella "actions" del progetto. Se occorre creare un nuovo set di azioni per eseguire operazioni su una particolare risorsa, si dovrà creare un file nel formato "<nome risorsa>.js". Questo schema organizzativo consente di mantenere una struttura chiara e coerente nel codice, facilitando la gestione delle operazioni specifiche per ciascuna risorsa.

Nel contesto di HealthApp le azioni utilizzate sono:

1) `getPatientWeights(patientid)`: richiede tramite axios i dati al back-end relativi alle misure del peso del paziente identificato dal parametro "patientId";
2) `compileQuestionnaire(patientId,questionnaire)`: invia tramite axios i dati relativi al questionario immagazzinati nell'oggetto "questionnaire" riferiti al paziente identificato dal parametro "patientId";
3) `getPatientQuestionnaires(patientId)`: richiede tramite axios la lista dei questionari compilati per il pazient identificato dal parametro "patientId";
4) `getPatientBloodAnalysis(patientId)`: richiede tramite axios la lista delle analisi del sangue riferite al paziente identiicato dal parametro "patientId";
5) `insertStepsThreshold(patientId,data)`: invia tramite axios i valori soglia dei passi immagazzinati nell'oggetto "data" e riferiti al paziente identificato dal parametro "patientId";

6) *getPatientQuestionnaire(patientId,questionnaireId)*: richiede tramite axios il dettaglio del questionario identificato dal parametro "questionnaireId" riferiti al paziente avente identificativo "patientId";
7) *getPatient(patientId)*: richiede tramite axios il dettaglio anagrafico del paziente identificato dal parametro "patientId";
8) *getPatientHearthRates(patientId,startDate,endDate)*: richiede tramite axios la lista dei dati biometrici dei battiti nell'intervallo temporale avente come data inizio "startDate" e data fine "endDate" e riferiti al paziente identificato dal parametro "patientId";
9) *getPatientSleep*: richiede tramite axios la lista dei dati biometrici del sonno nell'intervallo temporale avente come data inizio "startDate" e data fine "endDate" e riferiti al paziente identificato dal parametro "patientId"
10) *insertPatientWeigh(patientId,data)*: invia tramite axios la misura del peso immagazzinata nell'oggetto data e riferiti al paziente identificato dal parametro "patientId";
11) *insertBloodAnalysis(patientId,data)*: invia tramite axios I dati relativi all'analisi del sangue immagazzinati nell'oggetto "data" e riferiti al paziente identificato dal parametro "patientId";
12) *getPatientStepsThreshold(patientId)*: richiede tramite axios I dati relativi ai valori soglia dei passi riferiti al paziente identificato dal parametro "patientId";
13) *getPatientSteps(patientId,startDate,endDate*: richiede tramite axios la lista dei dati biometrici dei passi nell'intervallo temporale avente come data inizio "startDate" e data fine "endDate" e riferiti al paziente identificato dal parametro "patientId";
14) *getSleepThreshold(patientId)*: richiede tramite axios I dati relativi ai valori soglia del sonno riferiti al paziente identificato dal parametro "patientId";
15) *getHearthThreshold(patientId)*: richiede tramite axios I dati relativi ai valori soglia del sonno riferiti al paziente identificato dal parametro "patientId";
16) *insertHearhThreshold(patientId,data)*: invia tramite axios I valori soglia dei battiti cardiaci immagazzinati nell'oggetto "data" e riferiti al paziente identificato dal parametro "patientId";
17) *insertSleepThreshold(patientId,data)*: invia tramite axios I valori soglia del sonno immagazzinati nell'oggetto "data" e riferiti al paziente identificato dal parametro "patientId";
18) *getDoctors*: richiede tramite axios la lista dei dottori;

19) *getQuestionnaireTemplate(questionnaireId)*: richiede tramite axios il dettaglio del template del questionnario identificato dal parametro "questionnaireId";
20) *getQuestionnairesTemplates*: richiede tramite axios la lista dei template dei questionari;
21) *getPatients*: richiede tramite axios la lista dei pazienti;
22) *userDetail(id)*: richiede tramite axios il dettaglio dell'utente identificato dal parametro "id";
23) *getBloodAnalysisRange(patientId)*: richiede tramite axios la lista dei valori di riferimento delle analisi del sangue specifiche per il paziente identificato dal parametro "patientId";
24) *createQuestionnaire(questionnaire)*: invia tramite axios il dettaglio del template del questionario immagazzinato nell'oggetto "questionnaire";

2. React Material-UI

React Material-UI è la libreria usata per realizzare componenti per l'interfaccia utente basata su le linee guida di Material Design di Google [32]. La libreria contiene un vasto set di componenti predefiniti e pronti per l'utilizzo. L'utilizzo di una libreria per componenti grafici predefiniti e testati rende lo sviluppo e la messa in produzione di un'applicazione in tempi decisamente molto brevi rispetto a scrivere da zero l'html ,css e javascript. Tuttavia, i componenti messi a disposizione dalla libreria React MaterialUI possono essere personalizzati. Ciò permette di personalizzare la propria interfaccia utente e adattarla in base alle proprie esigenze.

Nel contesto di HealthApp, i componenti sono stati utilizzati nella loro forma nativa, senza apportare modifiche alle caratteristiche grafiche, ad eccezione di lievi ritocchi ai margini e alla spaziatura interna per adattarli ai layout delle pagine. Tuttavia, qualora fosse necessario apportare della personalizzazione sfruttando i componenti predefiniti della libreria, è possibile seguire le seguenti strategie:

1) props "style": si possono passare le proprietà di stile come props al componente Material-UI che si vuole modificare [33] (**Figura 46**);

```
import React from 'react';
import Button from '@material-ui/core/Button';

// We can use inline-style
const style = {
  background: 'linear-gradient(45deg, #FE6B8B 30%, #FF8E53 90%)',
  borderRadius: 3,
  border: 0,
  color: 'white',
  height: 48,
  padding: '0 30px',
  boxShadow: '0 3px 5px 2px rgba(255, 105, 135, .3)',
};
export default function InlineStyle() {
  return <Button style={style}>inline-style</Button>;
}
```

Figura 46 – Personalizzazione di un componente Button della libreria MaterialUI in modalità inline-style [33].

2) hook “styled” : questa API di Material-UI utilizza il concetto di hook di React per definire e applicare gli stili [34] (**Figura 47**).

```
import * as React from 'react';
import { styled } from '@mui/styles';
import Button from '@mui/material/Button';

const MyButton = styled(Button)({
  background: 'linear-gradient(45deg, #FE6B8B 30%, #FF8E53 90%)',
  border: 0,
  borderRadius: 3,
  boxShadow: '0 3px 5px 2px rgba(255, 105, 135, .3)',
  color: 'white',
  height: 48,
  padding: '0 30px',
});

export default function StyledComponents() {
  return <MyButton>Styled Components</MyButton>;
}
```

Figura 47 – personalizzazione di un componente Button della libreria MaterialUI tramite Hook API [34].

3. Axios

Axios è una popolare libreria JavaScript che facilita la comunicazione tra client e server attraverso il protocollo HTTP. Essa può essere impiegata in vari contesti, come ad esempio le applicazioni web o gli ambienti server come Node.js [35]. Questa libreria è spesso utilizzata per gestire le chiamate API nelle applicazioni front-end, integrandosi facilmente con i principali framework come React. Axios supporta i quattro principali metodi HTTP: GET, POST, PUT e DELETE. Questa gamma di metodi consente agli sviluppatori di realizzare tutte le operazioni tipiche di un'interazione con un'API REST, come la lettura, la creazione, l'aggiornamento e l'eliminazione dei dati [36]. Inoltre, sfrutta le Promise JavaScript che si rivelano particolarmente utili nella gestione delle risposte del server e che spesso richiedono un tempo di attesa non prevedibile [35].

Nel contesto di HealthApp, Axios è stato utilizzato all'interno delle azioni Redux per interrogare il back-end per richiedere dati o per effettuare delle operazioni di modifica su di essi (**Figure 48, 49**).

```
export const getPatientQuestionnaires = (id) => (dispatch) =>{
    return patientService.getPatientQuestionnaires(id).then(
      (data) =>{

        if(data.data instanceof Array){

          dispatch({
            type: "PATIENT_QUESTIONNAIRES",
            payload: data.data

          })
        }
        else{

          dispatch({
            type: "LOGOUT",

          })
        }


      },
      (err) =>{
        dispatch({
          type: "ERROR",
          payload: err.message
        })
      }
    )
  }
```

Figura 48 – Funzione che descrive l'azione redux che richiama un metodo per la richiesta dei dati al backend al fine di recuperare i questionari di un paziente in HealthApp.

```
const getPatientQuestionnaire = (patientId,questionnaireId) => {
    return axios

.get("/api/patients/"+patientId+"/questionnaires/"+questionnaireId);
}
```

Figura 49 – Funzione che utilizza il modulo axios per eseguire il metodo GET in HealthApp.

La libreria Axios offre inoltre una flessibilità notevole nella configurazione delle richieste, supportando l'invio di dati tramite Form o JSON e l'aggiunta di “headers” personalizzati (**Figura 50**) [37].

```
const getPatientSteps = (id,startDate,endDate) =>{
    return axios
    .get(API_URL.concat("/api/patients/"+id+"/activities/steps"),{ params: {
startDate: startDate,endDate : endDate } })
}
```

Figura 50 – Esempio di utilizzo di una chiamata POST con axios utilizzando i parametri nella request HTTP in HealthApp.

Inoltre, nello sviluppo di HealthApp è stato seguito un approccio modulare nell’organizzazione dei file Javascript per raggruppare le richieste effettuate al back-end in base al tipo di risorsa REST. Questo prevede l’uso di:

1) *Auth.service.js* : per effettuare le chiamate al server relative alle operazioni di login/logout e il dettaglio dell’utente di sessione;
2) *Patient.service.js*: per effettuare le chiamate al server relative alle operazioni di lettura e modifica dei dati collegati ai pazienti quali questionari, parametri biometrici e dettagli anagrafici;
3) *Doctor.service.js*: per effettuare le chiamate al server relative alle operazioni di lettura e modifica dei dati relativi ai pazienti, i template e le compilazioni dei questionari.

IV) IMPLEMENTAZIONE DELLA WEB APPLICATION

1. Architettura del Client web

Per lo sviluppo del client web di HealthApp, sono state utilizzate altre librerie di terze parti, oltre a quelle appena descritte, al fine di supportare al meglio la navigazione e la visualizzazione di ulteriori componenti grafici per rendere chiara e fruibile l'esperienza utente. Tali librerie vengono riportate nel package.json che contiene altre informazioni utili alla configurazione generale del progetto.

1.1 Configurazione del Package.json

Prima di procedere con lo sviluppo effettivo dell'applicazione Health App, è stato necessario configurare il package.json, che rappresenta il manifesto dell'applicazione. Al suo interno, sono elencate le dipendenze delle librerie di terze parti necessarie per il progetto, insieme alle rispettive versioni. In esso sono presenti anche altre configurazioni che riguardano gli script per eseguire la build del progetto e l'ambiente di sviluppo dove va specificato il proxy [38]. Quest'ultimo serve a reindirizzare le chiamate del front-end ad un back-end che può essere eseguito in locale, su un indirizzo del tipo http://localhost:<porta> , oppure in remoto. È importante notare che l'uso del proxy è limitato all'ambiente di sviluppo [38].

1.1.1 Scripts NPM

Nel campo "scripts", è possibile definire scripts personalizzati [39] come ad esempio "start", che avvia l'applicazione React in modalità di sviluppo. Durante il processo di distribuzione dell'applicazione Health App, lo script "build" nel package.json, viene eseguito per compilare il codice sorgente dell'applicazione e generare un pacchetto finale ottimizzato per la distribuzione che include i file HTML, CSS, JavaScript e altre risorse necessarie per l'esecuzione dell'applicazione [40].

1.1.2 Gestione delle dipendenze

Le dipendenze del progetto dichiarate nel package.json rappresentano le librerie esterne che l'applicazione web ha bisogno per funzionare correttamente. Quando si esegue il comando "npm install", il node package manager (NPM) scarica e installa automaticamente tutte le dipendenze e le colloca in una cartella denominata "node_modules". Questo rende facile per gli sviluppatori impostare l'ambiente di sviluppo e garantire che le librerie necessarie siano disponibili durante la distribuzione [41].

Nel caso di HealthApp, le librerie di terze parti utilizzate sono state le seguenti:

- *mui/material*: libreria di componenti grafici di Material-UI per React. Fornisce una vasta gamma di componenti predefiniti, come bottoni, modali, tabelle e molti altri, che possono essere utilizzati per creare interfacce utente moderne secondo le linee guida del design Material;
- *mui/icons-material*: Questa dipendenza fornisce una vasta gamma di icone predefinite nel formato Material Design, che possono essere utilizzate per aggiungere simboli visivi e migliorare l'aspetto e la comprensibilità dell'interfaccia utente;
- *mui/lab*: offre una serie di componenti e funzionalità sperimentali che non sono ancora state completamente integrate nella libreria principale di Material-UI;
- *mui/x-data-grid*: è una libreria che fornisce un componente di griglia dati avanzato basato sul design Material ;
- *mui/x-date-pickers*: è una libreria che consente agli utenti di selezionare una data da un calendario interattivo basato sul design Material ;
- *nivo/bar*: fornisce un componente per la visualizzazione di grafici a barre altamente personalizzabile. Inoltre, fornisce modalità di personalizzazione avanzate, come l'orientamento delle barre, la scala degli assi, la gestione degli eventi interattivi e l'aspetto visivo. È possibile configurare le proprietà dei dati, come valori, etichette e colori, per creare grafici a barre adatti alle proprie esigenze;
- *reduxjs/toolkit*: Redux Toolkit fornisce un set di API più semplice per definire azioni, funzioni reducers e per configurare lo store Redux ;
- *Axios*: è una libreria JavaScript che facilita le richieste HTTP asincrone sia lato client che lato server. È ampiamente utilizzata per effettuare chiamate API e interagire con servizi esterni in modo semplice ed efficiente;
- *dayjs*: è una libreria JavaScript per la manipolazione delle date e degli orari;

- *notistack*: è una libreria per la gestione delle notifiche in applicazioni React. Fornisce un'interfaccia semplice per mostrare notifiche di sistema o avvisi all'interno dell'interfaccia utente tramite snackbar;
- *react*: la libreria JavaScript open-source per la creazione di interfacce utente su cui è costruita la web application ;
- *react-dom*: un pacchetto complementare alla libreria React che fornisce metodi specifici per l'interazione con il Document Object Model (DOM) ;
- *react-icons*: è una libreria che fornisce un'ampia collezione di icone per l'utilizzo in applicazioni React;
- *react-redux*: è una libreria che fornisce un'integrazione ufficiale tra React e Redux, consentendo di gestire lo stato globale dell'applicazione in modo efficiente. React Redux semplifica l'utilizzo di Redux all'interno delle applicazioni React ;
- *react-router-dom*: è una libreria che fornisce il routing di navigazione per applicazioni React;
- *react-scripts*: è un pacchetto preconfigurato fornito con Create React App (CRA), uno strumento ufficiale per avviare rapidamente e configurare progetti React ;
- *redux*: libreria di gestione dello stato per applicazioni JavaScript. La principale differenza tra i moduli "redux" e "react-redux" è il livello di integrazione con la libreria React. Mentre Redux è una libreria di gestione dello stato indipendente da React, "react-redux" fornisce strumenti specifici per integrare Redux in applicazioni React.

1.2 Organizzazione dei componenti

Dal momento che i principali utilizzatori del front-end sono i medici, è stato necessario creare componenti specifici che costituiscono le pagine a cui essi possono accedere. Tuttavia, è previsto che il client web sia utilizzato anche dai pazienti, esclusivamente durante il primo accesso all'applicazione come spiegato in precedenza. Pertanto, l'albero dei componenti è organizzato in cartelle e sottocartelle al fine di identificare la funzionalità e gli utilizzatori. I componenti generici vengono posizionati nella cartella "components", quelli che riguardano le sezioni o pagine della web application utilizzate dai medici sotto la cartella “doctors”, e tutti i restanti nella cartella “patients” (**Figura 51**).

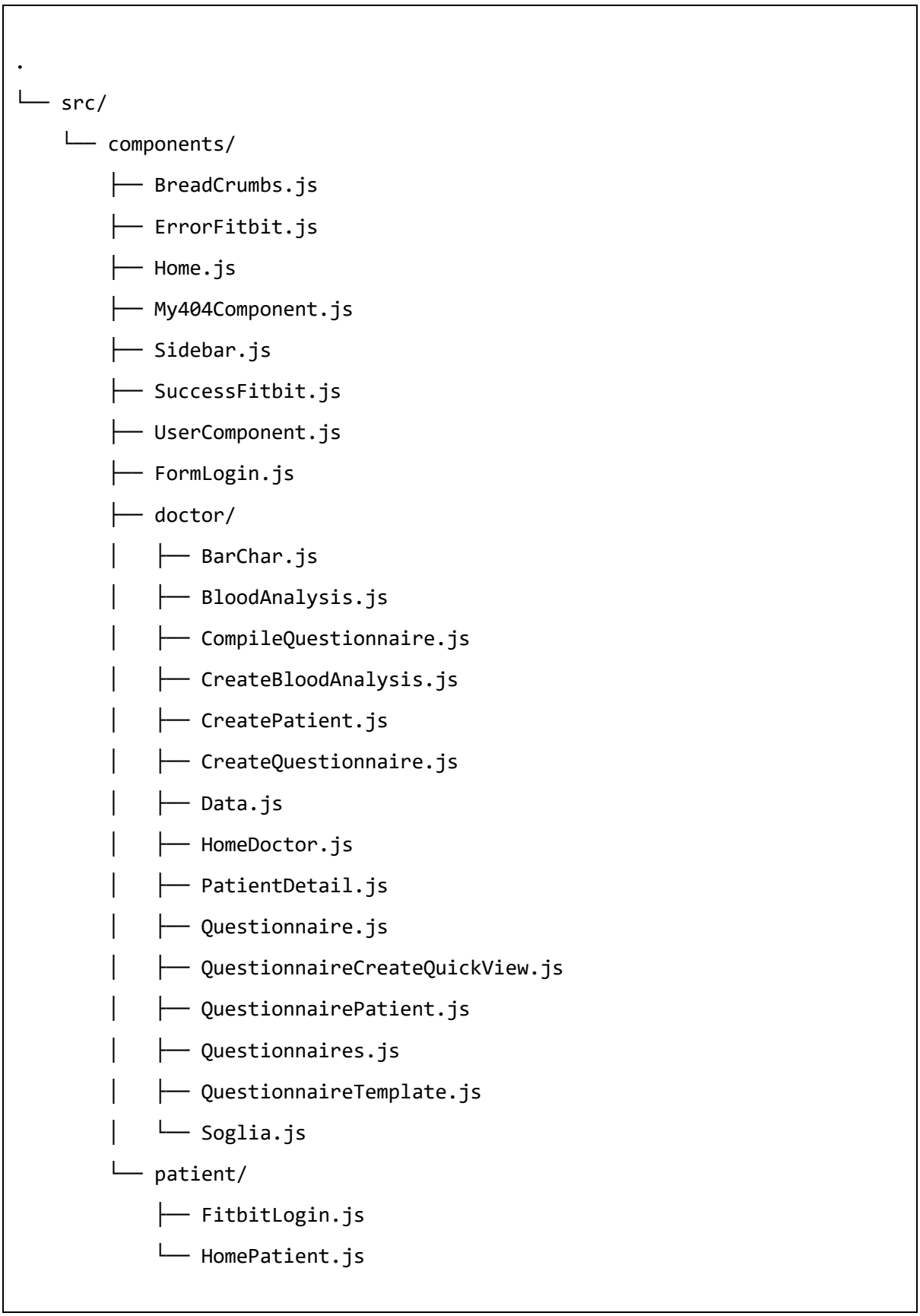

Figura 51 – Alberatura dei componenti nel progetto HealthApp del client-web.

Oltre ai componenti riportati in **Figura 51**, nell'alberatura generale del progetto è presente il componente "App" che fa parte di tutte le applicazioni React e inoltre può essere rinominato. Questo rappresenta il punto di ingresso delle applicazioni e si presenta nella forma di "functional component". Nel contesto di HealthApp, tra i componenti figli del componente principale, vi sono quelli "fissi", ovvero presenti in tutte le pagine, e quelli dinamici, specifici di ogni sezione. Tra i componenti fissi troviamo:

1) *`Sidebar.js`*: questo componente racchiude le informazioni dell'utente attualmente loggato e, in base al ruolo, fornisce una vista diversa dell'elenco delle macro-sezioni disponibili. Inizialmente, il client web era stato progettato anche per i pazienti, ma successivamente è stato ridefinito per essere utilizzato esclusivamente dai medici. Pertanto, le funzionalità in questione riguardano soltanto la visualizzazione e interazione con la lista dei pazienti e dei questionari (**Figura 52**).

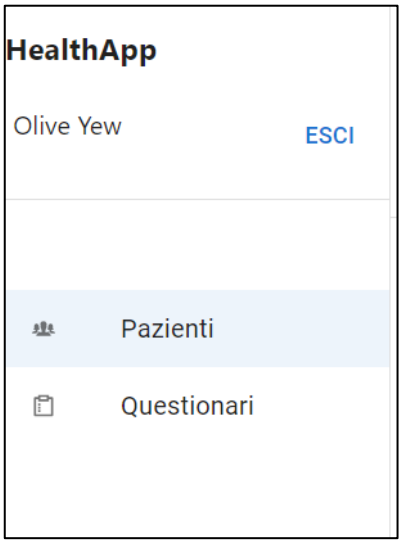

Figura 52 – Lista delle macro-sezione per un utente con ruolo di medico.

2) *`BreadCrumbs.js`*: questo componente mostra il percorso esatto (**Figura 53**) dell'applicazione in cui l'utente si trova permettendo di avere padronanza e conoscenza dello stato attuale dell'applicazione. È possibile utilizzare anche i tasti di navigazione del browser e la sezione degli URL, ma l'obiettivo è quello di consentire al medico di familiarizzare con l'applicazione HealthApp e non con il browser stesso.

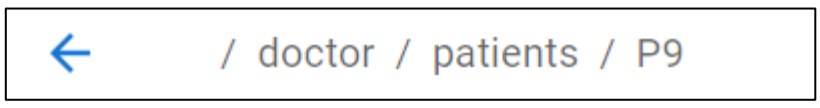

Figura 53 – Esempio di percorso: pagina del dettaglio del paziente.

3) *UserComponent.js*: questo componente verifica lo stato di autenticazione dell'utente. Ad esempio, se si aggiorna la pagina o dopo un periodo di inattività prolungata il medico effettua nuovamente l'accesso all'applicazione, questo componente si occupa di reindirizzare l'utente alla pagina di login nel caso in cui la sessione fosse scaduta, altrimenti viene ricaricata la pagina in cui si trovava l'ultima volta. Inoltre, è responsabile della visualizzazione degli errori che possono verificarsi durante l'utilizzo del client, in particolare quelli causati dalla comunicazione con il back-end quando si riceve una risposta http con codice di errore "4xx" o "5xx" [42]. È importante notare che gli errori possono essere molteplici e possono essere gestiti contemporaneamente. Ad esempio, nella sezione dei dati dei pazienti, potrebbero essere effettuate più richieste al back-end contemporaneamente, il che potrebbe portare a più errori. Gli errori vengono quindi gestiti come una pila in una "snackbar" [43] con la possibilità che scompaiano gradualmente (**Figura 54**). Inoltre, ha il compito di mostrare i messaggi di successo ogni qualvolta le operazioni di modifica dei dati nel back-end vengono effettuate senza generare errori (**Figura 55**).

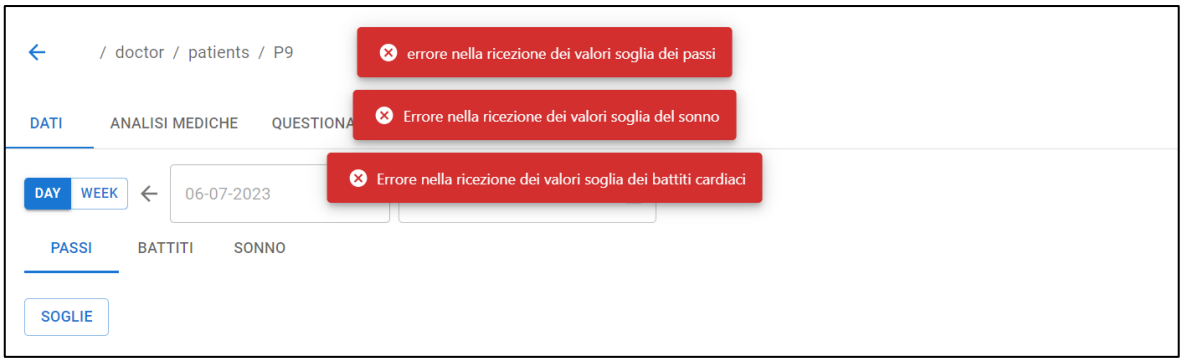

Figura 54 – Esempio di ricezione multipla di errori.

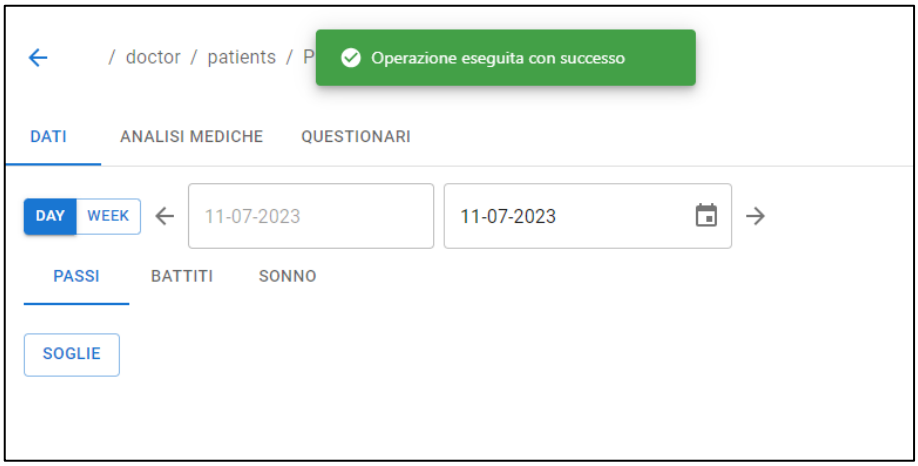

Figura 55 - Esempio di visualizzazione di messaggio di successo.

I componenti dinamici sono mostrati sulla base della posizione dell'utente all'interno dell'applicazione web, grazie al “routing”. In HealtApp, questo è stato reso possibile attraverso l’utilizzo del modulo “react-router-dom”. Il componente "Route" determina quale elemento visuale deve essere visualizzato in base all'URL attuale (**Figura 56**). La proprietà "exact" assicura che il componente referenziato da "Route" sia visualizzato solo quando il valore della sua proprietà "path" corrisponde esattamente a quello visualizzato nella barra degli indirizzi del browser. In generale, il modulo “react-router-dom” o semplicemente React Router adotta una strategia per cui il riscontro trovato è il primo ad avere un match con il valore della proprietà “path”, fra tutti gli altri componenti “Route”. Tuttavia, questa logica viene sovrascritta dalla presenza di "exact" [44].

È importante notare che in caso di inserimento da parte di un utente di un percorso che non rientra fra quelli specificati nelle properties “path” dei componenti Route, quindi di un percorso non valido, sarà visualizzato il componente “My404Component.js” come riportato in fondo al componente App in **Figura 56**. Il valore “*” indica in generale qualsiasi percorso, e in questo caso specifico il componente verrà visualizzato nello schermo nel caso in cui non venisse trovato un riscontro tra i componenti che lo precedono (**Figura 57**).

```
<Router>
  <SnackbarProvider variant="error" maxSnack={3} anchorOrigin={{horizontal: 'center', vertical: 'top' }}>


  <Box sx={{ display:"flex"}}>
  <Sidebar2></Sidebar2>
  <Box  component="main">

  <BreadCrumbs></BreadCrumbs>
  <AuthComponent></AuthComponent>
<Switch>
  <Route exact path="/" component={Home2} />
  <Route exact path="/patient/fitbit-login" component={FitbitLogin}/>
  <Route exact path="/success-fitbit" component={SuccessFitbit}/>
  <Route exact path="/error-fitbit" component={ErrorFitbit}/>
  <Route exact path="/patient" component = {HomePatient}/>
  <Route exact path="/form-login" component={FormLogin} />
  <Route exact  path="/admin/patients" component={HomeDoctor}/>
    <Route exact  path="/admin/patients/new" component={CreatePatient}/>
    <Route exact path="/admin/questionnaires" component={Questionnaires}/>
    <Route exact path="/admin/questionnaires/new" component={CreateQuestionnaire}/>
    <Route exact path="/admin/questionnaires/:questionnaireId" component={QuestionnaireTemplate}/>
    <Route exact path="/admin/patients/:id" component={PatientDetail}/>
    <Route exact path="/admin/patients/:patientId/questionnaires/new" component={Compilequestionnaire}/>
    <Route exact path="/admin/patients/:patientId/questionnaires/:questionnaireId" component={QuestionnairePatient}/>
    <Route exact path="/admin/patients/:patientId/newBloodAnalysis" component={CreateBloodAnalysis}/>
  <Route exact  path="/doctor/patients" component={HomeDoctor}/>
    <Route exact  path="/doctor/patients/new" component={CreatePatient}/>
    <Route exact path="/doctor/questionnaires" component={Questionnaires}/>
    <Route exact path="/doctor/questionnaires/new" component={CreateQuestionnaire}/>
    <Route exact path="/doctor/questionnaires/:questionnaireId" component={QuestionnaireTemplate}/>
    <Route exact path="/doctor/patients/:id" component={PatientDetail}/>
    <Route exact path="/doctor/patients/:patientId/questionnaires/new" component={Compilequestionnaire}/>
    <Route exact path="/doctor/patients/:patientId/questionnaires/:questionnaireId" component={QuestionnairePatient }/>
    <Route exact path="/doctor/patients/:patientId/newBloodAnalysis" component={CreateBloodAnalysis}/>
    <Route path="*" component={My404Component}/>
  <Route>
  </Route>
  <Route exact path='*'component={My404Component} />

</Switch>
</Box>
</Box>
</SnackbarProvider>
</Router>
);
```

Figura 56 – Struttura del componente App ed i componenti Route.

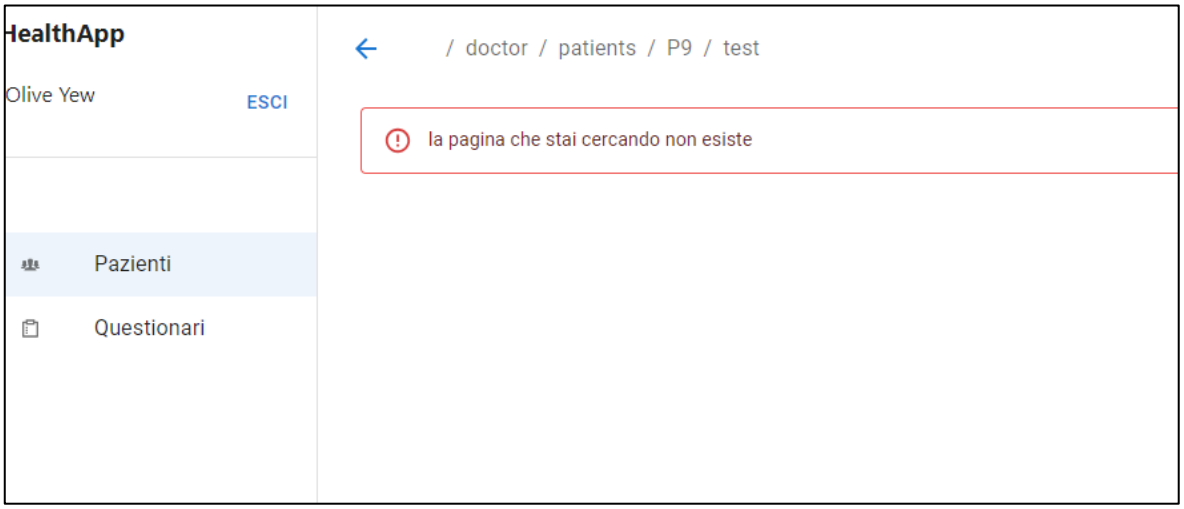

Figura 57 – Componente visualizzato per percorsi non validi.

1.3 Configurazione NGINX

Nginx gioca un ruolo cruciale nell'architettura complessiva, in quanto ha il compito di servire le risorse statiche dell'applicazione HealthApp e di reindirizzare le chiamate relative alle API al componente BE di HealthApp [45]. Quest'ultime iniziano con il prefisso "/api". Le altre richieste, invece, vengono utilizzate per servire gli elementi statici dell'applicazione web. Per configurare correttamente Nginx, è stato creato un file di configurazione chiamato ngnix.conf, (**Figura 58**) che fa parte del pacchetto finale ottenuto dal processo di build. Questo file contiene le informazioni sulla posizione delle risorse statiche, come file HTML, CSS, JavaScript e immagini, e mappa correttamente le richieste del browser alle risorse appropriate. Inoltre, viene specificata la porta sui cui il reverse proxy è in ascolto che, nel contesto di HealthApp, corrisponde alla porta 80. Infine, va riportato l'indirizzo dell'host in cui viene eseguito il componente BE per reindirizzare correttamente le richieste API.

```
server {
    listen 80;

    location /api {
        proxy_pass <host>;
        proxy_set_header Host $host;
    }

    location /api/login {
            proxy_pass <host>;
        }

    location /api/logout {
             proxy_pass <host>;
        }

    location ~ /static/media/(.*)\.(jpg|png|svg)$ {
            root /usr/share/nginx/html;
            index index.html index.htm;
            try_files /static/media/$1.$2 /index.html =404;
    }

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html =404;
    }

}
```

Figura 58 – File di configurazione Ngnix in HealthApp.

V) CONCLUSIONI E PROSPETTIVE FUTURE

Il progetto di sviluppo dell'applicazione web HealthApp mira a fornire ai medici uno strumento semplice e intuitivo per il monitoraggio dei pazienti, oltre all'inserimento di dati aggiuntivi come questionari personalizzati e valori soglia specifici per ciascun paziente. Questo consente la creazione di un database completo, facilitando l'indagine per il miglioramento del benessere dei pazienti. Ciò è reso possibile anche grazie all'integrazione di un motore di regole quale Drools nel componente back-end di HealthApp. La versione iniziale di HealthApp web ha funzionato come demo e punto di partenza per lo sviluppo dell'attuale versione, che è graficamente e funzionalmente più completa. Il progetto necessita di ulteriori integrazioni in relazione all'esperimento HealthApp. Come menzionato nei capitoli precedenti, si prevede che in futuro i dati raccolti da Fitbit e quelli qualitativi derivanti dai questionari potranno essere esportati attraverso l'applicazione web dai medici, permettendo analisi più avanzate con strumenti statistici di terze parti. L'adozione di librerie di terze parti come Redux per la gestione dello stato globale ha reso lo sviluppo di HealthApp efficiente e all'avanguardia. Questo ha permesso di istituire un framework di sviluppo che facilita l'identificazione di file, funzioni e le loro posizioni nel progetto, aspetto fondamentale per future integrazioni. È stato descritto il processo per integrare future API per interagire tramite operazioni REST, estendendo o creando i file *.service.js nella cartella corrispondente. È stato altresì illustrato come estendere o inserire nuove azioni nel contesto di Redux per aggiornare lo stato globale dell'applicazione, fornendo ai componenti un luogo unificato da cui prelevare i dati e visualizzarli a schermo. Queste azioni risiedono in file dedicati, il cui nome deve riflettere la risorsa e le informazioni che compongono lo stato. È stato implementato un meccanismo efficiente per mantenere la sessione dell'utente e le relative informazioni: salvare nel session storage del browser e nello stato globale Redux le informazioni relative all'utente connesso, evitando la propagazione delle stesse tramite props.

Tra le aree di miglioramento relative alla manutenzione dell'applicazione web HealthApp, rilevante è sicuramente la testabilità del codice, al fine di incrementare la sua robustezza. È in previsione lo sviluppo di Unit test e Integration test per permettere di individuare, nelle future integrazioni software, potenziali problematiche che potrebbero esporre l'applicazione a errori compromettenti durante l'esecuzione dell'esperimento.

Il processo di deployment è attualmente in fase di ottimizzazione. Al momento, l'applicazione viene eseguita su un container Docker, con le immagini create localmente e poi caricate su un Docker Hub riservato al gruppo di sviluppatori del progetto. Inoltre, si fa uso di un'istanza EC2 di Amazon, da cui si effettua il pull dell'immagine Docker e successivamente l'esecuzione del container. Queste operazioni possono essere migliorate e automatizzate, per permettere la condivisione di versioni demo dell'applicazione con il supervisore del progetto e con i medici. Questo permetterà di mostrare in anteprima alcune funzionalità e, una volta confermate, di trasferirle nell'ambiente ufficiale di produzione. È prevista la migrazione a un ambiente diverso da AWS, sostenuto direttamente dall'azienda ospedaliera di Verona, prima della messa in produzione dell'applicazione.

VI) BIBLIOGRAFIA

1. IBM Cloud Education, Three-Tier Architecture,
   https://www.ibm.com/cloud/learn/three-tier-architecture
2. Amazon Web Services, Three-tier architecture overview
   https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/three-tier-architecture-overview.html
3. Spring, Building web applications with Spring Boot and Kotlin
   https://spring.io/guides/tutorials/spring-boot-kotlin
4. Spring , Spring Data
   https://spring.io/projects/spring-data
5. Spring Boot, Spring for Apache Kafka
   https://spring.io/projects/spring-kafka
6. Spring Boot, Spring Security
   https://spring.io/projects/spring-security
7. Kinsta, What Is Express.js? Everything You Should Know
   https://kinsta.com/knowledgebase/what-is-express-js/
8. Rambabu Posa,Digitialocean, Node JS Architecture - Single Threaded Event Loop
   https://www.digitalocean.com/community/tutorials/node-js-architecture-single-threaded-event-loop
9. Drools, Drools Documentation
   https://docs.drools.org/7.73.0.Final/drools-docs/html_single/index.html
10. React native, Learn once, write anywhere.
    https://reactnative.dev/
11. React, The library for web and native user interfaces
    https://react.dev/learn
12. Cem Eygi, Freecodecamp, Cosa sono i componenti funzionali e di classe?
    https://www.freecodecamp.org/italian/news/componenti-funzionali-props-e-jsx-in-react-un-tutorial-per-principianti-su-react-js/
13. Oyedele Temitope,Freecodecamp, How Props Work in React – A Beginner's Guide
    https://www.freecodecamp.org/news/beginners-guide-to-props-in-react/
14. David Jöch,Medium, Functional vs Class-Components in React
    https://djoech.medium.com/functional-vs-class-components-in-react-231e3fbd7108
15. Geeksforgeeks, ReactJS setState()

https://www.geeksforgeeks.org/reactjs-setstate/

16. Rajesh Naroth,Medium, State Management within React Functional Components with hooks
https://rajeshnaroth.medium.com/component-state-management-in-react-functional-components-with-hooks-e7c412c05e71
17. Geeksforgeeks, Differences between Functional Components and Class Components in React
https://www.geeksforgeeks.org/differences-between-functional-components-and-class-components-in-react/
18. Gregório MJ, Rodrigues AM, Salvador C, Dias SS, de Sousa RD, Mendes JM, Coelho PS, Branco JC, Lopes C, Martínez-González MA, Graça P, Canhão H. Validation of the Telephone-Administered Version of the Mediterranean Diet Adherence Screener (MEDAS) Questionnaire. Nutrients. 2020 May 22;12(5):1511. doi: 10.3390/nu12051511. PMID: 32455971; PMCID: PMC7284796.
https://pubmed.ncbi.nlm.nih.gov/32455971/
19. Mui, Dialog
https://mui.com/material-ui/react-dialog/
20. Davide Cerbo, Medium, Redux in parole semplici, Davide Cerbo
https://davidecerbo.medium.com/redux-in-parole-semplice-6fec4a207c
21. Redux.js, Creating a Store
https://redux.js.org/tutorials/fundamentals/part-4-store
22. Redux.js,MiddleWare
https://redux.js.org/understanding/thinking-in-redux/glossary#middleware
23. Freecodecamp, Redux Thunk Explained with Examples
https://www.freecodecamp.org/news/redux-thunk-explained-with-examples/
24. Redux.js,Dispatching Actions
https://redux.js.org/tutorials/fundamentals/part-4-store#dispatching-actions
25. Redux.js, Splitting Reducers
https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers#splitting-reducers
26. Redux.js, todosSlice.js
https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers#splitting-reducers
27. Redux.js, combineReducers(reducers)
https://redux.js.org/api/combinereducers

28. David Ryan Morphew, Medium, Using Switch-Cases in React-Redux to Make State Change Dependencies
https://davidrmorphew.medium.com/using-switch-case-in-react-redux-to-make-state-change-dependencies-8ade636a4e39
29. Redux.js,Spread Operator
https://redux.js.org/usage/using-object-spread-operator
30. Andrea Chiarelli, Html.it, Definire le azioni che modificano lo stato
https://www.html.it/pag/65175/definire-le-azioni-che-modificano-lo-stato/
31. Redux.js, dispatch(action)
https://redux.js.org/api/store#dispatchaction
32. Mui, Material UI – Overview
https://mui.com/material-ui/getting-started/
33. Mui, Customizing components, Overriding with inline-styles
https://v4.mui.com/customization/components/#overriding-with-inline-styles
34. Mui, Styled components API
https://mui.com/system/styles/basics/#styled-components-api
35. Axios-http, What is Axios?
https://axios-http.com/docs/intro
36. Axios-http, Axios API, Request method aliases
https://axios-http.com/docs/api_intro
37. Axios-http, Request Config
https://axios-http.com/docs/req_config
38. Npmjs, Specifying dependencies and devDependencies in a package.json file
https://docs.npmjs.com/specifying-dependencies-and-devdependencies-in-a-package-json-file
39. Npmjs, scripts
https://docs.npmjs.com/cli/v6/using-npm/scripts
40. Appliedtechnology, Github, Using the proxy setting for React dev server
https://appliedtechnology.github.io/protips/ReactDevServerProxy.html
41. Lukas Oppermann, Builtin, Creating an NPM-Only Build Step for JavaScript — the Easy Way
https://builtin.com/software-engineering-perspectives/npm-only-build-step
42. Mozilla, HTTP response status codes
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
43. Material, Snackbars

https://m2.material.io/components/snackbars#usage

44. Geeksforgeeks, ReactJS Router
    https://www.geeksforgeeks.org/reactjs-router/
45. Mohammad Faisal, Medium, Using NGINX to serve React Application (Static vs Reverse Proxy)
    https://blog.devgenius.io/using-nginx-to-serve-react-application-static-vs-proxy-69b85f368e6c