



UNIVERSITA' DEGLI STUDI ROMA TRE

Dipartimento di Ingegneria Civile, Informatica e delle Tecnologie Aeronautiche

Laurea in Ingegneria Informatica

**REALIZZAZIONE DI UN SISTEMA DI
AMMINISTRAZIONE PER IL TRACCIAMENTO DELLE
PERFORMANCE DI UN CHATBOT**

Laureando

Edoardo Piazzolla

Matricola 577068

Relatore

Prof. Giuseppe Di Battista

Correlatore

Dott. Nicolò Palmiero

Anno Accademico 2024/2025

A Zio Ricky...

RINGRAZIAMENTI

L'unica cosa che realmente possediamo in questa vita, sono momenti nel tempo. Tipo questo esatto momento, quando sto per raggiungere uno dei miei più grandi obiettivi. Gli orologi, le auto, i soldi, non te li porti nella tomba. Ma questi momenti della vita sì, saranno tuoi per sempre e nessuno potrà mai levarteli. Mai. Qualunque cosa succeda, rimarranno nella tua anima, e quando sarà il momento, voleranno via con te.

Desidero esprimere la mia più profonda gratitudine verso coloro che hanno fatto parte del mio percorso accademico, e che in qualche modo, hanno contribuito a nutrire la fiamma dentro di me che ardeva per raggiungere questo traguardo.

Ringrazio il mio relatore, il professor Giuseppe Di Battista, per la disponibilità, i preziosi consigli e l'approccio sempre pacato e tranquillo che hanno reso il lavoro su questa tesi un'esperienza costruttiva e serena.

Ringrazio l'azienda Bytewise per avermi accolto nelle vesti di tirocinante, offrendo un ambiente stimolante che ha rappresentato per me un'importante introduzione al mondo del lavoro. Un ringraziamento particolare va a Lorenzo Giuseppi, al mio tutor aziendale Nicolò e agli altri membri della sezione sviluppo, che con la loro professionalità, disponibilità e cordialità hanno reso il mio tirocinio non solo interessante e formativo, ma anche coinvolgente e mai monotono.

Ringrazio Leonardo, Vincenzo e tutti i miei colleghi universitari con cui ho condiviso lezioni, sessioni di studio, esami, vittorie e sconfitte. La loro fiducia in me e il loro esempio sono stati una fonte di ispirazione costante, spronandomi a dare sempre il meglio di me stesso.

Ringrazio Alessandro e Barbara, i miei genitori, e mia sorella Priscilla, che mi hanno insegnato ad affrontare la vita come una gara di corsa, con impegno e sacrificio, un kilometro alla volta. Mi hanno aiutato a trovare il mio passo per raggiungere con soddisfazione i miei obiettivi e mi hanno teso la mano per aiutarmi a rialzarmi nei momenti difficili.

Ringrazio Nonna Anna e Nonno Ettore, che hanno sempre desiderato un nipote ingegnere. Nonno Vito e Nonna Carla, che nonostante la distanza mi hanno fatto sentire il loro amore e supporto lungo il mio percorso.

Ringrazio Gabriele, il mio migliore amico, per la pazienza che ha nell'ascoltarmi, sin dal primissimo giorno di liceo. Rappresenti un pilastro fondamentale, un punto di riferimento quando ho bisogno di confidarmi, che mi comprende e mi dà conforto.

Infine, desidero ringraziare me stesso per non aver ceduto durante questo percorso di studi, per aver affrontato ogni problema trasformandolo in un'opportunità. Ogni giorno ho avuto la possibilità di posizionare un mattone nel miglior modo possibile, costruendo passo dopo passo la vita che desidero. Voglio ricordarmi che i limiti, così come le paure, sono spesso solo un'illusione.

INDICE

RINGRAZIAMENTI	3
INDICE	6
INTRODUZIONE	8
1. CAPITOLO PRIMO: CONTESTO APPLICATIVO	11
1.1 Contesto Operativo	11
1.2 Dashboard di Amministrazione	12
1.3 Database Relazionali e Non Relazionali	13
1.4 Ricerche Precedenti	14
2. CAPITOLO SECONDO: TECNOLOGIE UTILIZZATE	17
2.1 Typescript	17
2.2 React	17
2.3 MariaDB	18
2.4 TypeORM	18
2.5 MongoDB	19
2.6 AmCharts	20
2.7 JSON Web Token	20
2.8 Insomnia e Postman	21
2.9 React DnD	21
2.10 Jest	21
3. CAPITOLO TERZO: RECUPERO E GESTIONE DEI DATI	23
3.1 Fetch e Gestione delle API	23
3.1.1 Struttura della Funzione Fetch	23
3.1.2 Gestione delle Richieste su Diversi Ambienti	24
3.2 Elaborazione e Salvataggio dei Dati	25
3.3 Utilizzo dei Dati nei Grafici	26
3.3.1 Assegnazione dei Dati alle Serie	27
3.4 Esempio di Flusso dei Dati	27
4. CAPITOLO QUARTO: ANALISI DEI REQUISITI	30
4.1 Requisiti Funzionali	30
4.2 Requisiti Non Funzionali	31
4.3 Casi D'Uso	31
4.4 Modello di Dominio	33
5. CAPITOLO QUINTO: FUNZIONALITÀ REALIZZATE	35

5.1	Architettura della Dashboard	35
5.2	Home Page	35
5.2.1	Gestione dei Widget tramite Drag & Drop	37
5.2.2	Componente Section come Area di Drop	37
5.2.3	Componente Card come Elemento Trascinabile	38
5.3	Side Bar	39
5.4	Top Bar	43
5.5	Sicurezza e Gestione della Privacy	44
5.5.1	Crittografia della Email	44
5.5.2	Decrittografia della Email	45
5.5.3	Hashing della Password	45
5.6	Gestione del Login	46
5.6.1	Verifica delle Credenziali	46
5.6.2	Gestione della Sessione	47
5.7	Gestione del Logout	47
5.8	State Management: Aggiornamento dei Grafici	48
6.	CAPITOLO SESTO: VERIFICHE FUNZIONALI	50
6.1	Test su Autenticazione e Sicurezza	50
6.1.1	Verifica della Validazione dei Campi	50
6.1.2	Gestione di Login Falliti	50
6.1.3	Gestione della Sessione	50
6.2	Test di Navigazione e Usabilità	51
6.3	Test di Aggiornamento e Correttezza dei Dati	51
7.	CAPITOLO SETTIMO: CONCLUSIONI E SVILUPPI FUTURI	53
	BIBLIOGRAFIA	55

INTRODUZIONE

I chatbot rappresentano negli ultimi anni uno strumento di fondamentale importanza per le aziende ai fini dell'ottimizzazione della comunicazione con i clienti, dei processi interni e della disponibilità del servizio h24. La crescita esponenziale di questa tecnologia è dimostrata dall'ecletticità di settori cui è possibile adattarsi, dall'e-commerce alla sanità, dal supporto tecnico all'educazione, sempre seguendo tutti un'unica regola: risposta immediata e personalizzata.

Di fronte a queste nuove opportunità, le organizzazioni, tuttavia, devono affrontare una serie di nuove sfide. Non è più sufficiente installare un chatbot, è necessario utilizzare strumenti per monitorare l'efficacia e migliorarne le prestazioni. Poiché le intelligenze artificiali sono utilizzate da umani, sapere come i chatbot vengono effettivamente utilizzati, quali domande ricevono, quali sono le loro debolezze e quali aspetti specifici possono essere migliorati è fondamentale per garantire un servizio di alta qualità e massimizzare il ritorno sull'investimento.

Il progetto descritto in questa tesi è stato realizzato in collaborazione con la società *Bytewise*, azienda leader nel mercato dell'Information e Communication Technology dal 1999.

L'obiettivo principale è stato la realizzazione di un sistema per la creazione di un chatbot addestrabile attraverso l'integrazione di un prodotto esterno, *Private-GPT*, che consente di addestrare un chatbot utilizzando documenti forniti dagli utenti e di interagirvi mediante domande specifiche.

Il mio contributo si è concentrato sulla progettazione e implementazione di un'interfaccia di amministrazione per la gestione e la visualizzazione delle statistiche del chatbot, permettendo di analizzare l'utilizzo e l'efficacia del chatbot stesso. Ho quindi progettato statistiche significative, e le ho estrapolate dai dati disponibili per poi presentarle in modo chiaro e intuitivo.

A tal fine, ho sfruttato ampiamente la libreria amCharts, che mi ha permesso di implementare grafici di vario tipo, progettati per essere intuitivi e di facile comprensione.

Nel primo capitolo è stato presentato il quadro generale entro il quale si inserisce il progetto e, nello specifico, il ruolo e l'importanza della figura dei chatbot all'interno delle aziende: ciò illustra i principali concetti generali fondamentali per affrontare le tematiche che sono oggetto di studio nei capitoli successivi.

Nel secondo capitolo verranno presentate le tecnologie e gli strumenti coinvolti nello sviluppo dell'applicativo: si tratta di un'introduzione di base, chiara e comprensibile, corredata dalle motivazioni delle scelte tecnologiche compiute.

Nel terzo capitolo verrà approfondito il processo di acquisizione e gestione dei dati utilizzati per la visualizzazione grafica delle statistiche del chatbot all'interno del dashboard di amministrazione. Verranno descritti i metodi di acquisizione dei dati, il loro trattamento per garantire coerenza e correttezza e le modalità con cui vengono resi compatibili con i grafici.

Il quarto capitolo riguarda l'analisi dei requisiti, condotta in modo accurato per guidare lo sviluppo del progetto in tutte le sue fasi. Verranno precisati gli obiettivi e le funzionalità del chatbot tramite l'analisi dei casi d'uso in relazione all'interpretazione del modello di dominio.

Il quinto capitolo descrive in dettaglio le operazioni e le funzionalità implementate, che rispondono alle specifiche individuate nell'analisi dei requisiti.

Nel sesto capitolo vengono presentate le verifiche funzionali condotte per dimostrare la corretta implementazione del sistema. Vengono illustrati i test case definiti per validare le singole funzionalità e i test funzionali eseguiti direttamente sull'applicazione, al fine di valutare il comportamento del sistema in un contesto reale.

Infine, nel settimo capitolo si presentano le conclusioni, accompagnate da un'analisi critica del processo di progettazione e sviluppo. Vengono messi in evidenza i risultati ottenuti e si suggeriscono possibili sviluppi futuri.

1. CAPITOLO PRIMO: CONTESTO APPLICATIVO

Il seguente capitolo ha lo scopo di delineare il contesto applicativo all'interno del quale si colloca il progetto discusso in questa tesi.

1.1 Contesto Operativo

Bytewise, azienda che ha ospitato il mio percorso di tirocinio, lavora in 4 principali ambiti:

- **Supporto IT:** si concentra sulla fornitura di servizi critici per garantire l'efficacia e la sicurezza delle infrastrutture tecnologiche aziendali. Grazie alla loro esperienza pluriennale, Bytewise offre soluzioni personalizzate che spaziano dalla sicurezza informatica alla gestione dei data center.
- **Farm e Cloud Computng:** Bytewise offre soluzioni personalizzate e infrastrutture dedicate progettate per garantire i più alti standard di sicurezza e privacy. Dispone di risorse tecnologiche proprie, tra cui sistemi di storage e backup, che servono ad ospitare soluzioni e infrastrutture IT dei clienti.
- **Sicurezza Informatica:** offre un approccio integrato alla sicurezza informatica, garantendo la conformità alle normative come il GDPR e progettando le risorse digitali di un'azienda attraverso processi avanzati di gestione e valutazione.
- **Sviluppi Applicativi:** La sezione di sviluppo di Bytewise si occupa di realizzare soluzioni avanzate, tra cui app per dispositivi mobili IOS/Android, web application, API RESTful ad alte prestazioni con Node.js e sistemi di indicizzazione e analisi statistica basati su Elasticsearch e RabbitMQ, e microservizi scalabili.

Per quanto riguarda il mio caso, ho avuto l'opportunità di lavorare nella sezione di sviluppi applicativi, dove vengono utilizzati i principali strumenti e tecnologie che sono stati necessari per lo sviluppo del progetto su cui si basa questo elaborato, garantendo una perfetta integrazione con le infrastrutture esistenti.

1.2 Dashboard di Amministrazione

Il progetto su cui è basato questo elaborato, denominato per semplicità dashboard di amministrazione, riguarda la progettazione e lo sviluppo di un'applicazione web destinata agli utenti amministratori di un'azienda o di un servizio. Questa applicazione permette di monitorare in maniera intuitiva, rapida e organizzata le prestazioni del chatbot utilizzato, attraverso grafici interattivi e statistiche numeriche, garantendo un'esperienza utente chiara e funzionale.

La successiva descrizione è strutturata come parte backend e frontend del progetto, solo per l'esigenza del lettore. Inoltre, le tecnologie utilizzate verranno approfondite nel secondo capitolo.

Quando si parla di backend, ci si riferisce alla parte software che gestisce la logica applicativa, l'elaborazione dei dati e la comunicazione con i database. Il backend è essenziale per il corretto funzionamento delle applicazioni anche se non visibile direttamente agli utenti, in quanto funge da intermediario tra il frontend e le risorse server. Nel caso di questo progetto, il backend elabora le richieste, accede ai dati e fornisce le risposte necessarie al lato frontend dell'applicazione. È stato sviluppato utilizzando TypeScript, un'estensione tipizzata di JavaScript.

È stato implementato un sistema basato su JSON Web Token per l'autenticazione, utilizzato per autenticare le richieste successive al login. I database utilizzati per la memorizzazione dei dati sono stati MongoDB, usato per salvare dei dati provenienti dal chatbot, cioè chat e messaggi, e MariaDB, usato per memorizzare le informazioni su degli utenti, nello specifico gli amministratori. La comunicazione in tempo reale tra i sotto sistemi è stata realizzata tramite WebSocket. Strumenti come Insomnia e Postman sono stati utilizzati per testare e validare i metodi esposti dai servizi backend, compresi le richieste HTTP implementate.

Il termine frontend si riferisce alla parte di un'applicazione che è visibile agli utenti, cioè l'interfaccia utente (User Interface). È la parte di un'applicazione con la quale gli utenti interagiscono. Il frontend è responsabile della presentazione dei dati ricevuti dal backend e dell'implementazione della logica di interazione. Nel caso di questo

progetto, il frontend è stato sviluppato principalmente con React e TypeScript, che hanno consentito la costruzione di un'interfaccia moderna, modulare e tipizzata, migliorando sia l'organizzazione del codice che la sua manutenibilità.

Node.js è stato utilizzato per supportare lo sviluppo e l'esecuzione degli strumenti necessari per il frontend. Per garantire un accesso sicuro è stata implementata una pagina dedicata alla gestione del login e della registrazione. Ogni utente dopo essersi può accedere alle funzionalità e informazioni presenti nel dashboard di amministrazione che sono protette.

Sono state inoltre integrate diverse librerie, come amCharts per la visualizzazione dei dati tramite grafici, e altre librerie utili per facilitare e ottimizzare la realizzazione delle funzionalità, per rendere l'interfaccia interattiva e facile da usare, e renderla visivamente accattivante. Queste tecnologie verranno descritte più nel dettaglio nei capitoli successivi.

1.3 Database Relazionali e Non Relazionali

Nel panorama moderno dello sviluppo software i database rappresentano la colonna portante della gestione dei dati. I principali approcci utilizzati a tale scopo sono i database relazionali e i non relazionali.

I database relazionali, come MariaDB, organizzano i dati in tabelle con righe e colonne, seguendo un modello strutturato basato su relazioni, che vengono definite da chiavi primarie (primary key) e chiavi esterne (foreign key). Questo tipo di database fa uso del linguaggio SQL (Structured Query Language) per eseguire delle operazioni sui dati. Sono vantaggiosi in quanto la loro struttura tabellare garantisce un'organizzazione rigorosa e una maggiore facilità nella gestione di relazioni complesse. Inoltre, permettono di mantenere coerenza e integrità tra i dati grazie ai vincoli che ho citato in precedenza, quali le chiavi primarie e esterne.

D'altra parte i database non relazionali, come MongoDB, sono progettati per gestire dati non strutturati o semi-strutturati, in quanto non seguono uno schema rigido e memorizzano i dati in formati flessibili, come documenti JSON. Conferiscono flessibilità per applicazioni web che gestiscono dati dinamici o non uniformi, come

in questo contesto, dati generati da un chatbot. Inoltre, la loro scalabilità orizzontale permette di gestire con più efficienza grandi volumi di dati attraverso il sharding e la distribuzione su più server.

1.4 Ricerche Precedenti

La storia dei chatbot risale agli anni '60 e ha inizio con un esperimento di Weizenbaum al MIT in cui nacque il primo chatbot in grado di sostenere conversazioni psicologiche semplici, ma sufficienti per soddisfare gli utenti dell'esperimento (Tscheligi et al., 2008). Lo sviluppo del chatbot ELIZA da parte di Weizenbaum ha stimolato gli sviluppi futuri nel sorprendente campo di interazione uomo-macchina.

L'avvento dei chatbot nei servizi di supporto clienti ha rivoluzionato il modo in cui le aziende si relazionano con i propri clienti. I chatbot possono elaborare grandi quantità di richieste e comporre risposte a quelle più comuni, fornendo informazioni e risolvere i problemi in tempo reale, il che migliora l'efficienza operativa e il livello di soddisfazione dei clienti. Sono, sicuramente, un'innovazione eccezionale sempre in fase di miglioramento, ma la loro efficacia deve essere controllata e verificata regolarmente.

Per questo, l'analisi delle statistiche e i feedback degli utenti sono rilevanti. Secondo Gartner, per il 67% degli utenti, la qualità delle risposte è la cosa più importante. Inoltre, le metriche di velocità come il tempo medio di risposta sono correlate all'efficienza. Anche il coinvolgimento degli utenti, valutato attraverso indicatori come il numero di interazioni e la durata media delle sessioni, fornisce una chiara visione dell'operato del chatbot.

Come Chatbots Magazine (2019) afferma, i layout visivi dei dati mediante dashboard e grafici creano un modo visivo per interpretare i dati e rendere gli amministratori più abili a prendere decisioni basate sui dati.

L'obiettivo principale della dashboard di amministrazione era quello di fornire statistiche presentate in modo chiaro ed intuitivo per facilitare la comprensione delle metriche e della soddisfazione degli utenti. Questo obiettivo è stato ampiamente

raggiunto e gli amministratori sono stati in grado di utilizzare lo strumento per distinguere esattamente dove necessitava di miglioramenti e ottimizzare ulteriormente l'offerta di servizi.

2. CAPITOLO SECONDO: TECNOLOGIE UTILIZZATE

Questo capitolo è dedicato ai dettagli dell'implementazione. Vengono descritte le tecnologie che sono state utilizzate per lo sviluppo del progetto da zero.

Per facilitarne l'utilizzo, ognuna di queste tecnologie è stata studiata accuratamente in modo da poterne comprendere il funzionamento per garantirne un uso corretto, sfruttandone al meglio le potenzialità e migliorare la qualità del progetto.

2.1 Typescript

È un superset di JavaScript sviluppato da Microsoft che aggiunge la tipizzazione statica e altre funzionalità avanzate al linguaggio, in modo da migliorare la produttività e manutenibilità del codice, rendendolo più leggibile e facile da scalare nei progetti complessi.

Essendo compilato in JavaScript, esso è compatibile con qualsiasi ambiente di sviluppo che supporti JavaScript.

I punti di forza di questo linguaggio di programmazione sono sicuramente la tipizzazione statica, che riduce gli errori a runtime e migliora l'affidabilità, la compatibilità con JavaScript permette di utilizzare qualsiasi libreria JavaScript senza avere problemi.

Queste caratteristiche lo rendono versatile, e per questo è ampiamente utilizzato nello sviluppo web con l'utilizzo di framework come React.

2.2 React

React è una libreria JavaScript open-source utilizzata principalmente per la costruzione di interfacce utente. Ciò che rende speciale questa libreria è la possibilità di creare delle componenti riutilizzabili, che semplificano lo sviluppo di interfacce dinamiche ed efficienti, come nel caso del dashboard di amministrazione su cui si basa questo elaborato.

In sostanza l'interfaccia utente viene rappresentata attraverso un albero dove ogni nodo corrisponde ad un elemento HTML. Questa rappresentazione prende il nome di

Document Object Model (DOM). In React, è presente una copia leggera del DOM, ovvero il Virtual DOM, utilizzata per migliorare le performance. Questo viene fatto perché quando viene modificato un componente, React aggiorna il Virtual DOM e confronta le differenze con il DOM reale, aggiornando effettivamente solo le parti modificate del DOM reale in modo da ottimizzare i tempi di rendering.

Per quanto riguarda i dati, React fa uso dello stato (State), una struttura che permette a un componente di memorizzare e gestire i dati che influenzano e caratterizzano la sua rappresentazione. Ogni volta che lo stato cambia, il componente si ri-renderizza, comportamento utile per gestire le interazioni dinamiche, come nel caso di questo progetto.

Un altro particolare strumento utilizzato di pari passo con lo stato sono gli hook, ovvero funzioni che permettono di gestire lo stato e altre funzionalità senza dover necessariamente costruire una classe. Sono quindi presenti hook per gestire lo stato locale del componente, gli effetti collaterali di operazioni come fetch e aggiornamenti del DOM e la condivisione dello stato tra componenti senza passare props manualmente.

2.3 MariaDB

Come anticipato nel capitolo precedente, MariaDB è un relational database management system (RDBMS), ovvero un sistema software progettato per consentire la creazione, l'interrogazione e la manipolazione di una o più basi di dati.

Si presta in maniera eccellente per la gestione di dati strutturati, ovvero informazioni che sono organizzate in tabelle e con relazioni ben definite. Nel contesto di questo progetto è stato utilizzato per la memorizzazione di informazioni di utenti e modelli sotto forma di tabelle, ognuna definita come uno schema che specifica i tipi dei dati delle colonne.

2.4 TypeORM

Un ORM (Object Relational Mapper) è uno strumento che permette di interagire con un database relazionale, come MariaDB, usando oggetti e metodi della programmazione orientata agli oggetti, senza scrivere direttamente query SQL.

TypeORM consente di definire i dati come entità in TypeScript o JavaScript, rappresentando le tabelle del database come classi. Permette anche di gestire modifiche al database nel tempo e di creare relazioni tra entità, che si traducono automaticamente in relazioni tra le tabelle.

```
@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  firstName: string;

  @Column()
  lastName: string;

  @Column({ unique: true, collation: "utf8mb4_bin" })
  username: string;

  @Column({ unique: true })
  email: string;

  @Column()
  password: string;

  @Column()
  salt: string;
}
```

Figura 2.1 Esempio Entità

I campi della classe sono arricchiti da decoratori che hanno la funzionalità di aggiungere metadati e comportamenti in modo dichiarativo.

Ad esempio, *@Entity* indica che la classe rappresenta una tabella in un database relazionale, il cui nome viene automaticamente dedotto dal nome della classe.

Il decoratore *@PrimaryGeneratedColumn*, invece, specifica che la colonna associata sarà la chiave primaria della tabella.

2.5 MongoDB

A differenza di MariaDB, questo è un tipo di database NoSQL, orientato ai documenti, progettato per gestire grandi volumi di dati. In questo progetto è servito a memorizzare i record delle chat provenienti dal chatbot.

MongoDB non utilizza tabelle e righe ma archivia i dati sotto forma di documenti JSON-like all'interno di collezioni, per intenderci l'equivalente delle tabelle in un database relazionale, ma senza schema fisso.

2.6 AmCharts

AmCharts è una libreria JavaScript progettata per creare grafici e visualizzazioni interattive su pagine web. La sua ampia gamma di grafici permette di rappresentare i dati in modo visivo ed efficace. Inoltre grazie alla sua interattività avanzata è possibile esplorare i grafici e i loro dati in maniera più coinvolgente e dinamica attraverso funzionalità come lo zoom e tooltip.

Ciò che ha reso AmCharts la migliore opzione è la sua alta personalizzazione che ha permesso di creare stili unici per ogni grafico. Questa flessibilità ha permesso di adattare perfettamente l'aspetto visivo dei grafici allo stile della pagina web del dashboard di amministrazione, con l'obiettivo di garantire una coerenza estetica e migliorare l'esperienza dell'utente.

Per rendere la consultazione dei dati più dinamica e ricca, sono stati aggiunti altri strumenti all'interno dei charts come tooltip, per avere informazioni aggiuntive sui dati che si stanno visualizzando, leggende per identificare le diverse serie di dati rappresentati, zoom e panoramica per variare l'area di esplorazione di un determinato grafico, assi temporali per tracciare l'evoluzione dei dati riscontrati nel tempo, filtri e selettori per includere ed escludere determinati dati per rendere più specifica la consultazione e infine animazioni per attirare l'attenzione sugli aggiornamenti e cambiamenti dei dati nel tempo.

2.7 JSON Web Token

Nell'ambito dell'autenticazione di un utente, sono stati utilizzati dei token per mantenere attiva la sessione e autorizzare le richieste successive. La creazione del token avviene in fase di login, ed è permessa attraverso l'utilizzo della libreria **jsonwebtoken** (jwt).

```
export const generateToken = (userId: number) => {  
  return jwt.sign({ userId }, process.env.JWT_SECRET as string, {  
    expiresIn: "24h",  
  });  
};
```

Figura 2.2 Esempio di metodo che genera il token

2.8 Insomnia e Postman

L'utilizzo di strumenti per testare le API e le richieste HTTP è stato di vitale importanza nel percorso di sviluppo di questo progetto. Per questo motivo Insomnia e Postman sono stati largamente utilizzati a tale scopo. Hanno permesso di inviare richieste http come GET, POST, PUT e DELETE per testare le funzionalità che sono state implementate nella parte Back End del progetto, senza necessariamente realizzare un Front End.

2.9 React DnD

React DnD è un insieme di utility per React che facilita la creazione di interfacce drag-and-drop complesse, mantenendo i componenti indipendenti tra loro. La sua integrazione nel progetto è stata essenziale per implementare il trascinamento e rilascio dei widget, migliorando l'interattività dell'interfaccia utente.

2.10 Jest

Jest è stato utilizzato come framework di testing per garantire la correttezza e l'affidabilità delle funzionalità implementate. La sua capacità di eseguire test unitari e di integrazione ha permesso di verificare il comportamento dei componenti e delle funzioni chiave del progetto, contribuendo a identificare e correggere eventuali errori. Grazie al supporto per il mocking e alla sua integrazione con React Testing Library, è stato possibile testare il codice in modo efficiente e automatizzato, assicurando il rispetto dei requisiti funzionali e non funzionali.

3. CAPITOLO TERZO: RECUPERO E GESTIONE DEI DATI

In questo capitolo verrà descritto il processo di recupero e gestione dei dati all'interno del progetto, attraverso l'analisi delle tecnologie utilizzate e le motivazioni dietro la loro scelta. Prima di entrare nel dettaglio dell'implementazione, verranno descritti gli strumenti adottati per la comunicazione tra il dashboard di amministrazione (lato frontend) e il database che fornisce i dati (lato backend).

3.1 Fetch e Gestione delle API

Per poter visualizzare le informazioni nei grafici della dashboard, è fondamentale recuperare i dati dal database gestito dal backend. La funzione *fetch()* è un metodo dell'interfaccia Window che consente di inviare richieste HTTP verso un server remoto, restituendo una Promise che viene risolta con un oggetto Response.

3.1.1 Struttura della Funzione Fetch

Il metodo *fetch()* accetta come primo parametro un percorso URL, che rappresenta l'indirizzo della risorsa da recuperare. Questo URL deve seguire una struttura ben definita affinché il backend possa interpretare correttamente la richiesta e fornire i dati richiesti. Questo perché in un backend ben strutturato tramite un sistema di routing come nel caso di questo progetto, il percorso dell'URL è fondamentale per indirizzare la richiesta al corretto gestore (handler), che a sua volta interagisce con il database per estrarre le informazioni richieste. Il secondo parametro è un oggetto opzionale che permette di configurare vari aspetti della richiesta, come il metodo HTTP (GET, POST, PUT, DELETE). Altri parametri che possono essere aggiunti sono gli headers, per specificare il formato dei dati scambiati.

```
const response = await fetch('http://localhost:3001/admin/chats', {
  method: "GET",
  headers: {
    "Content-Type": "application/json",
    Accept: "application/json",
    "ngrok-skip-browser-warning": "true",
  },
});
```

Figura 3.1 Esempio di richiesta fetch() verso indirizzo locale

Nel contesto dell'applicazione dashboard di amministrazione, l'uso del metodo GET permette di recuperare i dati dal database, mentre gli headers specificano che il contenuto della risposta sarà in formato JSON.

3.1.2 Gestione delle Richieste su Diversi Ambienti

Nel corso dello sviluppo del progetto sono stati utilizzati due approcci per il recupero dei dati, a seconda dell'ambiente in cui l'applicazione veniva eseguita. Quando il database era in esecuzione sulla stessa macchina del frontend, le richieste venivano indirizzate a un *API_BASE_URL* configurato su *http://localhost:3000/*, permettendo al frontend di comunicare direttamente con il backend in locale. Quando invece il backend e il database erano ospitati su un'altra macchina, le richieste venivano effettuate a un indirizzo IP specifico, come ad esempio *http://10.200.200.5/*, garantendo un accesso remoto ai dati senza modificare la logica dell'applicazione.

Si noti come per semplificare la gestione degli ambienti di sviluppo e produzione, è stata utilizzata una configurazione dinamica dell'*API_BASE_URL*, che permetteva di cambiare automaticamente l'indirizzo del server a seconda dell'ambiente in cui l'applicazione veniva eseguita. Questo ha evitato la necessità di modificare manualmente ogni singola richiesta, facilitando il passaggio tra i diversi ambienti.

```
const response = await fetch(
  `${API_BASE_URL}/admin/chats/chat/${chatId}`,
  {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
      Accept: "application/json",
      "ngrok-skip-browser-warning": "true",
    },
  },
);
```

Figura 3.2 Esempio di configurazione dinamica di una fetch()

Un aspetto fondamentale di questa architettura è che entrambe le macchine si trovavano nella stessa sottorete con un prefisso di indirizzo IP comune, come *10.200.200.X*. Questo ha semplificato notevolmente la comunicazione tra le due componenti, evitando problemi legati all'accesso a risorse esterne e garantendo una connessione più stabile ed efficiente.

3.2 Elaborazione e Salvataggio dei Dati

Dopo aver effettuato la richiesta al backend il primo passo fondamentale è verificare che la risposta ottenuta sia valida. Ogni volta che viene eseguita una chiamata *fetch*, la risposta viene controllata attraverso la proprietà *ok* dell'oggetto *Response*. Se il valore della proprietà è *false* significa che la richiesta ha avuto un esito negativo, di conseguenza viene generato un errore con il relativo codice di stato. Questo approccio garantisce che eventuali problemi nella comunicazione con il backend vengano immediatamente individuati e gestiti in modo appropriato.

```
if (!response.ok) {  
  throw new Error(`Errore HTTP! Status: ${response.status}`);  
}
```

Figura 3.3 Esempio di gestione di un errore nella richiesta http

D'altra parte, una volta confermato che la risposta è valida, l'operatore *await* assicura che la *Promise* restituita dalla *fetch()* venga risolta prima di procedere con l'elaborazione dei dati. Una volta ottenuta la risposta, i dati vengono estratti in formato JSON grazie al metodo *.json()*.

A questo punto, i dati ricevuti vengono salvati all'interno di un oggetto il cui tipo è stato precedentemente definito attraverso un'interfaccia. Questo garantisce che ogni campo della risposta sia correttamente assegnato e che i dati vengano strutturati in maniera rigorosa, evitando errori dovuti a valori mancanti o formattazioni errate.

```
interface ChatAPIResponse {  
  content?: string[];  
  _id: string;  
  guest: Guest;  
  createdAt: string;  
  endedAt?: string;  
  messageCount: number;  
  feedback?: number;  
}
```

Figura 3.4 Esempio di un'interfaccia

In alcuni casi, i dati ricevuti dal backend necessitano di essere ulteriormente elaborati prima di poter essere utilizzati nel frontend. Durante lo sviluppo dell'applicazione in questione, è stato necessario estrarre e riorganizzare i dati per adattarli alla logica dell'applicazione, rendendoli più facilmente accessibili ai componenti che li utilizzano.

```
formatChatData = (item: ChatAPIResponse): ChatData => ({
  content: item.content,
  id: item._id || "N/A",
  guestId: item.guest._id,
  firstName: item.guest.name,
  lastName: item.guest.surname,
  email: item.guest.email,
  createdAt: new Date(item.createdAt),
  endedAt: item.endedAt ? new Date(item.endedAt) : "-",
  msgTot: item.messageCount,
  feedback: item.feedback,
});
```

Figura 3.5 Esempio di formattazione dei dati

Infine, il metodo restituisce l'oggetto contenente i dati elaborati e salvati, che potrà poi essere utilizzato dalle diverse parti dell'applicazione.

3.3 Utilizzo dei Dati nei Grafici

Nel contesto dello sviluppo del progetto, i dati recuperati dal backend vengono utilizzati per popolare i grafici presenti nell'interfaccia utente. Per rendere questa operazione efficiente e modulare, il *dataService* viene importato all'interno di ogni file *.tsx* che gestisce un grafico, permettendo di centralizzare la logica di recupero dei dati. L'acquisizione delle informazioni avviene sfruttando l'hook *useEffect*, che consente di eseguire la richiesta al backend non appena il componente viene montato.

```
useEffect(() => {
  const fetchChatContent = async () => {
    // Ottieni i dati da getRatings
    const data = await dataService.getRatings();
    setSatisfactionCounts(data ?? []);
  };

  fetchChatContent();
}, []);
```

Figura 3.6 Esempio di acquisizione dei dati in un componente

3.3.1 Assegnazione dei Dati alle Serie

A seconda della tipologia di grafico, i dati recuperati devono essere opportunamente assegnati alle serie che lo compongono. Ogni libreria di visualizzazione grafica richiede una struttura dati specifica, in questo caso i dati vengono impostati attraverso metodi dedicati. Questa operazione garantisce che il grafico sia aggiornato con i dati più recenti, consentendo una rappresentazione chiara e accurata delle informazioni. L'approccio adottato assicura modularità e facilità di aggiornamento, permettendo di gestire in modo scalabile l'integrazione dei dati nei vari componenti grafici.

```
series1.data.setAll(voteCounts);
```

Figura 3.7 Esempio di metodo dedicato per l'assegnazione dei dati

3.4 Esempio di Flusso dei Dati

Di seguito viene illustrato un esempio di flusso dei dati all'interno del progetto, dalla richiesta al backend fino alla visualizzazione nei grafici nel caso specifico del metodo *getRatings()* per il chart *SolidGaugeRate*.

Passaggio 1: Invio della richiesta al backend

Il metodo *getRatings* del *dataService* invia una richiesta HTTP GET all'endpoint */admin/ratings* per ottenere le valutazioni.

```
async getRatings(): Promise<number[]> {
  try {
    console.log("Avvio della richiesta...");
    const response = await fetch(`${API_BASE_URL}/admin/ratings`, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        "Accept": "application/json",
        "ngrok-skip-browser-warning": "true",
      },
    });
  }

  if (!response.ok) {
    console.error(
      `Errore HTTP: ${response.status} - ${response.statusText}`
    );
    const errorText = await response.text();

    console.error("Corpo della risposta:", errorText);
    throw new Error(`Errore HTTP: ${response.status}`);
  }

  const data: number[] = await response.json();
  console.log("Dati ricevuti:", data);
  return data;
} catch (error) {
  if (error instanceof Error) {
    console.error("Errore nella richiesta al backend:", error);
  }
  return []; // Restituisce un array vuoto in caso di errore
}
```

Figura 3.8 Metodo *getRatings()*

Passaggio 2: Recupero dei dati nel componente React

Nel componente del grafico, viene utilizzato `useEffect` per chiamare `getRatings()` al momento del montaggio

```
useEffect(() => {
  const fetchChatContent = async () => {
    // Ottieni i dati da getRatings
    const data = await dataService.getRatings();
    setSatisfactionCounts(data ?? []);
  };

  fetchChatContent();
}, []);
```

Figura 3.9 Acquisizione dei dati in *SolidGaugeRate*

Passaggio 3: Assegnazione dei dati al grafico

Quando `satisfactionCounts` viene aggiornato, i dati vengono assegnati alla serie del grafico e viene avviata l'animazione.

```
series2.data.setAll(voteCounts);

// Animazione
series1.appear(1000);
series2.appear(1000);
chart.appear(1000, 100);

return () => root.dispose();
}, [satisfactionCounts, theme, isDashboard]);
```

Figura 3.10 Assegnazione dei dati al grafico

4. CAPITOLO QUARTO: ANALISI DEI REQUISITI

In questo capitolo verrà esposta l'analisi dei requisiti del progetto dashboard di amministrazione.

Durante lo sviluppo di un progetto software, come in questo contesto, l'analisi dei requisiti è fondamentale perché consente di definire chiaramente quali sono gli obiettivi che si vogliono raggiungere e le aspettative del progetto prima di realizzarlo.

Migliore è l'analisi minore è il rischio di dover apportare modifiche costose durante lo sviluppo, con l'obiettivo di migliorare la qualità e rispondere meglio alle esigenze di un eventuale cliente.

4.1 Requisiti Funzionali

1. Interfaccia Utente Interattiva

- La dashboard di amministrazione deve presentare un'interfaccia utente responsiva, che permetta agli amministratori di interagire con i vari grafici in tempo reale.
- La UI deve adattarsi in modo fluido a diverse dimensioni dello schermo.
- L'interfaccia deve presentare una lista di categorie nelle quali è possibile navigare tramite cursore.

2. Gestione dei Widget nella Home

- Gli utenti devono essere in grado di poter personalizzare a loro piacimento il layout della schermata home della dashboard di amministrazione.
- Gli utenti devono poter essere in grado di trascinare e spostare i widget e lasciarli nel punto che preferiscono tramite azioni di drag & drop.

3. Gestione di Login

- Il sistema deve permettere agli utenti registrati di effettuare il login utilizzando le proprie credenziali.
- Dopo l'autenticazione, l'utente deve ricevere un token di accesso che ne consenta la validazione nelle richieste successive.

- Deve essere possibile effettuare il logout per terminare la sessione in modo sicuro.

4.2 Requisiti Non Funzionali

1. Performance

- Il sistema deve essere ottimizzato per garantire tempi di risposta rapidi e un'esperienza utente fluida.
- L'interfaccia deve essere reattiva e performante, gestendo efficientemente le risorse per evitare rallentamenti.

2. Usabilità

- L'interfaccia deve essere intuitiva e facile da comprendere, in modo che gli utenti possano sfruttarne ogni funzionalità senza difficoltà.

3. Manutenibilità

- Il codice deve essere scritto seguendo le migliori pratiche di sviluppo in modo da facilitare future manutenzioni e aggiornamenti.

4. Scalabilità

- La dashboard deve essere progettata per supportare volumi crescenti di dati senza comprometterne le prestazioni.

4.3 Casi D'Uso

Questa sezione descrive i casi d'uso relativi alle principali funzionalità dell'applicazione, definiti a partire dai requisiti individuati.

Caso d'uso 1: Login dell'Utente

Attore principale: Utente registrato

1. L'utente accede alla pagina di login.
2. Il sistema richiede email e password.
3. L'utente inserisce le credenziali e invia la richiesta.
4. Il sistema verifica i dati e genera un token di accesso.
5. L'utente viene reindirizzato alla dashboard.

Estensioni:

- (3a) Se le credenziali non sono corrette, il sistema mostra un messaggio di errore.

Caso d'uso 2: Interazione con l'Interfaccia Utente della Dashboard

Attore principale: Amministratore

1. L'amministratore accede alla dashboard.
2. Il sistema carica i dati e mostra i grafici interattivi.
3. L'amministratore può selezionare una categoria dal menu laterale.
4. I grafici vengono aggiornati in base alla categoria selezionata.
5. L'amministratore può filtrare e personalizzare la visualizzazione dei dati.

Caso d'uso 3: Personalizzazione della Dashboard con i Widget

Attore principale: Amministratore

1. L'amministratore accede alla dashboard.
2. Il sistema mostra i widget disponibili.
3. L'amministratore seleziona un widget e lo trascina nella posizione desiderata.
4. Il sistema aggiorna dinamicamente la disposizione dei widget.

Estensioni:

- (3b) Se il blocco layout (lucchetto) è attivato il sistema impedisce il movimento di un widget
- (5a) Se si verifica un errore nel salvataggio della disposizione il sistema ripristina la configurazione precedente.

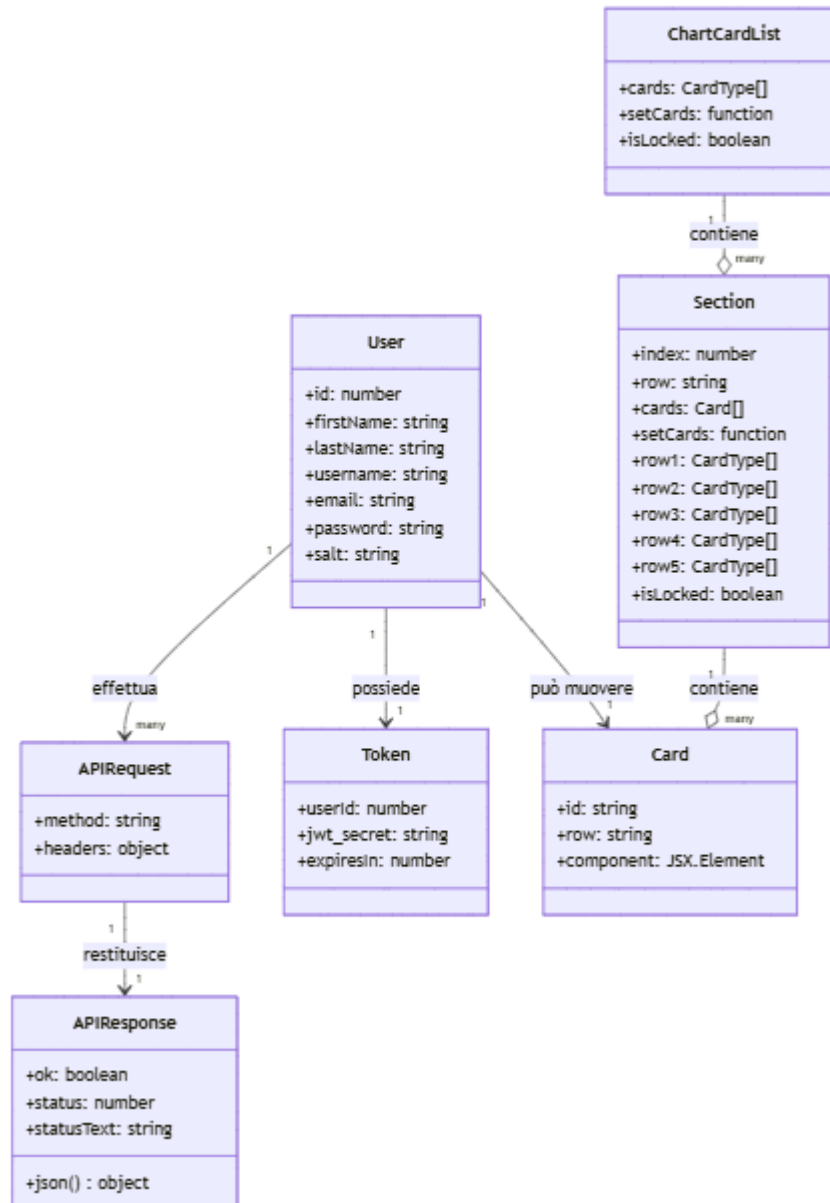
Caso d'uso 4: Logout dell'Utente

Attore primario: Utente registrato

1. L'utente accede alla dashboard.
2. L'utente seleziona l'opzione di logout.
3. Il sistema invalida il token di accesso.
4. L'utente viene reindirizzato alla pagina di login.

4.4 Modello di Dominio

La figura seguente rappresenta il modello di dominio che è stato sviluppato durante l'analisi, considerando tutti i casi d'uso ideati e riportati nella sezione precedente.



5. CAPITOLO QUINTO: FUNZIONALITÀ REALIZZATE

In questo capitolo vengono presentate le funzionalità implementate nel progetto, illustrando le operazioni eseguite per soddisfare i requisiti definiti nel capitolo precedente. Inoltre, verrà spiegato come le tecnologie descritte nel secondo capitolo siano state utilizzate per sviluppare le diverse componenti del sistema.

5.1 Architettura della Dashboard

Il sistema sviluppato si compone di due principali componenti, un backend e un frontend.

Lo scopo del backend è stato affrontato nel terzo capitolo. Come visto, esso ricopre il ruolo di soddisfare le richieste provenienti dal frontend per ottenere i dati e presentarli sotto forma di grafici. Inoltre gestisce l'intero processo di autenticazione per il login di un amministratore.

Il frontend rappresenta l'interfaccia utente vera e propria ed è suddivisibile in due macro-componenti: la pagina di login, che verrà trattata nella sezione 5.6, e la dashboard vera e propria. Quest'ultima si compone di tre elementi principali: il menu laterale, la topbar e la homepage, che verranno analizzati più avanti.

5.2 Home Page

La homepage dell'applicazione è organizzata in un layout flessibile che permette di gestire e personalizzare la disposizione dei widget. Ogni widget rappresenta un elemento informativo, come un grafico o uno statbox, e può essere posizionato su una delle cinque righe disponibili. Questa struttura garantisce una gestione chiara e ordinata delle informazioni, consentendo agli utenti di personalizzare la loro esperienza in base alle esigenze specifiche.

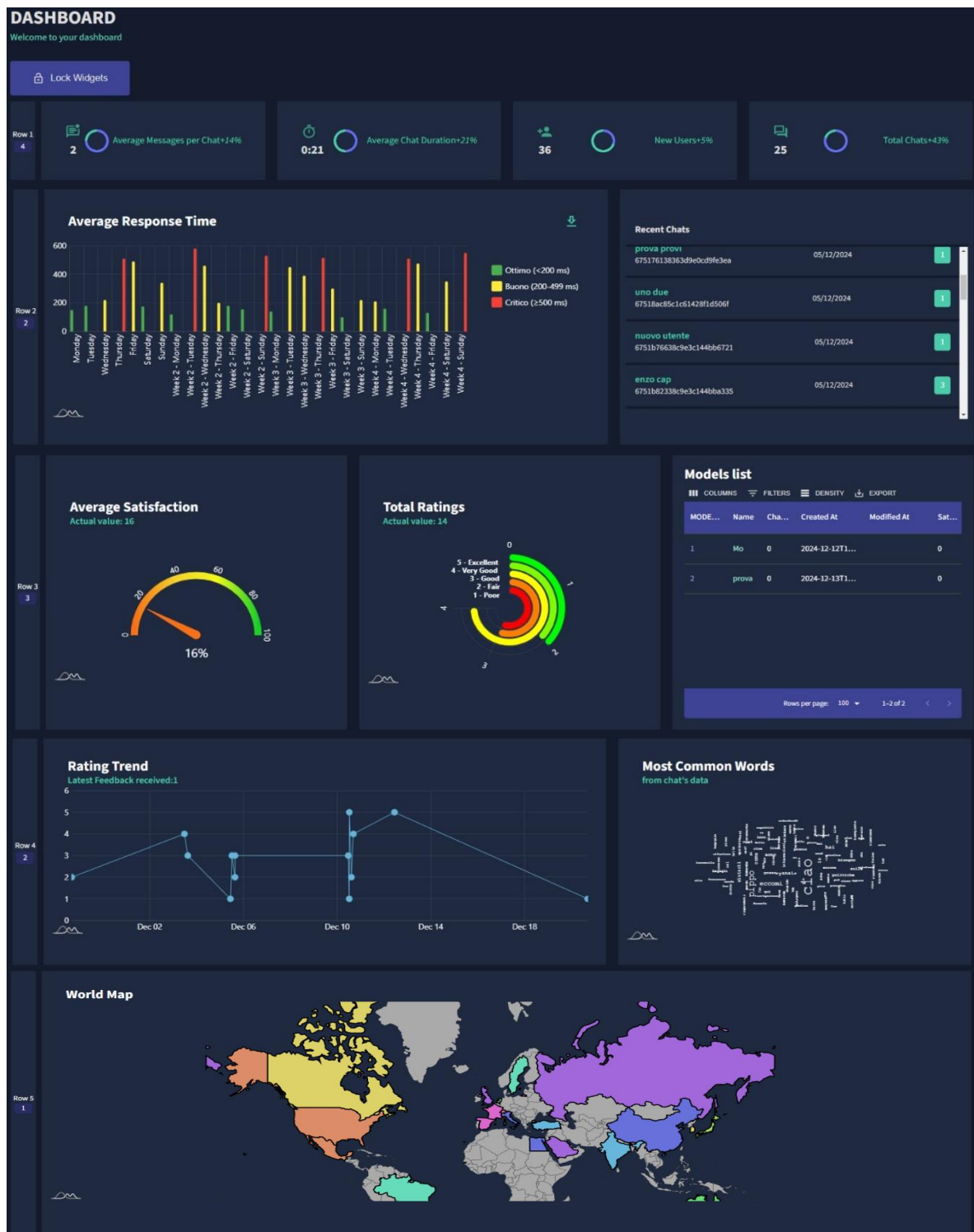


Figura 5.1 Home Page e i suoi grafici

Per mantenere il tracciamento della posizione di ogni widget, è necessario un meccanismo che permetta di aggiornare dinamicamente lo stato delle righe e delle carte presenti in esse. Questo viene gestito attraverso un macrocomponente principale, responsabile della gestione dello stato e della logica che consente lo spostamento dei widget.

5.2.1 Gestione dei Widget tramite Drag & Drop

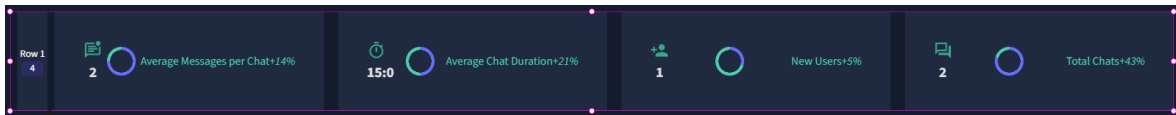


Figura 5.2 Esempio di una sezione

Il sistema di drag & drop è stato implementato utilizzando la libreria React DnD, che fornisce un meccanismo efficace per gestire il trascinamento e il rilascio degli elementi all'interno della homepage. Questo processo avviene attraverso due componenti chiave, le *section* e le *card*.

5.2.2 Componente Section come Area di Drop

Ogni riga della homepage rappresenta una sezione (Section), che funge da contenitore per i widget e supporta il drop degli elementi trascinati. Per implementare questa funzionalità la sezione utilizza l'hook *useDrop*, che permette di definire il comportamento dell'area di rilascio.

```
const [{ isOver }, drop] = useDrop(() => ({
  accept: "card",
  canDrop: () => !isLocked, // Impedisce il drop se isLocked è true
  drop: (item: CardType, monitor) => {
    if (!isLocked) {
      addItemToSection(item.id);
    }
  },
  collect: (monitor) => ({
    isOver: !!monitor.isOver(),
  }),
}));
```

Figura 5.3 Funzione che gestisce il drop

Questa funzione consente di definire che la sezione può accettare solo elementi di tipo *card*, impedire il rilascio quando il blocco *isLocked* è attivo e infine tracciare visivamente quando un elemento è sopra l'area di drop.

Una volta rilasciato il widget, viene chiamata la funzione *addItemToSection*, che aggiorna lo stato globale delle carte, modificando la riga di appartenenza del widget spostato.

```
const addItemToSection = (id: string) => {
  setCards((prevCards: CardType[]) => {
    const modifiedCard = prevCards.map((card) => {
      if (card.id === id) {
        return { ...card, row: row };
      }
      return card;
    });
    return modifiedCard;
  });
};
```

Figura 5.4 Funzione addItemToSection

L'utilizzo di *useDrop* garantisce che ogni rilascio venga gestito in modo reattivo ed efficiente, permettendo di rispettare i requisiti di personalizzazione e gestione dinamica della dashboard.

5.2.3 Componente Card come Elemento Trascinabile

Affinché un widget possa essere spostato, deve essere definito come elemento trascinabile utilizzando *useDrag*. Questo hook permette di assegnare una proprietà *draggable* alle carte tracciandone lo stato.

```
const [{ isDragging }, drag] = useDrag(() => ({
  type: "card",
  item: { id: card.id },
  canDrag: () => !isLocked, // Impedisce il drag se isLocked è true
  collect: (monitor) => ({
    isDragging: !!monitor.isDragging(),
  }),
}));
```

Figura 5.5 Funzione che gestisce il drag

Questa funzione definisce il tipo di elemento che può essere trascinato aggiungendo un controllo che impedisce il trascinamento se il blocco *isLocked* è attivo. Infine, monitora se l'elemento è attualmente in stato di trascinamento.

➤ Scelte Progettuali e Benefici

Tutte le scelte progettuali adottate nelle sezioni precedenti hanno permesso di soddisfare i requisiti del quarto capitolo e del caso d'uso numero 3. Inoltre, per mantenere un aspetto armonioso e garantire una chiara leggibilità, la dimensione delle card e dei grafici viene ricalcolata automaticamente a ogni modifica, assicurando proporzioni ottimali indipendentemente dalla posizione dei widget.

5.3 Side Bar

La Side Bar rappresenta il menu laterale di navigazione dell'applicazione. Si tratta di un pannello a scorrimento che offre un accesso rapido e strutturato a tutte le risorse disponibili.

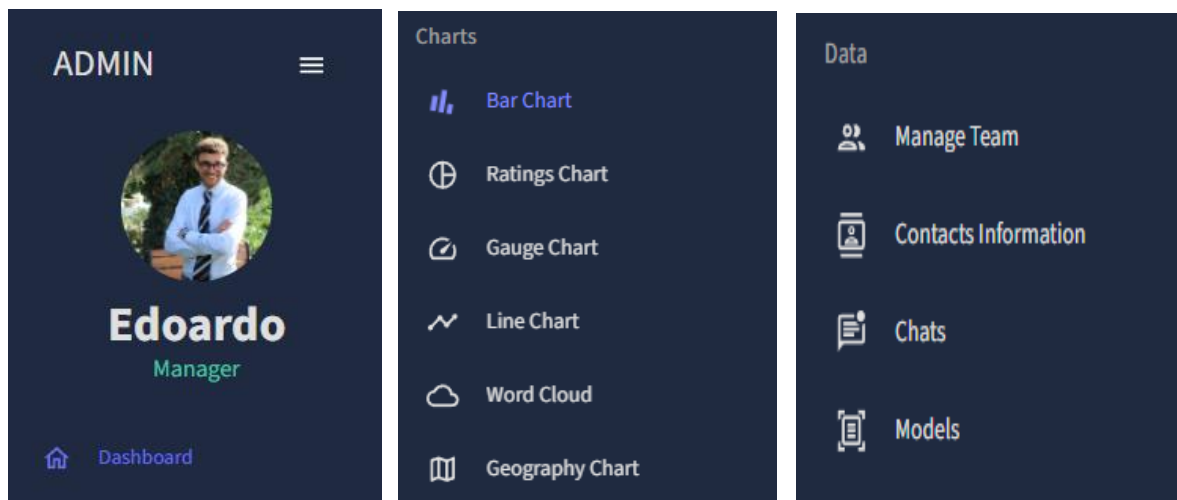


Figura 5.6 SideBar diviso

➤ Grafico a Barre: Bar Chart

Il grafico a barre è stato utilizzato per rappresentare il tempo medio di risposta (*average response time*), un parametro che misura il tempo impiegato dal chatbot per rispondere a una richiesta. I dati vengono suddivisi in categorie e associati a colori differenti per indicare la qualità del valore riscontrato.

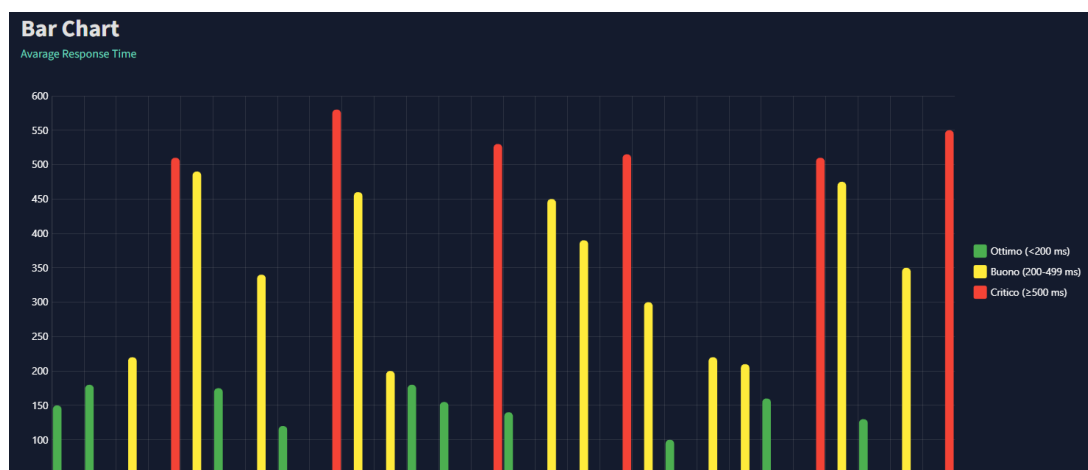


Figura 5.7 Grafico a Barre

➤ Grafico delle Valutazioni: Ratings Chart

Il grafico delle valutazioni è stato utilizzato per visualizzare la distribuzione delle valutazioni ricevute. Il chart utilizza un array di dati, dove l'indice rappresenta la valutazione e il valore associato indica il numero di occorrenze di tale valutazione.



Figura 5.8 Esempio Grafico delle Valutazioni

➤ Grafico a tachimetro: Gauge Chart

Il grafico a tachimetro è stato utilizzato per rappresentare la percentuale di soddisfazione basata sui feedback ricevuti dagli utenti del chatbot. I dati sono memorizzati in un oggetto che contiene il voto (da 1 a 5) e la data di registrazione. In breve, il sistema calcola il numero totale di valutazioni e genera una stima percentuale della soddisfazione.



Figura 5.9 Esempio Grafico a Tachimetro

➤ Grafico a Linee: Line Chart

Il grafico a linea utilizza un oggetto mappato che associa una data ai relativi valori di feedback, tracciando un trend nel tempo. Questo permette di monitorare l'andamento della qualità delle risposte del chatbot e identificare eventuali variazioni.



Figura 5.10 Esempio Grafico a Linea

➤ Nuvola delle Parole: Word Cloud

La word cloud rappresenta una nuvola di parole basata sulla loro frequenza nelle conversazioni con il chatbot. Passando il cursore su una parola, è possibile vedere il numero di occorrenze. Questo strumento aiuta a individuare gli argomenti più discussi e richiesti dagli utenti.



Figura 5.11 Esempio Nuvola di Parole

➤ Grafico Geografico: Geography Chart

Il grafico geografico è una mappa interattiva che mostra la distribuzione globale dei clienti. Ogni area della mappa indica la presenza e il numero di utenti che hanno interagito con il chatbot.



Figura 5.12 Esempio Grafico Geografico

➤ Lista dei Contatti

La lista dei contatti mostra un elenco di utenti con le relative informazioni principali: ID, nome, cognome, email e data di nascita. Questo consente di gestire facilmente i dati degli utenti, facilitando l’accesso rapido alle informazioni necessarie.

COLUMNS FILTERS DENSITY EXPORT				
ID	Name	Surname	Email Address	Birthdate
6745cf5f25b73815bfa580a9	fra	font	fra@gmail.com	2002-01-01T00:00:00.000Z
6745cfa025b73815bfa580bb	verro	anim	verro@gmail.com	2004-05-01T00:00:00.000Z
6745cfe225b73815bfa58297	cici	perretta	cici@gmail.com	2008-08-08T00:00:00.000Z

Figura 5.13 Esempio Lista dei Contatti

➤ Lista delle Chat e Cronologia

Questa funzionalità consente agli amministratori di consultare lo storico delle conversazioni, visualizzando informazioni sugli utenti che le hanno avviate e sui messaggi scambiati. Questa sezione, come altre, contiene dati sensibili, la cui gestione verrà approfondita nelle sezioni successive dedicate alla sicurezza e alla privacy.

COLUMNS FILTERS DENSITY EXPORT						
CHAT ID	Email	FirstName	LastName	Created At ↑	Ended At	Message Count
67530b2645c0266779e9496c	john.doe@example.com	John	Doe	Fri Dec 06 2024 ...	Fri Dec 06 2024...	3
67530b2645c0266779e9496b	john.doe@example.com	John	Doe	Fri Dec 06 2024 ...	-	1

Figura 5.14 Lista delle chat

Chat ID: 6751b82338c9e3c144bba335

User

quanto deve esser egrande il bagaglio a mano per voli ryanair?

Bot

Con Ryanair, puoi portare gratuitamente un piccolo bagaglio a mano di dimensioni massime 40 x 20 x 25 cm, da posizionare sotto il sedile di fronte a te. Se acquisti l'opzione Priorità e 2 bagagli a mano, puoi portare anche un secondo bagaglio fino a 55 x 40 x 20 cm e 10 kg, da riporre nella cappelliera.

User

grazie mille

Chat Details

Chat ID: 6751b82338c9e3c144bba335

Created At: 05/12/2024, 15:26:43

User ID: 674f0fee53b173e000e30e1c

Ended At: 05/12/2024, 15:27:18

First Name: enzo

Total Messages: 3

Last Name: can

Figura 5.15 Esempio cronologia di una chat

5.4 Top Bar

La Top Bar, come si deduce dal nome inglese, si trova nella parte alta dell'applicazione. Al suo interno presenta una barra di ricerca che consente di trovare facilmente una risorsa attraverso parole chiave. Inoltre, è presente un'icona che ha il compito di gestire il tema dell'applicazione (chiaro/scuro) e infine un'icona che gestisce la funzionalità di logout, che verrà approfondita più avanti.

➤ Gestione del Tema

La gestione del tema è centralizzata nel file *theme.tsx*, che si occupa di definire i colori di tutta l'applicazione, regolando sfondi e testi in base alla modalità selezionata. L'utilizzo dell'hook *useTheme* consente di utilizzare i temi in ogni componente dell'applicazione. In questo modo, è stato possibile manipolare i colori in base al tema corrente, utilizzando scale di tonalità standardizzate in *theme.tsx*.

```
const wichtheme =
  theme.palette.mode === "dark"
    ? am5themes_Dark.new(root)
    : am5themes_Animated.new(root);
root.setThemes([am5themes_Animated.new(root), wichtheme]);
```

Figura 5.16 Esempio di gestione del tema in un componente

➤ Gestione delle Ricerche

L'implementazione utilizza un array di dati che associa ogni parola chiave a un determinato percorso di destinazione della risorsa associata. Quando l'utente digita un termine nella barra di ricerca, il sistema filtra i risultati e li mostra in lista. L'utente può selezionare un risultato per essere reindirizzato alla relativa pagina.

```
const handleSearch = (e: React.ChangeEvent<HTMLInputElement>) => {
  const value = e.target.value;
  setInput(value);

  if (value.trim()) {
    const filtered = mockData.filter((item) =>
      item.name.toLowerCase().includes(value.toLowerCase())
    );
    setResults(filtered);
  } else {
    setResults([]);
  }
};
```

Figura 5.17 Funzione che gestisce la ricerca

5.5 Sicurezza e Gestione della Privacy

La sicurezza e la protezione della privacy sono aspetti fondamentali nella progettazione della dashboard di amministrazione. Il primo passo per accedere al sistema avviene attraverso una pagina di login oppure di registrazione, che verranno approfondite di seguito.

Quando si parla di sicurezza, è essenziale adottare tecniche di crittografia e hashing per proteggere le credenziali degli utenti nel database. In questo modo si riduce al minimo il rischio di accessi non autorizzati e furti di dati.

5.5.1 Crittografia della Email

L'email dell'utente è un'informazione sensibile che potrebbe essere sfruttata da eventuali malintenzionati per eseguire attacchi di phishing o compromettere l'identità dell'utente. Per evitare che venga salvata in chiaro nel database, essa viene crittografata utilizzando l'algoritmo **AES-256-GCM**. Questo algoritmo offre un alto livello di sicurezza grazie alla chiave segreta di 256 bit e al vettore di inizializzazione (IV), entrambi necessari per eseguire la crittografia e la successiva decrittografia dell'informazione.

```
encryptEmail(email: string): string {  
  const cipher = crypto.createCipheriv(  
    algorithm,  
    Buffer.from(secretKey, "hex"),  
    Buffer.from(iv, "hex")  
  );  
  let encrypted = cipher.update(email, "utf-8", "hex");  
  encrypted += cipher.final("hex");  
  return encrypted;  
}
```

Figura 5.1 Funzione per criptare l'email

Questa funzione utilizza l'algoritmo AES-256-GCM per trasformare l'email in un formato cifrato. L'IV (vettore di inizializzazione) garantisce che ogni crittografia sia univoca, mentre la chiave segreta assicura che solo i sistemi autorizzati possano eseguire la decrittografia.

5.5.2 Decrittografia della Email

Quando l'email deve essere recuperata, il sistema utilizza la funzione di decrittografia, che sfrutta la stessa chiave segreta e lo stesso IV.

```
decryptEmail(encryptedEmail: string): string {  
  const decipher = crypto.createDecipheriv(  
    algorithm,  
    Buffer.from(secretKey, "hex"),  
    Buffer.from(iv, "hex")  
  );  
  let decrypted = decipher.update(encryptedEmail, "hex", "utf-8");  
  decrypted += decipher.final("utf-8");  
  return decrypted;  
}
```

Figura 5.19 Funzione per decriptare l'email

Questa funzione permette di ripristinare l'email originale solo se si dispone della chiave corretta e dell'IV utilizzato in fase di crittografia, garantendo così che solo i sistemi autorizzati possano accedere ai dati reali.

5.5.3 Hashing della Password

A differenza dell'email, invece della crittografia per le password viene applicato un meccanismo di *hashing*. L'*hashing* è un'operazione unidirezionale, il che significa che non è possibile risalire alla password originale a partire dal suo hash. Nel caso di questo sistema, viene utilizzata una funzione di hashing basata sull'algoritmo SHA-512, combinato con un salt unico per ogni utente.

```
hashingFunction(param: string, salt: string): string {  
  return createHmac("sha512", salt).update(param).digest("hex");  
}
```

Figura 5.20 Funzione di hashing

Questa funzione di hashing, implementata tramite il modulo `crypto` di Node.js, utilizza una chiave segreta (salt) per creare un HMAC (Hash-based Message Authentication Code) sicuro. In breve la password dell'utente viene *salata* e poi sottoposta all'algoritmo SHA-512, che produce un valore hash, il quale viene poi memorizzato nel database.

Per raggiungere l'obiettivo di garantire sicurezza e privacy nell'applicazione, è stata utilizzata la libreria *crypto* per implementare una funzione in grado di generare un *salt* composto da 16 byte casuali, assicurando così la protezione dei dati sensibili.

```
generateSalt() {  
  return crypto.randomBytes(16).toString("hex");  
}
```

Figura 5.21 Funzione che genera il salt

5.6 Gestione del Login

La gestione del login è una componente fondamentale per garantire la sicurezza e l'accesso protetto all'applicazione. Quando un amministratore registrato accede alla pagina di login, viene richiesto di inserire le proprie credenziali, ossia l'username e la password.

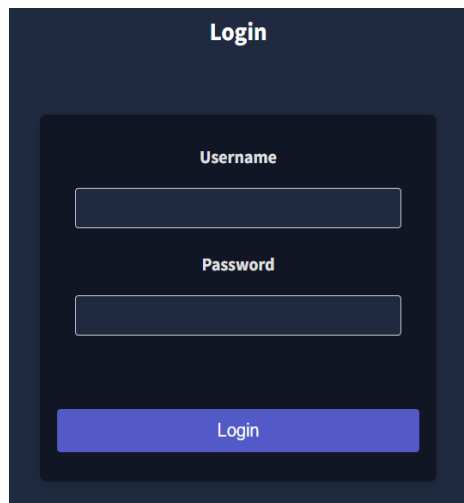
The image shows a dark-themed login interface. At the top, the word "Login" is centered in a light blue font. Below it, there is a dark gray rectangular box containing two input fields. The first field is labeled "Username" in a light blue font, and the second field is labeled "Password" in a light blue font. Both fields have a light blue border and are currently empty. Below the password field, there is a solid blue button with the word "Login" written in white text.

Figura 5.22 Pagina di Login

5.6.1 Verifica delle Credenziali

Una volta inviata la richiesta, il sistema verifica la correttezza dei dati inseriti confrontando la password fornita con quella memorizzata nel database. Come spiegato nella sezione precedente, il processo di hashing è unidirezionale. In altre parole, non si può eseguire un processo inverso per ricavare la password e verificare se essa corrisponde a quella immessa dall'amministratore. Quando l'amministratore effettua il login, la password immessa viene sottoposta alla stessa funzione di hashing, utilizzando il salt associato all'utente. Il risultato ottenuto viene quindi confrontato con l'hash memorizzato nel database. Se i valori corrispondono, l'autenticazione ha successo.

In caso di credenziali errate, il sistema restituisce un messaggio di errore generico, senza rivelare informazioni specifiche, come ad esempio quale delle credenziali sia errata. Questo approccio aumenta la sicurezza, impedendo che informazioni sensibili vengano utilizzate da eventuali attaccanti.

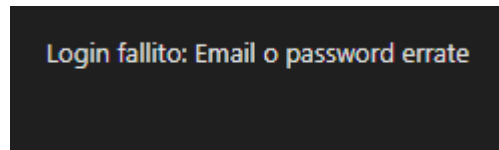


Figura 5.23 Messaggio di errore durante il Login

5.6.2 Gestione della Sessione

Una volta che l'utente ha effettuato correttamente il login, il sistema genera un *token* JWT che garantisce un accesso sicuro e protetto alle risorse sensibili dell'applicazione. Questo token funge da chiave digitale che consente di autenticare l'utente nelle richieste future senza dover ripetere la procedura di login per ogni operazione. Il token contiene informazioni codificate riguardanti l'utente e la sua sessione e viene inviato al client che lo memorizza nel *localStorage* o nei *cookies*.

Il vantaggio principale di utilizzare un JWT è che il server non ha bisogno di mantenere uno stato persistente tra le richieste. Ogni volta che il client effettua una richiesta, invia il token con la chiamata, che viene poi verificato dal server per confermare l'identità dell'utente e autorizzare l'accesso alle risorse protette.

5.7 Gestione del Logout

Per quanto riguarda la sicurezza dell'intera sessione, il sistema realizzato prevede anche una funzionalità di logout che invalida il token JWT. Quando l'utente decide di uscire o il token scade, il sistema rende inutilizzabile il token, impedendo che venga riutilizzato per accedere all'applicazione in futuro.

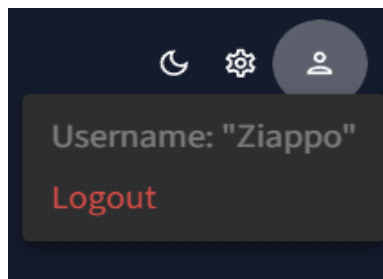


Figura 5.24 Finestra di Logout

Tutte le scelte architetturali e tecnologiche adottate sono in linea con i requisiti di sicurezza e privacy delineati nel capitolo precedente. L'uso del JWT risponde alla necessità di avere una gestione senza stato delle sessioni, riducendo il carico sul server e migliorando le performance, mentre il logout assicura che l'accesso sia sempre protetto, anche dopo la fine della sessione dell'utente.

5.8 State Management: Aggiornamento dei Grafici

Per garantire che i grafici mostrino dati dinamici e aggiornati in tempo reale, è essenziale gestire correttamente lo stato e il ciclo di vita dei componenti. Questo avviene attraverso l'uso degli hook di React, in particolare *useState* e *useEffect*.

➤ Gestione dello Stato dei Dati

In ogni grafico viene definita una variabile di stato per memorizzare i dati utilizzando *useState*. Ad esempio, nel caso della Nuvola di Parole, lo stato viene inizializzato come segue:

```
const [data, setData] = useState<MessageData[] | undefined>(undefined);
```

Figura 5.25 Esempio di inizializzazione dello stato di un grafico

i dati vengono successivamente raccolti tramite una funzione specifica, come dimostrato nel terzo capitolo.

➤ Aggiornamento del Grafico

Dopo aver recuperato i dati, un secondo *useEffect* si occupa di inizializzare il grafico e aggiornarlo ogni volta che lo stato cambia. In particolare, L'array delle dipendenze `[data, theme, isDashboard]` garantisce che il grafico venga ricreato ogni volta che cambiano i dati o il tema, mantenendo le informazioni sempre aggiornate e coerenti con il contesto visivo.

In questo modo è stato possibile modellare lo stato per ogni grafico, rispettando i requisiti non funzionali menzionati nel quarto capitolo.

6. CAPITOLO SESTO: VERIFICHE FUNZIONALI

In questo capitolo verranno presentate le verifiche funzionali effettuate per garantire che le funzionalità implementate rispettino i requisiti funzionali e non funzionali definiti in fase di progettazione.

6.1 Test su Autenticazione e Sicurezza

Per garantire la sicurezza e la correttezza del processo di autenticazione, sono stati implementati test dettagliati utilizzando *Jest* e *React Testing Library*. Questi test verificano la robustezza del sistema di login, simulando diversi scenari di autenticazione e validando la gestione della sessione utente tramite *sessionStorage*.

6.1.1 Verifica della Validazione dei Campi

Sono stati eseguiti test specifici per verificare il comportamento del sistema in caso di dati mancanti. Se l'utente tenta di effettuare il login senza inserire una password, viene mostrato un messaggio di errore che segnala la necessità di compilare il campo. Allo stesso modo, se il nome utente non viene inserito prima di inviare il form, il sistema avvisa l'utente indicando che il campo è obbligatorio.

```
test("renders an error message if password is empty", async () => { ...
});
test("renders an error message if username is empty", async () => { ...
});
```

Figura 6.1 Esempio di test per validazione dei campi

6.1.2 Gestione di Login Falliti

È stato simulato un login fallito attraverso il mock della funzione *fetch*, restituendo una risposta 401 Unauthorized. Il sistema, in questi casi, mostra un messaggio di errore all'utente sotto forma di alert. Inoltre, viene verificato che un eventuale errore nella richiesta venga correttamente loggato nella console per facilitare il debugging.

```
test("renders an alert if the login was unsuccessful", async () => { ...
});
```

Figura 6.2 Esempio di test per login falliti

6.1.3 Gestione della Sessione

Per testare il login avvenuto con successo, è stato effettuato un mock della risposta della *fetch* restituendo un token di autenticazione, verificando che dopo l'inserimento

corretto delle credenziali il token venga salvato in sessionStorage per garantire la persistenza della sessione dell'utente e che il sistema gestisca correttamente la risposta 200 OK, confermando l'avvenuta autenticazione.

```
it("should set token in sessionStorage on successful login", async () => { ...
});
```

Figura 6.3 Esempio di test per la sessione

6.2 Test di Navigazione e Usabilità

La progettazione dell'interfaccia e la gestione dei widget sono state sviluppate con l'obiettivo di garantire un'esperienza utente fluida e intuitiva.

L'uso di una struttura modulare ha favorito la flessibilità, consentendo la riutilizzabilità dei componenti senza compromettere la navigazione. Inoltre, l'adozione di React ha permesso di mantenere un'interfaccia reattiva e dinamica, evitando ricaricamenti della pagina e garantendo transizioni fluide tra le sezioni dell'applicazione.

Dal punto di vista dell'accessibilità, la progettazione ha tenuto conto di principi di usabilità, assicurando che l'interazione con i widget fosse intuitiva anche per utenti meno esperti. I test condotti hanno confermato che il sistema di gestione dei widget è chiaro e immediato, riducendo la necessità di operazioni complesse e migliorando l'efficienza nell'utilizzo dell'applicazione.

6.3 Test di Aggiornamento e Correttezza dei Dati

Per garantire l'affidabilità della comunicazione tra il front-end e il back-end, sono stati eseguiti test approfonditi su tutti i metodi responsabili dell'esecuzione delle fetch per ottenere e aggiornare i dati. Ogni test aveva lo scopo di verificare che le funzioni di recupero restituissero i dati attesi in modo corretto, assicurando che le informazioni ricevute fossero coerenti con quelle memorizzate nel database.

```
it("should return correct duration for a chat lasting exactly one hour", () => { ...
});
it("Should handle dates spanning across midnight correctly", () => { ...
});
it("should throw an error when start date is after end date", () => { ...
});
it("should return '00:00:00' for start and end dates that are the same", () => { ...
});
it("should throw an error for invalid date formats", () => { ...
});
```

Figura 6.4 Esempio test per la verifica dei dati

7. CAPITOLO SETTIMO: CONCLUSIONI E SVILUPPI FUTURI

L'esperienza maturata lavorando con un chatbot ha evidenziato come, se opportunamente addestrato, esso sia in grado di rispondere efficacemente a domande di diversa complessità. La sua preparazione e la qualità delle informazioni fornite incidono direttamente sulla rapidità con cui un utente può ottenere le risposte desiderate. Questo aspetto assume particolare rilevanza in ambito aziendale, dove ridurre i tempi di ricerca delle informazioni significa migliorare l'efficienza operativa e aumentare la soddisfazione del cliente.

Un chatbot ben progettato elimina la necessità per l'utente di esplorare autonomamente un sito web alla ricerca di informazioni, semplificando l'interazione e ottimizzando il tempo impiegato. Questa facilità di accesso ai dati si traduce in un vantaggio strategico per le aziende, che possono garantire un supporto clienti più rapido ed efficace. La frase "il tempo è denaro" trova qui una concreta applicazione, la rapidità nel risolvere le richieste degli utenti si riflette in una maggiore fidelizzazione e potenzialmente, in un incremento delle vendite e del ritorno economico.

Un altro aspetto fondamentale emerso da questo progetto è il valore della *dashboard di amministrazione*. Essa non è solo un'interfaccia di controllo, ma un potente strumento per la visualizzazione e l'analisi di dati complessi. La sua implementazione consente di interpretare in modo intuitivo le informazioni raccolte dalle interazioni con il chatbot, facilitando decisioni aziendali più consapevoli. Un sistema ben progettato permette di individuare pattern di comportamento degli utenti, evidenziare eventuali criticità e migliorare costantemente il servizio di supporto.

Dal punto di vista tecnologico, il progetto ha sfruttato soluzioni avanzate, rendendo però accessibili le informazioni anche a utenti non tecnici. La dashboard è stata concepita come uno strumento intuitivo che permette di accedere a dati rilevanti senza la necessità di comprendere nel dettaglio i processi sottostanti. Questo approccio democratizza l'accesso alle informazioni e consente a diverse figure aziendali di prendere decisioni mirate basate su dati concreti.

➤ Prospettive future

Il futuro dei chatbot si prospetta ricco di opportunità, grazie ai progressi costanti nell'intelligenza artificiale generativa. L'integrazione con strumenti innovativi, come il riconoscimento vocale e la realtà aumentata, potrà rendere le interazioni con gli umani ancora più fluide e coinvolgenti. Oltre al miglioramento della comprensione del linguaggio naturale, l'evoluzione dei chatbot punterà a una maggiore consapevolezza del contesto culturale e sociale, consentendo conversazioni più naturali e adattabili alle esigenze degli utenti.

Un ulteriore sviluppo riguarda l'uso dell'intelligenza artificiale per analizzare e prevedere le esigenze degli utenti prima ancora che essi pongano una domanda. Questo potrebbe rivoluzionare il servizio clienti, trasformandolo da reattivo a proattivo.

Dal punto di vista aziendale, i chatbot non saranno solo strumenti di assistenza, ma diventeranno vere e proprie risorse strategiche per ottimizzare i processi interni e migliorare l'esperienza utente. La raccolta e l'analisi dei dati delle interazioni permetteranno di affinare le strategie di marketing, personalizzare le offerte e incrementare il coinvolgimento degli utenti.

In conclusione, il progetto sviluppato ha dimostrato come i chatbot possano rappresentare una risorsa fondamentale per le aziende, migliorando l'efficienza del supporto clienti e favorendo decisioni data-driven. La loro evoluzione continua porterà a interazioni sempre più avanzate e integrate con altre tecnologie, aprendo nuove opportunità per il futuro dell'intelligenza artificiale nel settore dei servizi digitali.

BIBLIOGRAFIA

- [1] TypeScript, <https://www.typescriptlang.org/>
- [2] React, <https://react.dev/reference/react>
- [3] MongoDB, <https://www.mongodb.com/docs/manual/>
- [4] MariaDB, <https://mariadb.org/documentation/>
- [5] amCharts, <https://www.amcharts.com/docs/v5/>
- [6] TypeORM, <https://typeorm.io/>
- [7] Prospettive future sui chatbot, <https://www.agendadigitale.eu/cultura-digitale/competenze-digitali/chatbot-cosi-con-lai-gen-si-creano-interlocutori-sofisticati/>
- [8] Global trend and analysis 2019, <https://chatbotsmagazine.com/chatbot-report-2019-global-trends-and-analysis-a487afec05b>