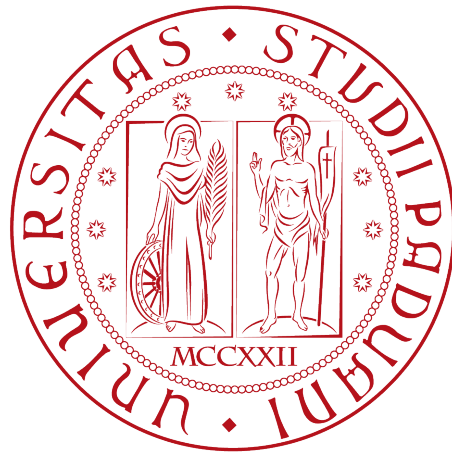


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA “TULLIO LEVI-CIVITA”

CORSO DI LAUREA IN INFORMATICA



**Integrazione di servizi di Intelligenza
Artificiale in una piattaforma web**

Tesi di laurea

Relatore

Prof. Luigi De Giovanni

Laureando

Zhen Wei Zheng

Matricola

1229141

ANNO ACCADEMICO 2022-2023

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa duecentottanta ore, dal laureando Zhen Wei Zheng presso l'azienda RiskApp s.r.l.

L'obiettivo principale è integrare ChatGPT, sviluppato da OpenAI, all'interno della piattaforma web aziendale. Tale integrazione permette agli utenti di eseguire ricerche testuali avanzate, offrendo la possibilità di filtrare rapidamente e con precisione i dati presenti in specifiche tabelle. Il risultato principale di questa sinergia è rappresentato dal notevole risparmio di tempo per gli utenti quando si tratta di filtrare tabelle di grandi dimensioni.

Nella tesi vengono esposte con dettaglio le diverse fasi affrontate, partendo dall'analisi dei requisiti, per poi proseguire con le fasi di progettazione e di codifica. Infine, si approfondisce l'importante fase di verifica e validazione delle funzionalità implementate.

Convenzioni tipografiche

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*^[g];
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*;
- nei codici, quando compare "[...]", ciò indica che sono state omesse porzioni di codice non rilevanti per il contesto o per la chiarezza della spiegazione.

“Life is really simple, but we insist on making it complicated”

— Confucius

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Luigi De Giovanni, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, Settembre 2023

Zhen Wei Zheng

Indice

1	Introduzione	1
1.1	L'azienda	1
1.2	Introduzione al progetto	1
1.3	Organizzazione del testo	2
2	Descrizione dello stage	3
2.1	Prodotto da ottenere	3
2.2	Pianificazione delle attività	4
2.3	Analisi preventiva dei rischi	4
3	Analisi dei requisiti	7
3.1	Casi d'uso	7
3.2	Tracciamento dei requisiti	14
4	Progettazione e codifica	17
4.1	Tecnologie e strumenti	17
4.1.1	Strumenti utilizzati	17
4.1.2	Tecnologie utilizzare	19
4.2	Architettura preesistente	23
4.2.1	Struttura e tipi di dati della tabella	23
4.2.2	Integrazione del nuovo componente	25
4.3	Architettura del nuovo componente	26
4.3.1	Frontend	27
4.3.2	Backend	28
4.3.3	Libreria Redux Toolkit	29
4.4	Codifica	30
4.4.1	Componenti React	30
4.4.2	Modale per il filtraggio testuale	31
4.4.3	Riconoscimento vocale	33
4.4.4	Invio richiesta di filtraggio e ricezione risultati	34
4.4.5	Visualizzazione riepilogo del filtraggio	36
4.4.6	API di OpenAI	38
5	Verifica e validazione	45
5.1	Verifica	45
5.1.1	Addestramento del chatbot	45
5.1.2	Controllo dei Token	46
5.1.3	Controllo delle risposte	46
5.2	Analisi statica e dinamica	46

5.3	Validazione	47
6	Conclusioni	49
6.1	Consuntivo finale	49
6.2	Raggiungimento degli obiettivi	50
6.3	Soddisfacimento dei requisiti	50
6.4	Valutazione personale	52
	Acronimi e abbreviazioni	53
	Glossario	55
	Riferimenti bibliografici	57

Elenco delle figure

1.1	Logo RiskApp	1
3.1	Use case: scenario principale	8
3.2	Use case: UC3-Visualizza modale di filtraggio	9
3.3	Use case: UC3.6-Controllo disponibilità del servizio	10
3.4	Use case: UC4-Visualizza riepilogo query di filtraggio	12
3.5	Use case: UC5-Applica filtraggio	14
4.1	Logo di Visual Studio Code	18
4.2	Logo di Slack	18
4.3	Logo di Figma	18
4.4	Logo di Github	19
4.5	Logo di Typescript	19
4.6	Logo di Python	20
4.7	Logo di React	20
4.8	Logo di CSS	21
4.9	Logo di Node JS	21
4.10	Logo di Ant Design	22
4.11	Logo di OpenAI	22
4.12	Tabella da filtrare	24
4.13	Esempio filtraggio manuale della colonna preesistente con selezione dei valori predefiniti	24
4.14	Esempio filtraggio manuale della colonna preesistente con selezione di una range di date	25
4.15	Esempio filtraggio manuale della colonna preesistente con una ricerca	25
4.16	Diagramma delle classi del frontend	26
4.17	ciclo delle operazioni per il filtraggio della tabella	28
4.18	Modale del filtraggio testuale	32
4.19	Modale del filtraggio testuale con componenti espansi	32
4.20	Riepilogo del filtraggio	37

Elenco delle tabelle

2.1	Obiettivi fissati	4
2.2	Distribuzione delle ore	5
3.1	Tabella del tracciamento dei requisiti funzionali	15
3.2	Tabella del tracciamento dei requisiti qualitativi	16
3.3	Tabella del tracciamento dei requisiti di vincolo	16
4.1	Differenze tra JavaScript e TypeScript [9][35][19]	31
6.1	Distribuzione delle ore svolte	49
6.2	Obiettivi raggiunti	50
6.3	Tabella del soddisfacimento dei requisiti qualitativi	50
6.4	Tabella del soddisfacimento dei requisiti di vincolo	51
6.5	Tabella del soddisfacimento dei requisiti funzionali	51

Elenco dei Codici

4.1	Organizzazione dei file nelle cartelle	26
4.2	Componente a classi	30
4.3	Componente a funzioni	30
4.4	Variabili di stato	32
4.5	Renderizzazione del componente Help	33
4.6	Gestione del riconoscimento vocale	34
4.7	Definizione di un API utilizzando Redux RTK	34
4.8	Utilizzo del hook useLazyGetAdvancedSearchQuery	35
4.9	Formato JSON delle risposte	35
4.10	Definizioni dei variabili necessari per il filtraggio	36
4.11	Metodi per il filtraggio della tabella	37

4.12 Richiesta alle API di OpenAI	39
4.13 JSON schema per il Function Call con più funzioni	40
4.14 JSON schema per il Function Call	41
4.15 Richiesta alle API di OpenAI usando il Function Calling	42

Capitolo 1

Introduzione

In questo capitolo viene rappresentata una breve introduzione all'azienda ospitante, al progetto e all'organizzazione del documento.

1.1 L'azienda

RiskApp (logo riportato nella [Figura 1.1](#)) è una startup [Insuretech](#)^[8] che si basa sulla trasformazione digitale e l'analisi avanzata del rischio. Aiuta le compagnie di piccole e medie imprese nelle esigenze insurance, dalla previsione di possibili catastrofi naturali alle attività di media monitoring online.

La piattaforma RiskApp offre inoltre una serie di servizi che aiutano gli intermediari assicurativi che lavorano con le imprese; per esempio il "Cacciatore di dati" che raccoglie tutte le informazioni che presenti online del cliente, dalle informazioni finanziarie, al rating, dalle notizie alle esposizioni naturali; i programmi "NatCat" e "Supply-chain" tramite i quali gli intermediari possono determinare il livello di esposizione dei rischi a causa del cambiamento climatico a cui sono sottoposti gli elementi chiave delle attività dei clienti, come i fornitori che potrebbero tardare la consegna delle merci [33].



Figura 1.1: Logo RiskApp

1.2 Introduzione al progetto

Nelle molteplici applicazioni che richiedono la visualizzazione di dati tabellari, spesso ci si imbatte in tabelle estese, composte da numerose colonne. Gli utenti, a loro volta, spesso necessitano di filtrare queste tabelle in base a criteri specifici. Senza il supporto dell'intelligenza artificiale, i programmatori devono prevedere opzioni di filtraggio per ciascuna colonna, come bottoni per l'ordinamento crescente e decrescente, la selezione di intervalli o la ricerca di dati specifici all'interno di ciascuna colonna. Questo approccio ha il difetto di richiedere un notevole sforzo da parte dell'utente,

specialmente quando vuole applicare filtri su più colonne, navigando tra le opzioni colonna per colonna.

La sfida consiste quindi nel semplificare queste operazioni, trasformandole in richieste testuali e sfruttando l'intelligenza artificiale. Questo consente agli utenti di esprimere le loro esigenze in linguaggio naturale e inviare tali richieste all'[Artificial Intelligence \(AI\)](#)^[8], che si incaricherà di eseguire i filtri richiesti sulla tabella in modo efficiente e senza la necessità di navigare tra diverse opzioni.

1.3 Organizzazione del testo

Il secondo capitolo approfondisce gli obiettivi dello stage, la pianificazione delle attività e l'analisi dei rischi.

Il terzo capitolo approfondisce l'analisi dei requisiti.

Il quarto capitolo approfondisce le tecnologie utilizzate, lo sviluppo del codice e tecniche di utilizzo delle [Application Program Interface \(API\)](#)^[8] di OpenAI [23].

Il quinto capitolo approfondisce la verifica del codice, l'addestramento dell'[AI](#).

Nel sesto capitolo descrive le conclusioni valutando il percorso stage e l'esperienza ottenuta.

Capitolo 2

Descrizione dello stage

In questo capitolo viene rappresentata la pianificazione e l'implementazione dello stage presso RiskApp, attraverso un'introduzione al progetto e illustrando gli obiettivi affrontati. Verranno esaminati la pianificazione delle attività e analisi, e come i potenziali rischi siano stati considerati e gestiti.

2.1 Prodotto da ottenere

L'obiettivo del progetto è integrare un nuovo servizio in una piattaforma web esistente. Nello specifico si cerca di creare un bottone sopra una tabella esistente il quale fa accedere ad una [modale](#)^[g], che consenta agli utenti di inserire e modificare testi interagendo con il servizio dell'AI chiamato ChatGPT. Inoltre, questa [modale](#) permette la visualizzazione e la modifica dei risultati ottenuti dalla comunicazione con ChatGPT [3], offrendo agli utenti la flessibilità di apportare modifiche ai risultati in base alle loro esigenze.

Nel caso in cui gli utenti non siano soddisfatti dei risultati ottenuti, devono avere la possibilità di tornare indietro e effettuare una nuova ricerca. Una volta confermati i risultati, essi possono essere utilizzati per applicare il filtraggio dei dati sulla tabella. Nel [backend](#)^[g] del sistema, è necessario sviluppare un metodo per costruire le [query](#)^[g] basate sui dati forniti dall'utente, al fine di filtrare correttamente i dati nella tabella.

Un aspetto da sottolineare è che la funzione di filtraggio dei dati è già implementata all'interno della tabella esistente. Gli utenti attuali hanno la possibilità di personalizzare la visualizzazione della tabella in base alle loro preferenze, navigando tra le diverse colonne e utilizzando le opzioni di filtraggio esistenti. Quello che il nuovo servizio introduce è una semplificazione di queste operazioni. Invece di dover utilizzare opzioni di filtraggio predefinite, gli utenti avranno la possibilità di esprimere i loro requisiti di filtraggio utilizzando il linguaggio naturale. Ciò consentirà agli utenti di risparmiare tempo, eliminando la necessità di cercare manualmente le opzioni di filtraggio appropriate per ciascuna colonna.

2.2 Pianificazione delle attività

Obiettivi fissati

Nella [Tabella 2.1](#) vengono riportati gli obiettivi fissati del progetto. Durante lo stage è stato integrato un ulteriore obiettivo, marcato come O05.

Si farà riferimento agli obiettivi secondo le seguenti notazioni:

- **O**: obiettivi obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
- **D**: obiettivi desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- **F**: obiettivi facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Codice	Descrizione
O01	Presenza di uno spazio di inserimento testuale per interagire con il servizio di intelligenza artificiale.
O02	Creazione del prompt da usare nei servizi di OpenAI [23] per la creazione delle query dal testo.
O03	Possibilità di visualizzare e modificare i dati individuati che verranno utilizzati per il filtraggio.
O04	Utilizzo dei risultati di OpenAI per filtrare la tabella.
O05	Presenza degli esempi sull'utilizzo del servizio implementato.
D01	Possibilità di correzione del filtraggio precedente con una nuova ricerca testuale.
F01	Possibilità dell'inserimento vocale.

Tabella 2.1: Obiettivi fissati

Distribuzione delle ore

Nella [Tabella 2.2](#) vengono riportate le ore distribuite per le varie attività durante lo stage.

2.3 Analisi preventiva dei rischi

Durante la fase di analisi iniziale, sono stati individuati alcuni possibili rischi a cui si potrà andare incontro. Si è quindi proceduto a elaborare delle possibili soluzioni per far fronte a tali rischi.

1. Uso di nuove tecnologie

Descrizione: l'integrazione del servizio di intelligenza artificiale rappresenta una sfida, in quanto si tratta di una tecnologia relativamente nuova e mancano esempi e guide esaustive sull'utilizzo di questo servizio.

Soluzione: implementare una soluzione preliminare, ispirandosi a esempi analoghi, che si avvicini all'obiettivo desiderato. Successivamente, si intende perfezionare gradualmente questa soluzione.

2. Richieste non pertinenti al filtraggio

Descrizione: gli utenti potrebbero formulare richieste che non sono pertinenti al processo di filtraggio dei dati all'interno della tabella.

Soluzione: scrivere una guida sull'uso del prodotto.

3. Accuratezza dei risultati

Descrizione: i risultati ottenuti dal servizio potrebbero risultare non accurati e non rispecchiare le richieste dell'utente.

Soluzione: si prevede l'implementazione di un controllo a livello del [backend](#) al fine di validare e verificare i risultati prima che vengano presentati all'utente.

Durata in ore	Descrizione dell'attività
92	Formazione sulle tecnologie
40	Formazione Typescript
20	Formazione React
16	Formazione servizi OpenAI
16	Formazione API Riskapp
40	Analisi e pianificazione
120	Sviluppo
8	Implementazione di uno spazio di inserimento testuale
60	Implementazione del prompt nei servizi di OpenAI per creazione delle query
24	Implementazione di uno spazio di visualizzazione e modifica dei dati elaborati per filtraggio dei dati
28	Utilizzo dei dati per filtraggio della tabella
48	Collaudo Finale
24	Stesura dei test
16	collaudo e verifica
8	Stesura della documentazione
Totale ore	300

Tabella 2.2: Distribuzione delle ore

Capitolo 3

Analisi dei requisiti

Nel presente capitolo, verranno affrontati dettagliatamente gli aspetti legati all'analisi dei requisiti del progetto.

3.1 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto, sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo [Unified Modeling Language \(UML\)](#)^[8] dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso. Essendo il progetto finalizzato alla creazione di un tool per l'automazione di un processo, le interazioni da parte dell'utilizzatore devono essere ovviamente ridotte allo stretto necessario. Per questo motivo, i diagrammi d'uso risultano semplici e in numero ridotto.

Nella [Figura 3.1](#) viene mostrato lo scenario generale.

UC1: Autenticazione

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non ha eseguito il login.

Descrizione: L'utente esegue il login.

Postcondizioni: L'utente ha eseguito il login con successo.

UC2: Visualizzazione bottone del modale di filtraggio

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha effettuato il login e fa parte degli utenti abilitati a usare la [modale](#) di filtraggio.

Descrizione: L'utente nella piattaforma web vede il bottone per accedere al servizio del filtraggio avanzato.

Postcondizioni: L'utente ha visto il bottone.

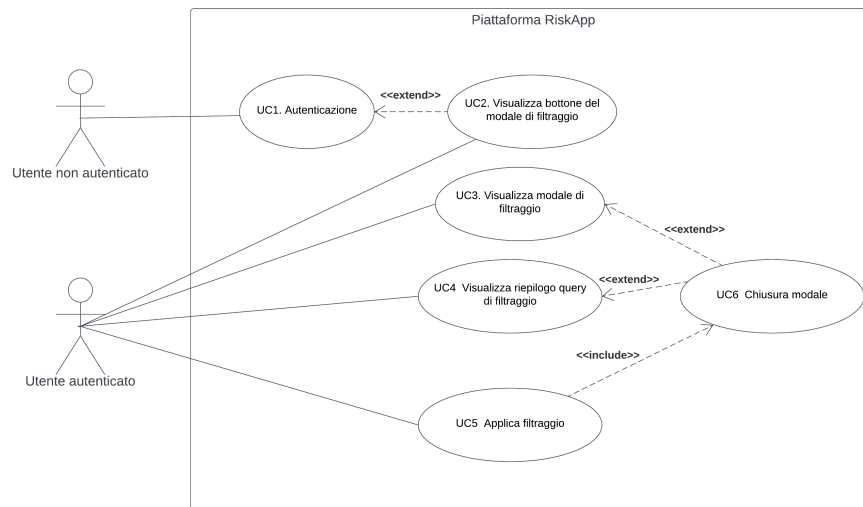


Figura 3.1: Use case: scenario principale

UC3: Visualizzazione modale di filtraggio

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha effettuato il login e fa parte degli utenti abilitati a usare la **modale** di filtraggio.

Descrizione: L'utente nella piattaforma web clicca il bottone e visualizza la **modale** per il filtraggio avanzato.

Postcondizioni: L'utente visualizza la **modale**.

Nella [Figura 3.2](#) viene mostrato il caso d'uso descritto.

UC3.1: Inserimento testo manualmente

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha effettuato ha aperto la **modale** di filtraggio.

Descrizione: L'utente inserisce il testo nell'area di testo della **modale** per eseguire il filtraggio avanzato.

Postcondizioni: L'utente ha inserito il testo.

UC3.2: Visualizza suggerimento

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la **modale** di filtraggio.

Descrizione: L'utente visualizza i suggerimenti su come scrivere il testo per il filtraggio avanzato.

Postcondizioni: L'utente ha visualizzato i suggerimenti.

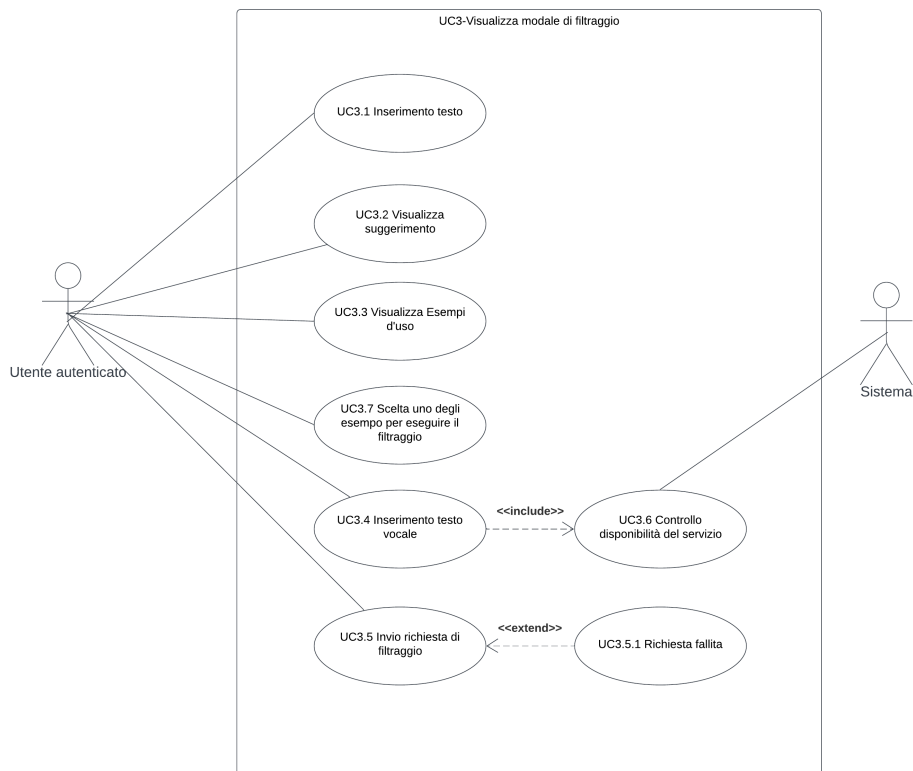


Figura 3.2: Use case: UC3-Visualizza modale di filtraggio

UC3.3: Visualizza esempi d'uso

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la [modale](#) di filtraggio.

Descrizione: L'utente visualizza gli esempi su come usare il filtraggio avanzato.

Postcondizioni: L'utente ha visualizzato gli esempi.

UC3.4: Inserimento vocale

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la [modale](#) di filtraggio.

Descrizione: L'utente inserisce il testo nell'area di testo utilizzando l'inserimento vocale.

Postcondizioni: L'utente ha inserito il testo.

UC3.5: Invio richiesta di filtraggio

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha inserito il testo per la richiesta del filtraggio.

Descrizione: L'utente invia la richiesta di filtraggio.

Postcondizioni: L'utente ha inviato la richiesta.

UC3.5.1: Richiesta fallita

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha inviato la richiesta del filtraggio.

Descrizione: Il sistema non riceve risposta da parte delle [API](#) di ChatGPT.

Postcondizioni: Viene mostrato un messaggio di errore all'utente .

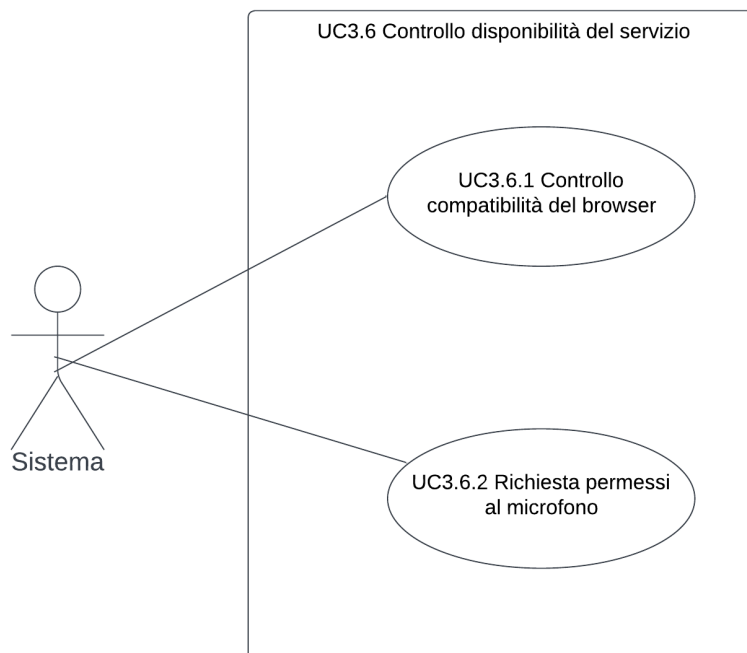


Figura 3.3: Use case: UC3.6-Controllo disponibilità del servizio

UC3.6: Controllo disponibilità del servizio

Attori Principali: Sistema.

Precondizioni: L'utente vuole eseguire l'inserimento vocale.

Descrizione: La piattaforma RiskApp [33] controlla i requisiti per l'uso del microfono.

Postcondizioni: Viene attivato il microfono.

Nella [Figura 3.3](#) viene mostrato il caso d'uso descritto.

UC3.6.1: Controllo compatibilità del browser

Attori Principali: Sistema.

Precondizioni: Viene avviato il controllo per attivazione del microfono.

Descrizione: La piattaforma RiskApp controlla la compatibilità del browser per l'uso del microfono.

Postcondizioni: Viene controllato il browser.

UC3.6.2: Richiesta permessi al microfono

Attori Principali: Sistema.

Precondizioni: Viene avviato il controllo per attivazione del microfono.

Descrizione: La piattaforma RiskApp controlla e richiede i permessi per accesso al microfono se necessario.

Postcondizioni: Viene dato i permessi per accesso al microfono.

UC3.7: Scelta uno degli esempi per eseguire il filtraggio

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha visualizzato gli esempi su come usare gli esempi di filtraggio.

Descrizione: L'utente sceglie tra le opzioni presenti un esempio che viene inserito nell'area di testo automaticamente una volta scelta.

Postcondizioni: L'esempio d'uso scelto è inserito nell'area di testo.

UC4: Visualizza riepilogo query di filtraggio

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha inviato e ricevuto la risposta da ChatGPT.

Descrizione: Viene dato all'utente la possibilità di visualizzare una lista dei filtri che verranno applicati alla tabella come riepilogo.

Postcondizioni: L'utente ha visualizzato i filtri.

Nella [Figura 3.4](#) viene mostrato il caso d'uso descritto.

UC4.1: Visualizza la lista delle query di filtraggio valide

Attori Principali: Utente autenticato.

Precondizioni: L'utente sta visualizzando il riepilogo dei filtri.

Descrizione: L'utente vede una lista dei filtri che verranno applicati alla tabella e questi filtri sono validi, cioè possono effettivamente essere applicati.

Postcondizioni: L'utente ha visualizzato i filtri.

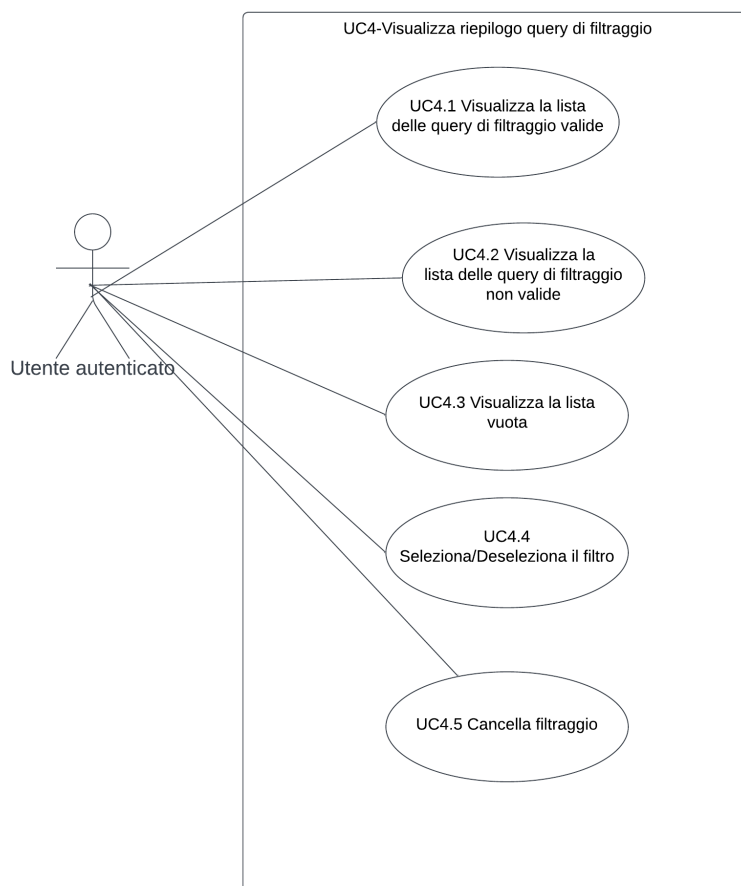


Figura 3.4: Use case: UC4-Visualizza riepilogo query di filtraggio

UC4.2: Visualizza la lista delle query di filtraggio non valide

Attori Principali: Utente autenticato.

Precondizioni: L'utente sta visualizzando il riepilogo dei filtri.

Descrizione: L'utente vede una lista dei filtri che verranno applicati alla tabella e questi filtri non sono validi, cioè non possono e non verranno applicati.

Postcondizioni: L'utente ha visualizzato i filtri.

UC4.3: Visualizza la lista vuota

Attori Principali: Utente autenticato.

Precondizioni: L'utente sta visualizzando il riepilogo dei filtri.

Descrizione: L'utente vede una lista dei filtri vuota, cioè ChatGPT ha determinato

che la richiesta mandata dall'utente non si riferisce al filtraggio della tabella.

Postcondizioni: L'utente ha visualizzato la lista dei filtri vuota.

UC4.4: Seleziona/Deseleziona i filtri

Attori Principali: Utente autenticato.

Precondizioni: L'utente sta visualizzando il riepilogo dei filtri validi.

Descrizione: L'utente può decidere se applicare o meno un determinato filtro selezionando o deselegionando quel filtro.

Postcondizioni: L'utente ha selezionato i filtri da applicare.

UC4.5: Cancella filtraggio

Attori Principali: Utente autenticato.

Precondizioni: L'utente sta visualizzando il riepilogo dei filtri.

Descrizione: L'utente non è soddisfatto dei filtri presenti nella lista e decide di ripetere la richiesta di filtraggio cambiando il testo.

Postcondizioni: L'utente torna nella fase di invio di richiesta di filtraggio.

UC5: Applica filtraggio

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha visualizzato il riepilogo dei filtri.

Descrizione: L'utente ha visto la lista dei filtri validi e può applicare effettivamente i filtri.

Postcondizioni: Vengono applicati i filtri.

Nella [Figura 3.5](#) viene mostrato il caso d'uso descritto.

UC5.1: Applica filtri validi

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha visualizzato il riepilogo dei filtri.

Descrizione: L'utente ha visto la lista dei filtri, la lista contiene filtri validi e l'utente ha selezionato almeno un filtro da applicare.

Postcondizioni: Vengono applicati i filtri selezionati.

UC5.2: Applica filtri non validi o vuoti

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha visualizzato il riepilogo dei filtri.

Descrizione: L'utente ha visto la lista dei filtri, la lista contiene solo filtri non validi o la lista è vuota e l'utente comunque procede all'applicazione del filtro.

Postcondizioni: Non viene applicato nessun filtro e la tabella rimane invariata.

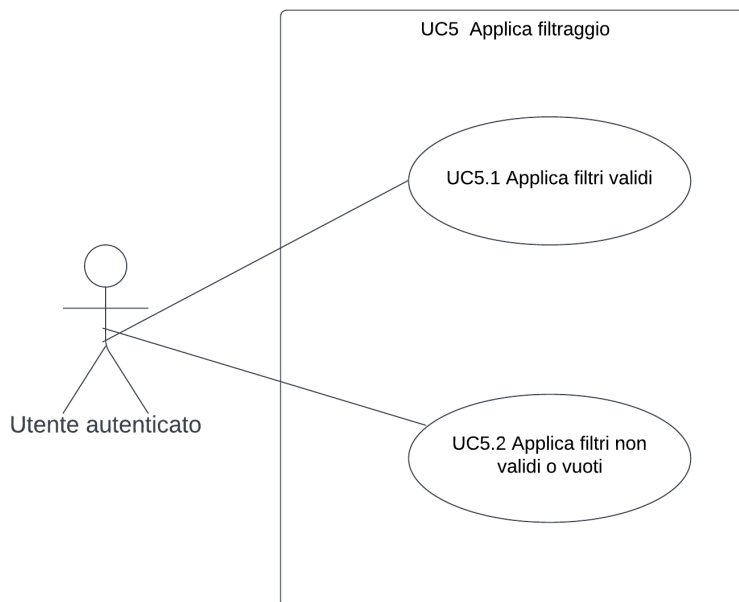


Figura 3.5: Use case: UC5-Applica filtraggio

UC6: Chiusura modale

Attori Principali: Utente autenticato.

Precondizioni: La **modale** del filtraggio è aperto.

Descrizione: L'utente può chiudere la **modale** del filtraggio in qualunque momento.

Postcondizioni: Viene chiuso la **modale** del filtraggio.

3.2 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto, è stata stilata la tabella che traccia i requisiti in rapporto agli use case.

Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli.

Il codice dei requisiti è così strutturato R(F/Q/V)(N/D/O) dove:

R = requisito

F = funzionale

Q = qualitativo

V = di vincolo

N = obbligatorio (necessario)

D = desiderabile

Z = opzionale

Nelle Tabelle 3.1, 3.2 e 3.3 sono riassunti i requisiti e il loro tracciamento con i fonti o gli use case delineati in fase di analisi.

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione	Use Case
RFN-1	Solo utenti autorizzati devono poter accedere al servizio di filtraggio testuale	UC2
RFN-2	Il servizio di filtraggio testuale deve essere accessibile tramite un bottone dedicato.	UC2
RFN-3	Tutte le funzionalità del servizio devono essere contenute all'interno di una finestra modale .	UC3
RFN-4	L'interfaccia deve includere un'area di testo appositamente dedicata all'inserimento della richiesta di filtraggio	UC3.1
RFN-5	L'area di testo per l'inserimento deve avere una lunghezza massima specificata e notificare l'utente in caso di superamento.	UC3.1
RFN-6	Il pulsante di invio deve essere attivo solo se è stato inserito del testo nell'area dedicata.	UC3.5
RFN-7	L'utente deve poter resettare i valori inseriti nell'area di testo mediante un apposito bottone.	UC3.1
RFN-8	All'interno della finestra modale , deve essere fornita una guida sull'uso del servizio e suggerimenti sulla struttura del testo per il filtraggio.	UC3.2
RFN-9	Dovrebbero essere disponibili degli esempi di testo predefiniti che l'utente può scegliere per provare il filtraggio.	UC3.3
RFN-10	L'utente deve poter visualizzare tutti i filtri individuati e avere il controllo per selezionare quali filtri desidera applicare.	UC4.4
RFD-11	L'utente deve poter ripetere la richiesta di filtraggio, apportando modifiche al testo iniziale.	UC4.5
RFZ-12	Possibilità di utilizzare l'inserimento vocale come alternativa per l'input testuale nel servizio.	UC3.4

Tabella 3.2: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Fonte
RQD-1	Il codice deve essere ben commentato al fine di garantire la comprensione, la manutenibilità e la collaborazione all'interno del team di sviluppo. I commenti devono essere chiari, concisi e fornire spiegazioni adeguate sulle parti complesse del codice.	Proponente
RQD-2	Il riconoscimento vocale deve essere implementato in modo che sia disponibile nella maggior parte dei browser moderni.	Proponente

Tabella 3.3: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Fonte
RVN-1	La parte frontend ^[8] deve essere sviluppata utilizzando il linguaggio di programmazione Typescript [35]	proponente
RVN-2	La parte backend deve essere sviluppata utilizzando il linguaggio di programmazione Python [27]	proponente
RVN-3	Deve essere utilizzata la libreria Ant Design [1] per una coerenza nel design e nell'aspetto dell'applicazione.	Proponente

Capitolo 4

Progettazione e codifica

Nel presente capitolo, si descrive la fase di progettazione e codifica del progetto. Questa sezione del documento si concentra sullo sviluppo del progetto e descrive l'architettura del frontend. In particolare, vengono analizzate le scelte architetturali cruciali che guidano lo sviluppo del sistema. Uno degli aspetti centrali è l'integrazione delle API di ChatGPT e il modo in cui sono state adoperate per migliorare l'interazione con l'utente.

Facciamo notare che, a causa delle ragioni legate alla sicurezza e alla riservatezza dei dati aziendali, non è stato consentito condividere immagini o screenshot relativi al codice prodotto dall'azienda. Pertanto, non saranno inclusi elementi grafici che rappresentano il codice proprietario.

4.1 Tecnologie e strumenti

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati.

4.1.1 Strumenti utilizzati

VS Code

Nella [Figura 4.1](#) viene mostrato il logo di [Visual Studio Code \(VS Code\)](#), un editor di codice sorgente sviluppato da Microsoft che si è rapidamente affermato come uno degli strumenti più popolari nella comunità di sviluppatori [36]. Una delle sue caratteristiche distintive è l'elevata personalizzazione: gli utenti possono estendere le funzionalità dell'editor tramite una vasta gamma di estensioni, coprendo argomenti che vanno dall'integrazione di linguaggi specifici alla gestione di *repository Git*.

La preferenza per questo strumento deriva dalla familiarità e dalla comodità data dall'esperienza in passato.

Slack

Nella [Figura 4.2](#) viene mostrato il logo di Slack, una piattaforma di comunicazione primaria utilizzata nel corso del progetto. Si tratta di uno strumento per la messaggistica istantanea, la condivisione di file e l'organizzazione delle comunicazioni di gruppo [34]. L'adozione di Slack è stata una scelta richiesta del proponente, in quanto è uno strumento ampiamente adottato dall'azienda.



Figura 4.1: Logo di Visual Studio Code



Figura 4.2: Logo di Slack

Figma

Nella [Figura 4.3](#) viene mostrato il logo di Figma, una piattaforma collaborativa basata su cloud che consente la progettazione di interfacce utente, la creazione di [mockup](#)^[8] e prototipi interattivi [11]. Figma è stato adottato su richiesta del proponente come strumento principale per la progettazione e la creazione dei prototipi grafici.



Figura 4.3: Logo di Figma

Github

Nella [Figura 4.4](#) viene mostrato il logo di GitHub, una piattaforma per il controllo delle versioni e la gestione del lavoro di sviluppo [16]. L'adozione di GitHub è stata una scelta supportata sia dalle esperienze passate personali che dalla richiesta espressa dal proponente.



Figura 4.4: Logo di Github

4.1.2 Tecnologie utilizzare

Typescript

Nella [Figura 4.5](#) viene mostrato il logo di TypeScript, un linguaggio di programmazione [open source](#) sviluppato da Microsoft. Si tratta di un'estensione di JavaScript che basa le sue caratteristiche su [ECMAScript](#)^[6] [10]; capo del progetto è Anders Hejlsberg. Il linguaggio estende la sintassi di JavaScript in modo che qualunque programma scritto in JavaScript sia anche in grado di funzionare con TypeScript senza nessuna modifica [35].

TypeScript inoltre rappresenta un linguaggio che amplia e arricchisce le caratteristiche attuali di JavaScript, introducendo una serie di miglioramenti tra cui:

- Firma dei metodi;
- Utilizzo di classi;
- Definizione di interfacce;
- Organizzazione in moduli;
- Impiego dell'operatore "`=>`" per la creazione di funzioni anonime;
- Introduzione di tipi di dato (anche opzionali);
- Utilizzo di enumerazioni;
- Applicazione di mixin tra classi.

L'adozione di Typescript è stata una richiesta del proponente per garantire una coerenza nell'ambito tecnologico del progetto, in quanto la parte della piattaforma web da integrare con ChatGPT è sviluppata utilizzando TypeScript.



Figura 4.5: Logo di Typescript

Python

Nella [Figura 4.6](#) viene mostrato il logo di Python, un linguaggio di programmazione ad alto livello, interpretato e orientato agli oggetti, creato e rilasciato per la prima volta nel 1991. Python è noto per la sua sintassi chiara e leggibile, che lo rende particolarmente adatto per sviluppatori di ogni livello di esperienza [27].

L'adozione di Python è stata motivata dalla necessità di soddisfare la sicurezza e la protezione delle *API key* (è una chiave segreta che concede l'accesso ai servizi di OpenAI) [25].



Figura 4.6: Logo di Python

React

Nella [Figura 4.7](#) viene mostrato il logo di React, un framework JavaScript [open source](#) sviluppato da Facebook. È ampiamente utilizzato per la creazione di interfacce utente dinamiche e reattive. La caratteristica distintiva di React è la sua architettura basata su componenti, che consente agli sviluppatori di costruire interfacce complesse suddividendo l'applicazione in piccoli pezzi riutilizzabili [29].

Il [frontend](#) della piattaforma RiskApp è sviluppata con React.

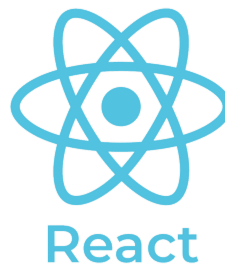


Figura 4.7: Logo di React

CSS3

Nella [Figura 4.8](#) viene mostrato il logo di CSS3, linguaggio dichiarativo per la formattazione di pagine Web [8].



Figura 4.8: Logo di CSS

Node JS

Nella [Figura 4.9](#) viene mostrato il logo di Node JS, un framework [open source](#) per l'esecuzione di codice JavaScript [22].



Figura 4.9: Logo di Node JS

Ant Design

Nella [Figura 4.10](#) viene mostrato il logo di Ant Design, una libreria di componenti [User Interface \(UI\) open source](#). Essa offre un vasto insieme di componenti pre-stilizzati che semplificano la creazione di interfacce utente. Ant Design è ampiamente utilizzata nella comunità di sviluppatori React per la sua facilità d'uso, la completezza dei componenti e la flessibilità nell'adattamento ai diversi progetti [1].

L'adozione di Ant Design è stata una richiesta del proponente per garantire una coerenza nell'ambito estetico del progetto, in quanto la piattaforma RiskApp è stata sviluppata utilizzando componenti di Ant Design.



Figura 4.10: Logo di Ant Design

React Speech Recognition

React Speech Recognition è un [hook](#)^[g] di React che fornisce accesso all'[API](#) di riconoscimento vocale del web (*Web Speech API*). Questa [API](#) consente di convertire il parlato catturato dal microfono del dispositivo in testo utilizzabile all'interno dei componenti React dell'applicazione [31].

OpenAI API

Nella [Figura 4.11](#) viene mostrato il logo di OpenAI, un'organizzazione di ricerca sull'intelligenza artificiale con lo scopo di promuovere e sviluppare un'intelligenza artificiale amichevole [23].

Una delle realizzazioni più rilevanti di OpenAI è ChatGPT [3], un modello di linguaggio generativo in grado di creare testo coerente e convincente. ChatGPT è stato accolto con grande interesse per la sua capacità di rispondere a input testuali in modo naturale e dettagliato, simulando conversazioni significative con gli utenti. Il modello ha dimostrato competenze in molteplici ambiti, dalla risposta a domande all'elaborazione del linguaggio naturale [25].

Il progetto mira proprio a integrare ChatGPT nella piattaforma web di RiskApp sfruttando le [API](#) fornite da OpenAI. Questa integrazione consentirà agli utenti di interagire con il modello al fine di ottimizzare il processo di filtraggio dati attraverso richieste testuali.



Figura 4.11: Logo di OpenAI

4.2 Architettura preesistente

RiskApp è una piattaforma basata su web. Questa piattaforma opera su un modello modulare, strutturato attraverso una serie di servizi distribuiti su vari repository. Gli utenti possono accedere solo a un sottoinsieme specifico dei servizi disponibili in base alla loro sottoscrizione.

Nel contesto di questo progetto, è stato dato l'accesso ad una parte specifica del [frontend](#) relativa alla la gestione e visualizzazione delle trattative attraverso una tabella. Questa parte del codice è stata sviluppata utilizzando il linguaggio TypeScript [35], il framework React [29] e la libreria Redux [30].

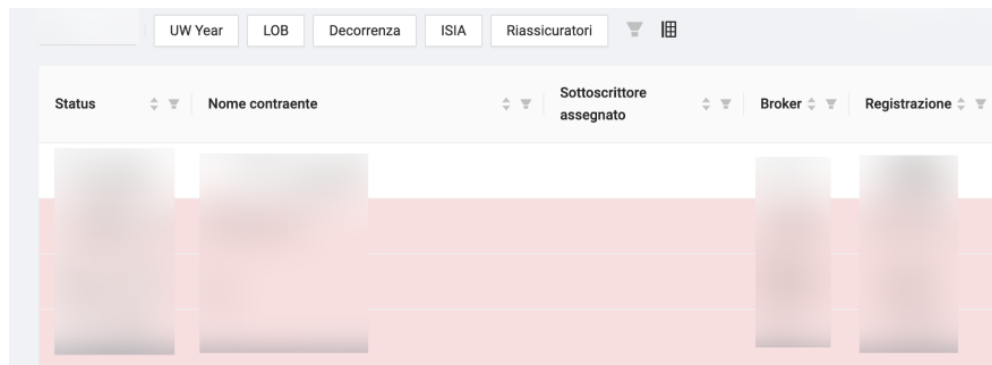
4.2.1 Struttura e tipi di dati della tabella

Come rappresentato nella [Figura 4.12](#), è possibile osservare la tabella che sarà sottoposta al processo di filtraggio. Va notato che l'immagine mostra solo una porzione limitata delle colonne disponibili, poiché il numero totale di colonne è considerevole (circa 40). Per ogni colonna, gli utenti hanno la possibilità di applicare il filtraggio attraverso la ricerca o selezionando tra le opzioni predefinite, come illustrate nella [Figura 4.13](#), [Figura 4.14](#) e [Figura 4.15](#). Inoltre, è consentita la scelta di visualizzare i dati della tabella in ordine crescente o decrescente.

È importante notare che è possibile accedere a tutte le colonne disponibili e selezionare quali di esse mostrare nella tabella cliccando l'apposito bottone posizionato nell'angolo in alto a destra dell'interfaccia. Le diverse colonne della tabella rappresentano differenti tipologie di dati. Dall'ottica del sistema, ciascuna colonna può contenere uno dei seguenti tipi di dati:

- Booleani: *true* o *false*;
- Stringhe: testi o parole in stringa;
- Stringhe a scelta multipla: una set di valori in stringa selezionabili;
- Range di numeri: due valori numerici;
- Range di date: due valori di data;
- Numeri a scelta multipla: una set di numeri selezionabili;
- ID.

La comprensione dei tipi di dati associati a ciascuna colonna è estremamente importante in quanto contribuisce all'addestramento del modello ChatGPT, il quale deve essere in grado di riconoscere e manipolare tali dati. Tale informazione sarà successivamente impiegata nello sviluppo di funzioni atte a adattare i dati prima dell'esecuzione delle [query](#) di filtraggio.



The screenshot shows a data table interface. At the top, there are several filter buttons: 'UW Year', 'LOB', 'Decorrenza', 'ISIA', and 'Riassicuratori'. Below these, the table header includes columns for 'Status', 'Nome contraente', 'Sottoscrittore assegnato', 'Broker', and 'Registrazione'. The table body contains several rows of data, which are mostly blurred out.

Figura 4.12: Tabella da filtrare

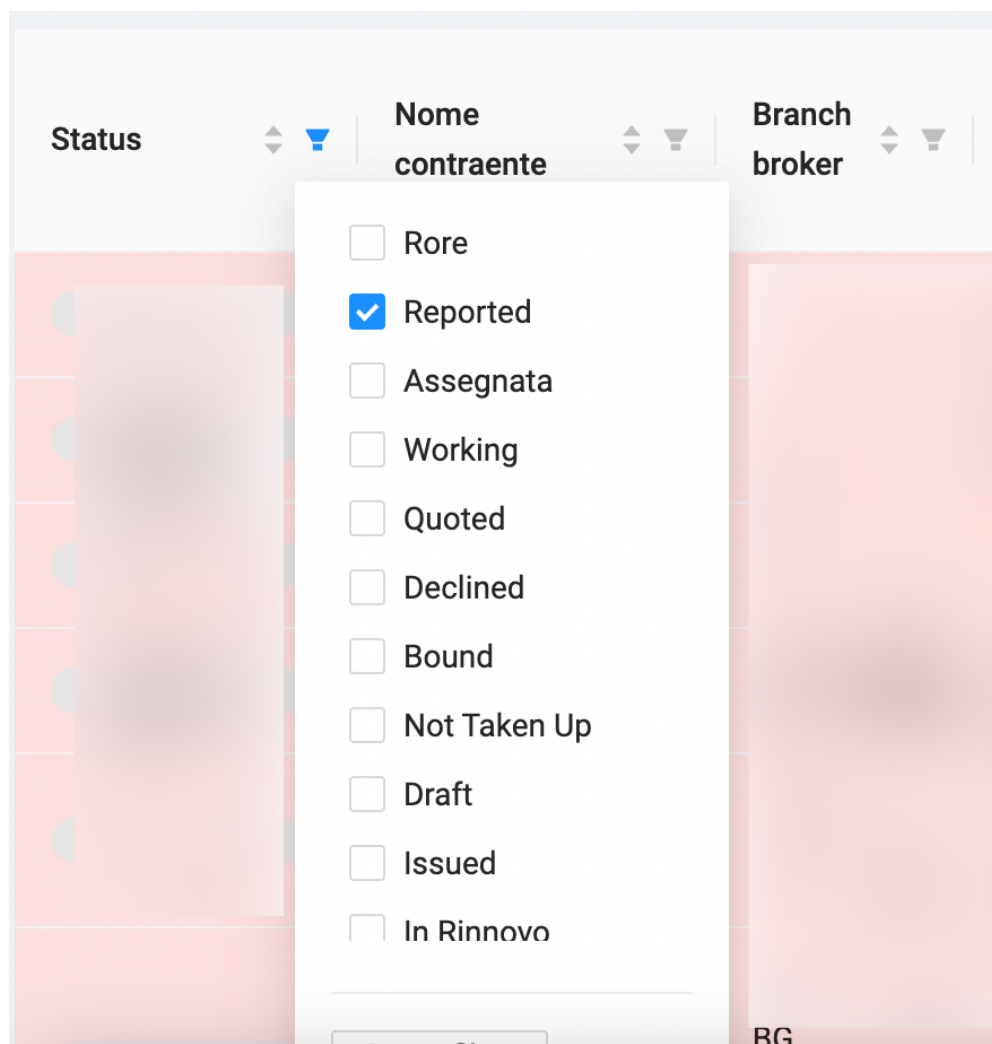


Figura 4.13: Esempio filtraggio manuale della colonna preesistente con selezione dei valori predefiniti

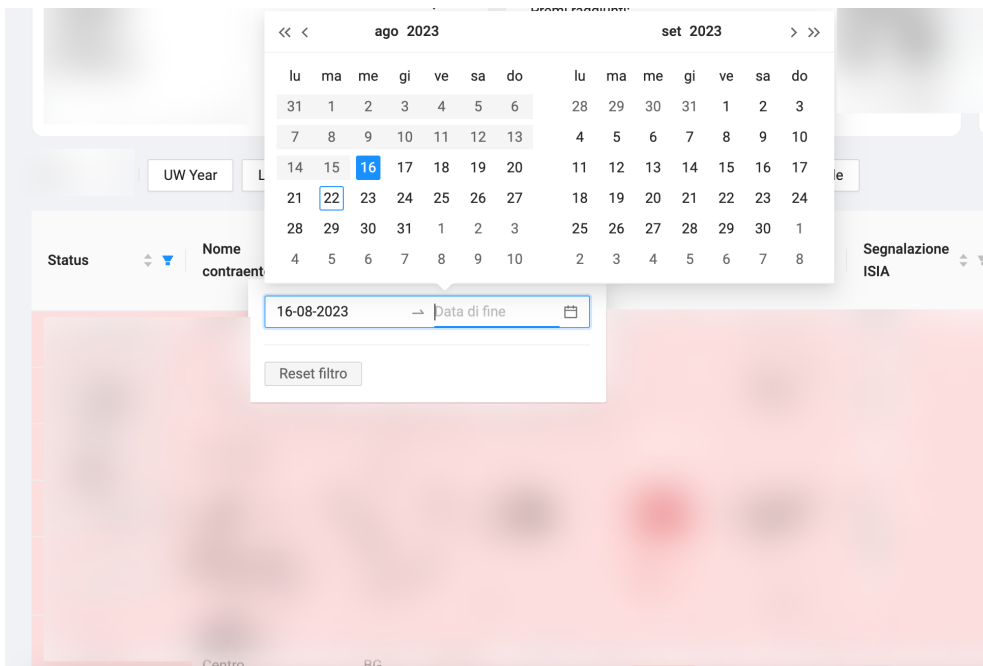


Figura 4.14: Esempio filtraggio manuale della colonna preesistente con selezione di una range di date

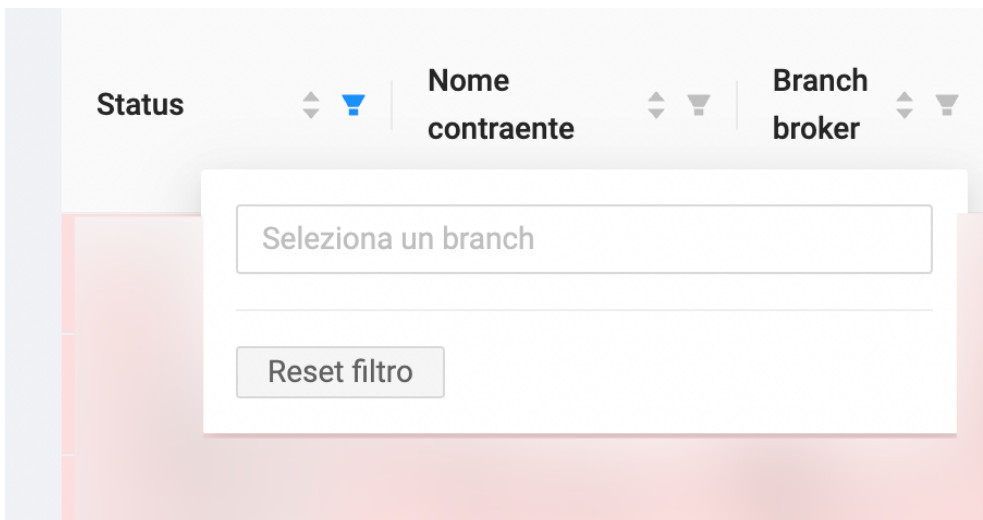


Figura 4.15: Esempio filtraggio manuale della colonna preesistente con una ricerca

4.2.2 Integrazione del nuovo componente

Durante lo sviluppo, non sono state apportate numerose modifiche ai file preesistenti. Questo è dovuto al fatto che tutte le nuove funzionalità implementate relative al filtraggio della tabella utilizzando un linguaggio naturale, d'ora in poi chiamata semplicemente "filtraggio testuale", sono state raggruppate all'interno di una [modale](#).

È importante sottolineare che l'accesso alla [modale](#) avviene esclusivamente attraverso un bottone dedicato. Questo bottone è stato integrato nell'interfaccia della piattaforma RiskApp, offrendo agli utenti un punto di ingresso chiaro e intuitivo per accedere alle nuove funzionalità. Dal punto di vista dell'implementazione, è stata solamente aggiunto un collegamento al componente per il filtraggio testuale con la tabella, come mostrato nella [Figura 4.16](#).

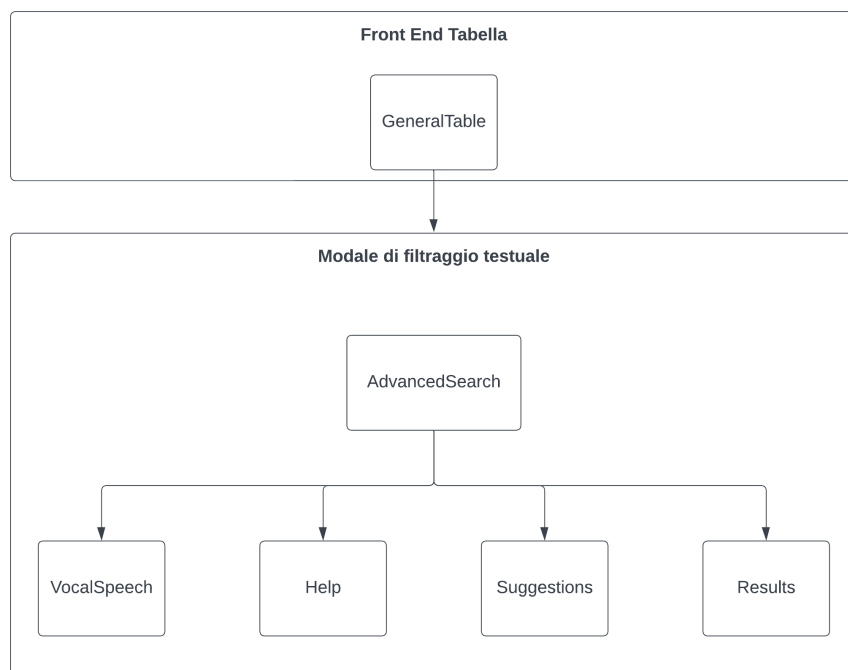


Figura 4.16: Diagramma delle classi del frontend

4.3 Architettura del nuovo componente

Inizialmente, l'approccio progettuale era orientato a sviluppare l'intero progetto esclusivamente come parte del [frontend](#). In questo contesto, tutte le operazioni erano concepite per essere integrate nella sezione [frontend](#) della piattaforma RiskApp. Tuttavia, con un'analisi più approfondita delle [API](#) di OpenAI, è emerso che tali chiamate dovrebbero essere effettuate attraverso il server [backend](#) [14]. Di conseguenza, l'architettura del progetto è stata riveduta e suddivisa in due componenti principali: una parte [frontend](#) dedicata all'implementazione dell'interfaccia utente e una parte [backend](#) responsabile della gestione delle chiamate [API](#) e dell'addestramento dell'intelligenza artificiale.

Nel [Codice 4.1](#), vengono mostrati come sono organizzati i file creati durante lo sviluppo dello [UI](#) all'interno delle cartelle della parte [frontend](#) della piattaforma RiskApp.

```
- src
```

```

|- components
  |-...
  |- tables
    |-...
    |- negotiationsTables
      |- generalTables
        |- advancedSearch
          |-AdvancedSearchButton.tsx
          |-Help.tsx
          |-Hint.tsx
          |-Result.tsx
        |-...
      |- redux
    |- api
      |- advancedSearchApiRequest.ts
    |-...
  |- types
    |- advancedSearchData.ts
    |- specialFields.ts
  |-...

```

Codice 4.1: Organizzazione dei file nelle cartelle

Per la parte [backend](#), non è stato dato il permesso d'accesso alle cartelle, quindi non si è potuto mostrare l'organizzazione delle cartelle. Tuttavia, è possibile fornire una panoramica dei file creati. Sono stati sviluppati due file fondamentali: "functionCall.py" e "gptRequest.py". Entrambi i file sono scritti in linguaggio Python e sono responsabili della gestione delle chiamate alle [API](#) di OpenAI.

4.3.1 Frontend

L'architettura del [frontend](#) del progetto segue un modello di sviluppo basato su componenti, focalizzato sull'uso di React e Ant Design. Il modello ha le seguenti caratteristiche [29]:

- **Componenti React:** sono delle unità autonome di codice riutilizzabili e ciascuna rappresenta un'interfaccia utente o una funzionalità specifica dell'applicazione;
- **Struttura gerarchica:** i componenti vengono organizzati in una struttura a gerarchia, simile a un albero, quindi i componenti possono essere annidati all'interno di altri componenti;
- **Gestione dello Stato:** lo stato è un oggetto che può contenere dati che cambiano nel tempo. Quando lo stato di un componente cambia, il componente si aggiorna e viene nuovamente renderizzato.

Il file principale dell'applicazione, spesso denominato "App.jsx" in caso di JavaScript e "App.tsx" se TypeScript, funge da componente principale. All'interno di questo componente, vengono integrati vari elementi dell'interfaccia utente, come la [modale](#), l'area di testo e i bottoni, tutti forniti dalla libreria Ant Design. Nella [Figura 4.16](#) si può vedere come i vari componenti della [modale](#) di filtraggio testuale sono gestiti e connessi tra loro, in particolare:

- **AdvancedSearch:** questo componente rappresenta il fulcro dell'intero sistema, gestendo sia la renderizzazione della [modale](#) stessa che la coordinazione di tutti

gli altri componenti. La componente *AdvancedSearch* costituisce il punto di connessione per tutti gli altri moduli e determina la visualizzazione o l'occultamento dei vari componenti in base alle interazioni dell'utente;

- **VocalSpeech**: questo componente gestisce la possibilità dell'inserimento vocale del testo di filtraggio, controlla i permessi del microfono e ne gestisce l'attivazione e la disattivazione;
- **Help**: questo componente gestisce la visualizzazione degli esempi di testi da utilizzare per il filtraggio. L'utente può selezionare uno di questi esempi e inserirlo nell'area di testo;
- **Suggestions**: questo componente gestisce la visualizzazione dei suggerimenti e delle istruzioni su come scrivere il testo di filtraggio per ottenere risultati più precisi;
- **Results**: questo componente gestisce la visualizzazione del riepilogo dei filtri che verranno applicati alla tabella. I filtri sono individuati dal servizio di intelligenza artificiale e l'utente può selezionare quelli da applicare alla tabella.

L'architettura si basa sull'utilizzo di uno stato interno a ciascun componente, che è responsabile di controllare la visualizzazione e il comportamento dinamico. Ad esempio, l'uso di uno stato chiamato "showResult" determina se la componente "results" sarà visibile all'interno della *modale*. Questo approccio di gestione dello stato locale è una caratteristica chiave di React e consente di creare interazioni complesse tra i vari componenti. Nella [Figura 4.17](#) vengono mostrate le operazioni che l'utente esegue per un filtraggio testuale.

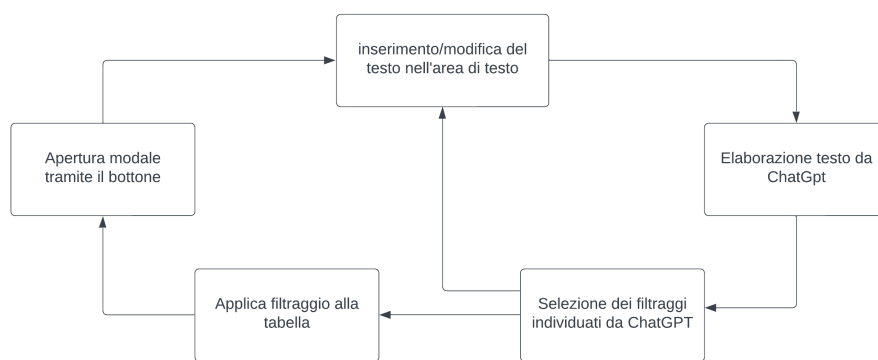


Figura 4.17: ciclo delle operazioni per il filtraggio della tabella

4.3.2 Backend

Come già menzionato, sono stati implementati due file dedicati per soddisfare la necessità di effettuare chiamate alle *API* di OpenAI. Questa scelta è motivata dalla sicurezza delle chiavi *API*, che rappresentano un elemento sensibile nella fase di accesso ai servizi di OpenAI [25]: per garantirne la protezione, è consigliato che le richieste

siano inviate attraverso il server [backend](#). Ciò assicura che la chiave [API](#) sia gestita in un ambiente sicuro, come variabile d'ambiente o servizio di gestione delle chiavi.

La parte [backend](#) del progetto gestisce le operazioni di comunicazione con ChatGPT, tra cui:

- Determinazione delle impostazioni di conversazione, come la selezione del modello e del metodo di conversazione da utilizzare;
- Implementazione dello [JSON schema](#)^[g] per [function calling](#)^[g];
- Addestramento di ChatGPT tramite prompt specifici per migliorare la pertinenza delle risposte;
- Implementazione di funzioni per il controllo e la correzione delle risposte generate da ChatGPT.

Un aspetto da sottolineare è che sono state implementate solamente la logica per consentire la comunicazione con le [API](#) di OpenAI e le funzioni per il controllo dei dati necessari al funzionamento del sistema. La trasmissione dei dati tra il [backend](#) e il [frontend](#) del progetto è stata sviluppata da altri membri dell'azienda. Questa suddivisione è stata adottata poiché, come accennato nella [Sezione 4.3](#), l'originario focus del progetto era principalmente sullo sviluppo [frontend](#).

Function Calling

La tecnica usata per l'addestramento di ChatGPT si chiama [function calling](#). Questa tecnica è introdotta nella *Chat Completions API* nel 1/07/2023 e permette ai modelli *gpt-4-0613* e *gpt-3.5-turbo-0613* di comprendere e rispondere alle richieste degli sviluppatori attraverso una forma più strutturata e orientata alla chiamata di funzioni [15].

Nello specifico, gli sviluppatori possono descrivere le funzioni alle quali desiderano far riferimento nei loro input testuali. Quando il modello riceve una richiesta che contiene una descrizione di una funzione, esso è in grado di interpretare tale richiesta e generare un oggetto [JavaScript Object Notation \(JSON\)](#) contenente gli argomenti necessari per chiamare effettivamente quella funzione.

4.3.3 Libreria Redux Toolkit

[Redux Toolkit \(RTK\)](#) [30] è una libreria che semplifica la gestione dello stato globale in un'applicazione JavaScript, rendendo più agevole la gestione dei dati con Redux. Redux è uno strumento molto potente per gestire lo stato dell'applicazione in modo coerente e scalabile, ma può richiedere una configurazione complessa, [RTK](#) semplifica questa configurazione e introduce convenzioni per uno sviluppo più veloce ed efficiente.

Il funzionamento si basa su concetti chiave come "store", "action" e "reducer":

- **Store:** è il contenitore centrale dello stato dell'applicazione, contiene tutti i dati che devono essere condivisi tra i diversi componenti. Questo stato è immutabile, il che significa che non può essere modificato direttamente, ma solo attraverso "action";
- **Action:** rappresenta un cambiamento nello stato dell'applicazione. È un oggetto JavaScript che contiene un tipo e, facoltativamente, dei dati associati all'azione;

- **Reducer**: è una funzione che riceve lo stato corrente e un'azione, e calcola il nuovo stato in base all'azione. I reducer sono puri e immutabili, il che significa che non modificano lo stato esistente, ma restituiscono uno stato nuovo.

4.4 Codifica

Di seguito viene descritta la fase di codifica, riportando anche le funzioni più rilevanti.

4.4.1 Componenti React

Nello sviluppo con React, l'interfaccia utente di un'applicazione è costituita da componenti. Questi componenti rappresentano parti autonome e riutilizzabili di codice. Per creare un componente, si possono seguire due approcci diversi [13]:

- Approccio basato su classi
- Approccio basato su funzioni

Nel [Codice 4.2](#) viene mostrato un componente definito tramite classe in JavaScript che estende la classe `React.Component`. Questa classe contiene un metodo obbligatorio chiamato `render()`, che restituisce l'output visuale del componente.

Nel [Codice 4.3](#) viene mostrato un componente definito tramite funzione in JavaScript. Questa funzione restituisce l'output visuale del componente.

Entrambi gli approcci hanno lo stesso obiettivo: creare componenti riutilizzabili per costruire l'interfaccia utente. Tuttavia, l'approccio basato su classi è meno utilizzato negli ultimi tempi, poiché l'approccio basato su funzioni che sfrutta gli [hook](#), come lo "useState" e "useEffect", consentono alle funzioni di gestire lo stato e l'effetto, rendendo le funzioni di React altamente potenti e flessibili [32].

```
1 class ComponenteClasse extends React.Component {
2   render() {
3     return <h1>Componente basato su classi</h1>;
4   }
5 }
```

Codice 4.2: Componente a classi

```
1 function ComponenteFunzione() {
2   return <h1>Componente basato su funzioni</h1>;
3 }
```

Codice 4.3: Componente a funzioni

Differenza tra JavaScript e TypeScript

JavaScript e TypeScript sono entrambi linguaggi di programmazione che vengono spesso utilizzati per lo sviluppo di applicazioni web e altre applicazioni basate su browser, ma presentano alcune differenze chiave come mostrate nella [Tabella 4.1](#).

	JavaScript	TypeScript
Tipizzazione	È un linguaggio a tipizzazione dinamica, il che significa che non è necessario dichiarare il tipo di una variabile prima di assegnarle un valore. I tipi sono determinati a runtime.	È un linguaggio a tipizzazione statica, il che significa che bisogna dichiarare il tipo di una variabile prima di assegnarle un valore. Questo permette di catturare errori di tipo durante la fase di sviluppo.
Dato	Ha tipi di dati primitivi come stringhe, numeri e booleani, ma non supporta una definizione di tipo strutturata per oggetti complessi.	Offre tipi di dati primitivi e consente di definire tipi personalizzati, interfacce e union types per oggetti più complessi.
Type Annotations	Non ha annotazioni di tipo formali. I tipi vengono implicitamente assegnati in base ai valori assegnati alle variabili.	Richiede l'uso di annotazioni di tipo per dichiarare il tipo di una variabile, parametro di funzione o valore di ritorno di una funzione.
Compatibilità	È ampiamente supportato da tutti i browser e può essere eseguito senza la necessità di trasformazioni aggiuntive.	Deve essere compilato in JavaScript prima di essere eseguito nei browser. Il codice TypeScript deve essere trasformato in JavaScript valido.
Orientamento agli oggetti	Supporta l'orientamento agli oggetti ma in modo meno strutturato rispetto a TypeScript.	Offre una gestione più chiara dell'orientamento agli oggetti grazie all'introduzione di classi e interfacce.

Tabella 4.1: Differenze tra JavaScript e TypeScript [9][35][19]

Su richiesta del proponente, il **frontend** è stato sviluppato utilizzando il linguaggio Typescript, sia per la necessità di mantenere una coerenza con il resto del codice, sia per sfruttare i vantaggi offerti dal controllo dei tipi e dalla personalizzazione dei dati. Invece, per l'approccio di sviluppo dei componenti React, è stata data libertà di scelta e di conseguenza è stato adottato l'approccio basato su funzioni, poiché offre una sintassi più concisa e leggibile.

4.4.2 Modale per il filtraggio testuale

Nella **Figura 4.18**, viene mostrata la **modale** creata per il filtraggio testuale. Seguendo il percorso dall'alto verso il basso, si possono vedere i componenti in sequenza: "Suggerimenti", "Esempi", un'area di testo e infine un pulsante per attivare il riconoscimento vocale. L'assemblaggio di questi componenti è stato realizzato utilizzando le strutture predefinite di Ant Design [1].

Figura 4.18: Modale del filtraggio testuale

Figura 4.19: Modale del filtraggio testuale con componenti espansi

È da notare che sia i componenti "Suggerimenti" che "Esempi" possono essere espansi (Figura 4.19) o ridotti (Figura 4.18) se cliccati. Per gestire la visualizzazione o l'occultamento di questi componenti, ci si affida alle variabili di stato messe a disposizione da React [29], mostrate nel Codice 4.4.

```

1 const [isModalOpen, setIsModalOpen] = React.useState(false);
2 const [textAreaValue, setTextAreaValue] = React.useState("Estrai le
  trattative ");
3 const [results, setResults] = React.useState(false);
4 const [showHelp, setShowHelp] = React.useState(false);
5 const [showHint, setShowHint] = React.useState(false);
6 const [vocalButtonDisabled, setVocalButtonDisabled] = React.useState(
  false);

```

```
7 const [isVoiceRecognitionActive, setIsVoiceRecognitionActive] = React.
   useState(false);
```

Codice 4.4: Variabili di stato

Un esempio di ciò è rappresentato dalla riga 4 del codice: `"const [showHelp, setShowHelp] = React.useState(false);"`. In questa linea, si sta creando una variabile di stato chiamata "showHelp" e un metodo associato chiamato "setShowHelp". Si inizializza "showHelp" a "false", il che significa che inizialmente il componente "Suggerimenti" è nascosto. Quando si vuole mostrare il componente, basta semplicemente chiamare "setShowHelp(true)" e durante la fase di renderizzazione del componente basta controllare lo stato di "showHelp" e applicare lo stile al componente come mostrato nel Codice 4.5 alla riga 4.

```
1 export default function Help({ show, handleInsertHelp }: propHelp) {
2   return (
3     <>
4     <div style={show? { display: 'block', marginBottom: '1em' } :
5     { display: 'none'}}>
6       <Card size="small">
7         {data.map((element, index)=>{
8           return <Button
9             key={index}
10            onClick={()=>handleInsertHelp(
11              element)}
12            style={{whiteSpace: 'normal', height:
13              'auto', display: 'flex', alignItems: 'center', marginBottom: '4px'}}
14            >{element}<SendOutlined/></Button>
15          )}}
16       </Card>
17     </div>
18   </>
19 );
20 }
```

Codice 4.5: Renderizzazione del componente Help

4.4.3 Riconoscimento vocale

Per lo sviluppo del riconoscimento vocale, ovvero la conversione del parlato in testo scritto, è stato necessario integrare la libreria React Speech Recognition [31], un progetto [open source](#) che fa affidamento sulla *Web Speech API* [37] sottostante del browser per gestire il riconoscimento vocale. React Speech Recognition semplifica notevolmente l'implementazione, offrendo una serie di componenti e [hook](#) predefiniti che consentono di catturare l'input vocale e trasformarlo in testo utilizzabile all'interno dell'applicazione. A livello di codice la libreria offre:

- **SpeechRecognition**: una classe che fornisce metodi per controllare il riconoscimento vocale, tra cui metodi per iniziare, interrompere o fermare l'ascolto vocale;
- **useSpeechRecognition**: un [hook](#) personalizzato che fornisce uno stato e delle funzioni per gestire il riconoscimento vocale in un componente funzionale. Utilizzando questo [hook](#), si ottiene un oggetto con varie proprietà e funzioni per interagire con il riconoscimento vocale.

Nel [Codice 4.6](#) viene riportata la gestione del riconoscimento vocale usando la classe e il [hook](#).

```

1 //Voice recognition service
2 const { transcript, listening: isListening, resetTranscript,
  browserSupportsSpeechRecognition, isMicrophoneAvailable } =
  useSpeechRecognition();
3 //check if the browser has microphone permission and toggle on/off
  voice recognition
4 function handleVoiceRecognition(){
5   if(browserSupportsSpeechRecognition){ //check if the browser
  supports speech recognition
6     if(isMicrophoneAvailable){ //check if the browser has
  microphone permission
7       if(isVoiceRecognitionActive){
8         stopListening();
9       } else {
10        startListening();
11      }
12    }else{
13      message.warning("Il microfono non e' disponibile o i permessi
  sono stati negati.")
14    }
15    } else {
16      message.warning("Il tuo browser non supporta tale servizio.")
17    }
18  }
19  //start voice recognition
20  async function startListening(){
21    await SpeechRecognition.startListening({ continuous: true,
  language: "it-IT"});
22    setIsVoiceRecognitionActive(true);
23  }
24  //stop voice recognition and save the transcript in the textarea
25  async function stopListening(){
26    await SpeechRecognition.stopListening();
27    setIsVoiceRecognitionActive(false);
28    setTextAreaValue(prevState => `${prevState} ${transcript}`)
29    resetTranscript();
30  }

```

Codice 4.6: Gestione del riconoscimento vocale

4.4.4 Invio richiesta di filtraggio e ricezione risultati

L'invio del testo e ricezione della risposta viene gestita attraverso alla libreria [RTK](#) [30] di React. Nel [Codice 4.7](#) viene mostrato il funzionamento.

```

1 export const advancedSearch = apiSlice.injectEndpoints({
2   endpoints: (builder) => ({
3     getAdvancedSearch: builder.query<advancedSearchData[], {
  instruction?: string}>({
4       query: (param) => {
5         return {
6           url: '/negotiation/advancedsearch',
7           method: 'POST',
8           body: { "text": param.instruction },
9         };
10      },
11    }),
12  }},

```

```

13 }})
14 export const { useLazyGetAdvancedSearchQuery } = advancedSearch

```

Codice 4.7: Definizione di un API utilizzando Redux RTK

Gli elementi fondamentali sono:

- **apiSlice.injectEndpoints(...)**: definisce gli [endpoint](#)^[g] dell'API. Gli [endpoint](#) sono configurati all'interno della funzione *(builder) => (...)*;
- **getAdvancedSearch: builder.query<advancedSearchData[], instruction?: string>(...)**: definisce l'[endpoint](#) *getAdvancedSearch* come una [query](#). Accetta un parametro opzionale *instruction* di tipo stringa, questo parametro corrisponde al testo inserito dall'utente nell'area di testo nella [modale](#). Quando viene effettuata una chiamata a questo [endpoint](#), verrà eseguita una richiesta POST all'URL *'/negotiation/advancedsearch'* con il corpo contenente il testo fornito tramite *param.instruction*;
- **export const useLazyGetAdvancedSearchQuery = advancedSearch:** esporta un [hook](#) *useLazyGetAdvancedSearchQuery* che può essere utilizzato nei componenti React per iniziare la chiamata API, inoltre fornisce anche accesso alle variabili di stato associate alla chiamata.

Nel [Codice 4.8](#) viene mostrato come avviene l'invio della richiesta (riga 8) e ricevimento della risposta (riga 9).

```

1 //get the response from AI services
2 const [getData, {isLoading, isFetching, error: errorGpt}] =
  useLazyGetAdvancedSearchQuery()
3 //managing the response of AI services
4 const [objects, setObjects] = React.useState<advancedSearchData[]>([]);
5 [...]
6 //send request to chat gpt, wait for response and show results
7 function sendRequestAndShowResults() {
8   getData({instruction: textAreaValue}).then((response) => {
9     const data=response.data;
10    setObjects(data || []); //save response from AI services
11    searchUIDS(data || []); //search for uids if required
12    setShowHelp(false); //hide Help component
13    setVocalButtonDisabled(true); //disable the voice
  recognition button
14    setResults(true); //show results component
15  });
16 };

```

Codice 4.8: Utilizzo del hook useLazyGetAdvancedSearchQuery

Formato JSON della risposta

Per agevolare la gestione e il controllo dei dati ricevuti in risposta, il [backend](#) formatta le risposte in un array di oggetti [JSON](#). Ciascun oggetto, che segue il formato illustrato nel [Codice 4.9](#), rappresenta un filtro specifico applicato a una colonna.

```

1 export interface advancedSearchData{
2   systemLabel: string,
3   userLabel: string,
4   systemValue: number[] | boolean[] | string[] | { label: string, value:
  string }[],

```

```

5   userValue: string[],
6   applyChanges: boolean,
7   disabled: boolean
8 }

```

Codice 4.9: Formato JSON delle risposte

L'adozione di questa formattazione è motivata dal fatto che il nome delle colonne e il formato dei valori di filtraggio variano a seconda che vengano visualizzati dall'utente o dal sistema. Ad esempio, considerando la colonna "Status" nella [Figura 4.12](#), i suoi valori sono selezionabili come opzioni predefinite (come mostrato in [Figura 4.13](#)). Tuttavia, dal punto di vista del sistema, essa è identificata come "state" e i valori sono rappresentati come numeri. Inoltre, la necessità di presentare all'utente un riepilogo dei filtri trovati e da applicare ([Sottosezione 4.4.5](#)) ha portato alla creazione di questa struttura.

Quindi, al fine di agevolare l'estrazione dei dati rilevanti durante le operazioni di filtraggio tabellare, è stato deciso di conservare sia il nome della colonna (*systemLabel*, *userLabel*) sia il valore di filtraggio (*systemValue*, *userValue*) sia dal punto di vista dell'utente che da quello del sistema. In aggiunta, sono presenti due valori booleani, *applyChanges* e *disabled*, che indicano rispettivamente se il filtro deve essere effettivamente applicato (decisione presa dall'utente durante la visualizzazione del riepilogo dei filtri) e se il filtro è applicabile (se non lo è, *applyChanges* è impostato su "false" e l'utente non avrà modo di selezionare quel filtro).

4.4.5 Visualizzazione riepilogo del filtraggio

Dopo che ChatGPT ha restituito i risultati, il componente "Results" (descritto nella [Sottosezione 4.3.1](#)) utilizza il formato JSON per mostrare all'utente un riepilogo visuale delle colonne e dei relativi filtri che verranno applicati a ciascuna colonna. Un esempio di ciò è evidenziato nella [Figura 4.20](#), in cui, a partire dalla richiesta espressa in linguaggio naturale nell'area di testo, sono stati individuati tre campi da filtrare: "Importo assicurato", "LPS" e "Decorrenza", insieme ai loro valori corrispondenti. In questa fase, l'utente ha la possibilità di deselegionare eventuali filtri che non desidera applicare.

Una volta cliccato il pulsante "Applica filtraggio" posizionato in basso a destra, vengono estrapolati dal JSON i dati necessari per il successivo processo di filtraggio, come mostrato nel [Codice 4.10](#). La variabile "filters" raccoglie i valori dei filtri selezionati, "sorters" gestisce l'ordinamento ascendente o discendente delle colonne, e "selected columns" contiene l'elenco delle colonne da visualizzare nella tabella, tenendo conto delle selezioni dell'utente e dei filtri applicati.

```

1 //filters
2 const filters = objects
3   .filter(el => el.applyChanges &&
4     el.systemValue.length!==0 &&
5     el.systemLabel!=="ascending_ordering" &&
6     el.systemLabel!=="descending_ordering" &&
7     el.systemLabel!="reset_filters")
8   .map((item) => ({key: item.systemLabel, range: item.systemValue}));
9
10 //ascend or descend ordering of columns
11 const sorters: {key: string, type: "ascend" | "descend"}[] = objects
12   .filter(el => (el.systemLabel==="ascending_ordering" || el.
13     systemLabel==="descending_ordering") && el.applyChanges)
14   .flatMap((item) => {

```



```

14     return item.systemValue.map((column) => ({
15         key: column.toString(),
16         type: item.userLabel==="Ordinazione crescente"? "ascend" : "
17         descend"
18     }));
19
20 //show columns
21 const selected_columns_set = [...(userSelectedColumns ?? []), ...objects
22     .flatMap((item) => {
23         if(item.systemLabel!=="order") return item.systemLabel;
24         else return item.systemValue.map(column => column.toString())
25     }]);
26 const selected_columns= selected_columns_set.filter((value, index, self)
27     => {
28     return self.indexOf(value) === index;
29 });

```

Codice 4.10: Definizioni dei variabili necessari per il filtraggio

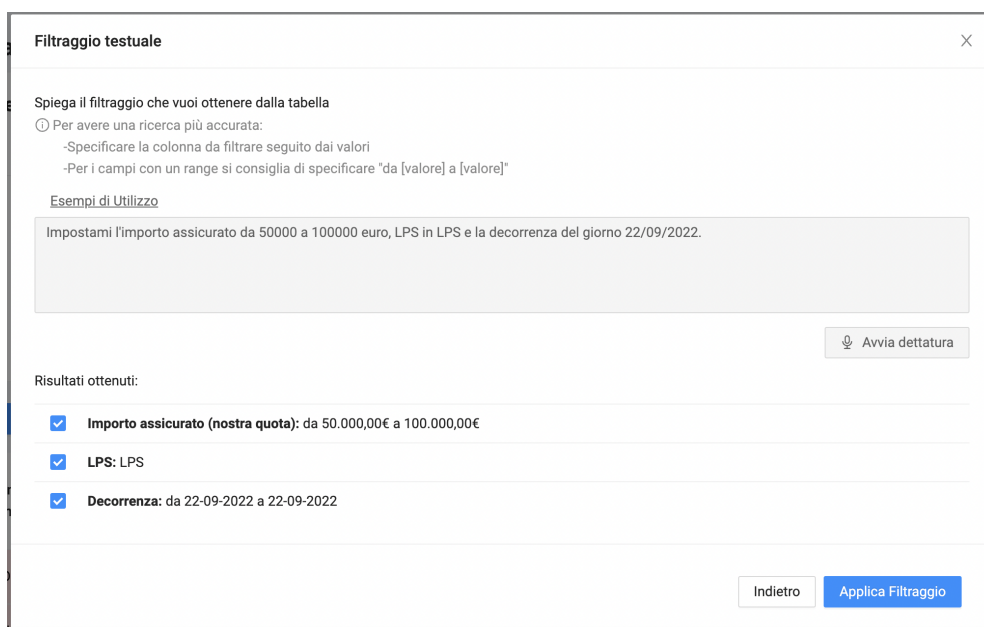


Figura 4.20: Riepilogo del filtraggio

Infine, completando il processo di filtraggio, le tre variabili vengono passate alle funzioni di filtraggio già preesistenti e definite, come illustrato nel Codice 4.11. Questo passo chiude l'operazione di filtraggio, risultando nella visualizzazione della tabella con i filtri applicati come richiesto.

```

1 const dispatch = useDispatch();
2 dispatch(setTablesPreferences({
3     [Views[Views.general]]: {
4         sorters: sorters? sorters : [],
5         filters: filters,
6         selected_columns: selected_columns,
7         page: DEFAULT_CURRENT_PAGE_TABLE,
8         page_size: DEFAULT_CURRENT_PAGE_SIZE_TABLE

```

```

9      }
10   })
11   updatePreferences({
12     [Views[Views.general]]: {
13       sorters: sorters? sorters : [],
14       filters: filters,
15       selected_columns: selected_columns,
16     }
17   })

```

Codice 4.11: Metodi per il filtraggio della tabella

4.4.6 API di OpenAI

Per la richiesta alle [API](#) di OpenAI, il sito ufficiale di OpenAI fornisce librerie sia per Python che per Node.js. Una volta installata la libreria e inserita correttamente la *API key*, è possibile procedere con le chiamate alle [API](#) [25].

Il processo di chiamata alle [API](#) è relativamente semplice, come illustrato nel [Codice 4.12](#). In questo frammento di codice in Python, prima viene impostata la chiave attraverso l'attributo "api key" dell'oggetto "openai". Successivamente, la funzione "run conversation" definisce una conversazione tra un utente e un assistente. La conversazione è rappresentata da una lista di oggetti "messages", ciascuno con un ruolo ("role") e un contenuto ("content"); il ruolo può essere "user" per l'utente e "system" per l'assistente. La risposta ritornata sarà anche a sua volta un oggetto contenente diversi dati, tra cui il campo "message" che contiene la risposta vera e propria.

La chiamata avviene attraverso il metodo "openai.ChatCompletion.create", il quale richiede i seguenti parametri fondamentali:

- **Modello:** specifica il modello di ChatGPT che verrà utilizzato per generare le risposte. Nel progetto, su richiesta del proponente, è stato utilizzato il modello [gpt-3.5-turbo](#)^[8]. Si tratta di un modello che migliora *GPT-3* ed è in grado di comprendere e generare linguaggio naturale o codice [5];
- **Messaggi:** definisce la conversazione tra l'utente e l'assistente. La conversazione è rappresentata come una lista di oggetti "messages", ognuno contenente un ruolo ("user" o "system") un contenuto ("content") del messaggio. È consigliabile iniziare la conversazione con un messaggio di sistema, il quale svolge un ruolo introduttivo o di contesto in modo di addestrare l'assistente prima di affacciarlo nella conversazione con l'utente. Ad esempio, fornendo un contesto all'assistente su cosa si aspettarsi dalla conversazione o definendo il ruolo specifico dell'assistente nell'interazione. Successivamente, l'utente e l'assistente ("assistant", quindi ChatGPT) possono scambiarsi messaggi alternati, come dimostrato nell'esempio di codice fornito. Questa struttura di conversazione aiuta a guidare l'assistente nelle interazioni desiderate e a ottenere risposte coerenti con il contesto.

Inoltre, sono disponibili alcuni parametri opzionali che consentono di personalizzare ulteriormente la generazione delle risposte:

- **Temperature:** controlla la creatività delle risposte. Valori più alti, come ad esempio 1.0, rendono le risposte più creative e imprevedibili, mentre valori più bassi, come 0.2, rendono le risposte più deterministiche e coerenti [4]. Nel progetto, il valore è impostato a 0 in quanto le risposte che si vogliono ottenere devono essere più deterministiche possibile per facilitare l'elaborazione dei dati;

- **Function_call**: permette di chiamare direttamente funzioni personalizzate durante la conversazione, il valore predefinito è "auto" così da lasciare liberamente ChatGPT a richiamare le funzioni; può essere impostato su "none" se non si vuole che vengano richiamate le funzioni oppure si può indicare esplicitamente la funzione da richiamare. Nel progetto per questo campo è lasciato il valore predefinito;
- **Functions**: viene specificata se si sta usando la [function calling](#), sono funzioni personalizzate da utilizzare durante la generazione delle risposte. Le funzioni possono essere definite separatamente e richiamate nel testo per influenzare il comportamento dell'assistente.

```
import openai
openai.api_key = [myKey]

def run_conversation():
    messages = [
        {
            "role": "system",
            "content": "Sei un assistente che aiuta l'utente a
prendere decisioni su che pasto prendere.",
        },
        {"role": "user", "content": "Consigliami cosa mangiare
oggi a pranzo."},
    ]
    completion: dict = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=messages,
        temperature=0.0,
    )
    response_message = completion["choices"][0]["message"]

#Risposta di ChatGPT (response_message)
{
    "content": "Certo, posso darti alcune idee per il pranzo!
Insalata di Quinoa, Wrap di Pollo, Pasta Integrale con Pesto
[...] ",
    "role": "assistant"
}
```

Codice 4.12: Richiesta alle API di OpenAI

Chiamata API utilizzando Function Call

La chiamata con la funzione chiamata *Function call* consente all'assistente di comprendere, all'interno di una conversazione, quando è necessario eseguire chiamate a funzioni specifiche e quale funzione invocare. Questa capacità è resa possibile attraverso il parametro "functions" nel [JSON schema](#), il quale definisce una serie di funzioni che possono essere richiamate durante la conversazione. Ogni funzione è descritta tramite uno [JSON schema](#) e all'interno ha un nome, una descrizione e i parametri dati attesi dalla funzione [25].

Nell'esempio fornito nel [Codice 4.13](#), vengono definiti secondo la documentazione di OpenAI, due esempi di funzioni: "set_isia" e "set_lps", che sono progettate per

impostare i filtri rispettivamente per le colonne ISIA e LPS. Entrambe le funzioni accettano un parametro "value", che è una stringa con valori specifici. Questo array rappresenta le opzioni che l'utente può selezionare per filtrare i dati.

Durante l'interazione con l'assistente, se l'utente richiede un'azione che corrisponde a una chiamata di funzione, come ad esempio "Impostami la colonna LPS in No LPS", l'assistente riconoscerà la situazione grazie alla descrizione delle funzioni nel [JSON schema](#). L'oggetto di risposta generato dall'assistente conterrà un campo "function_call" che indica il nome della funzione da richiamare e i relativi argomenti [24].

```

functions = [
  {
    "name": "set_isia",
    "description": "imposta la colonna ISIA",
    "parameters": {
      "type": "object",
      "properties": {
        "value": {
          "type": "string",
          "enum":["ISIA", "No ISIA"],
        }
      },
      "required": ["value"],
    },
  },
  {
    "name": "set_lps",
    "description": "imposta la colonna LPS",
    "parameters": {
      "type": "object",
      "properties": {
        "value": {
          "type": "string",
          "enum":["LPS", "No LPS"],
        }
      },
      "required": ["value"],
    },
  },
]
[...]
completion: dict = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=messages,
    functions=functions,
    temperature=0.0,
    function_call="auto",
)
response_message = completion["choices"][0]["message"]
[...]
#Risposta di ChatGPT (response_message)
{
  'role': 'assistant',
  'content': None,

```

```

    'function_call': {'name': 'set_lps', 'arguments': '{"value":
    'No LPS'}}
  }

```

Codice 4.13: JSON schema per il Function Call con più funzioni

Difetto della attuale Function Calling

La funzionalità di [function calling](#), pur essendo una novità interessante e innovativa, presenta alcune limitazioni che ne influenzano l'utilizzo, specialmente durante il periodo in cui è stato sviluppato il progetto.

Innanzitutto, un primo limite consiste nel fatto che ogni coppia di messaggi user-assistant (o tre messaggi, includendo il messaggio di sistema) è indipendente dalle conversazioni precedenti. Questo significa che ChatGPT non ha memoria di ciò che è stato discusso in passato e quindi non può fornire risposte contestualizzate. Per mitigare questa limitazione, il messaggio del sistema che ha il compito di istruire e addestrare ChatGPT deve essere il più completo possibile fin dall'inizio, al fine di evitare risposte confuse o fuori contesto.

In secondo luogo, un'altra limitazione risiede nella chiamata delle funzioni. In ogni conversazione, se viene effettuata una chiamata a una funzione, verrà eseguita al massimo una sola funzione, anche se il sistema richiede esplicitamente l'esecuzione di tutte le funzioni pertinenti alla situazione. Questa limitazione può influenzare la completezza delle risposte generate dall'assistente e la capacità di adattarsi a scenari complessi.

Infine, un terzo limite è che una stessa funzione non può essere chiamata più volte all'interno di una singola conversazione. Questo potrebbe limitare la flessibilità nel trattare situazioni in cui una stessa azione deve essere ripetuta o adeguata in risposta a vari input utente.

Soluzione adottata

La complessità delle richieste che coinvolgono la necessità di richiamare più volte la stessa funzione o di combinare diverse funzioni ha portato a difficoltà nella generazione delle risposte corrette. Ad esempio, richieste come "Impostami la colonna LPS con valori LPS e No LPS" che richiedono di richiamare una stessa funzione più volte o richieste come "Impostami la colonna LPS in No LPS e la colonna ISIA in No ISIA" che combinano diverse funzioni hanno dimostrato di essere problematiche da gestire.

Di conseguenza, il [JSON schema](#) alla fine implementato, come mostrato nel [Codice 4.14](#), è stato soggetto a modifiche rispetto a quello presente nella documentazione ufficiale di OpenAI [24]. Una delle differenze principali risiede nell'implementare un'unica funzione; ogni parametro di questa funzione corrisponde a una colonna, e i valori di ciascun parametro sono stati organizzati in forma di array.

```

functions = [
  {
    "name": "multi_func",
    "description": "funzione per impostare i diversi campi",
    "parameters": {
      "type": "object",
      "additionalProperties": False,
      "properties": {
        "set_lps": {

```

```

        "description": "imposta LPS",
        "type": "array",
        "items":{
            "type": "string",
            "enum":["LPS", "No LPS"],
        }
    },
    "set_isia": {
        "description": "imposta ISIA",
        "type": "array",
        "items":{
            "type": "string",
            "enum":["ISIA", "No ISIA"],
        }
    }
    [...]
},
},
]

#Esempio di risposta che si ottiene
{
    'role': 'assistant',
    'content': None,
    'function_call': {
        'name': 'multi_func',
        'arguments': '{
            "set_lps": ["LPS", "No LPS"],
            "set_ISIA": ["ISIA "]
        }
    }
}

```

Codice 4.14: JSON schema per il Function Call

Infine, come spiegato nella [Sezione 4.4.4](#), per ottenere il formato **JSON** desiderato utilizzando la risposta ottenuta (vedi [Codice 4.14](#)), è possibile richiamare le funzioni utilizzando i parametri "set_lps" e "set_isia". Questo processo è illustrato nel [Codice 4.15](#).

```

import openai
openai.api_key = [myKey]

def set_isia(userValue):
    [...]
    for value in userValue:
        if value=="ISIA" :
            systemValue.append(True)
        if value=="No ISIA" :
            systemValue.append(False)
    info = {
        "systemLabel": "is_isia",
        "userLabel": "ISIA",
        "systemValue": systemValue,
    }

```

```
        "userValue": userValue,
        "applyChanges": True,
        "disabled": False
    }
    return info

def set_lps(userValue: list):
    [...]
    for value in userValue:
        if value=="LPS" :
            systemValue.append(True)
        if value=="No LPS" :
            systemValue.append(False)
    info = {
        "systemLabel": "is_lps",
        "userLabel": "LPS",
        "systemValue": systemValue,
        "userValue": userValue,
        "applyChanges": True,
        "disabled": False
    }
    return info

def run_conversation():
    messages = [
        {"role": "system", "content": [...],},
        {"role": "user", "content": [...]},
    ]
    completion: dict = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=messages,
        functions=functions,
        temperature=0.0,
        function_call="auto",
    )
    response_message = completion["choices"][0]["message"]

    if response_message.get("function_call"):
        available_functions = {
            "set_lps": set_lps,
            "set_isia": set_isia,
            [...]
        }
        output= []
        arguments: dict = json.loads(response_message["function_call"]
        ["arguments"])

        for function, value in arguments.items():
            if (value):
                function_name = function
                function_to_call = available_functions[function_name]
                function_param: list = value
                function_response = function_to_call(
                    userValue=function_param,
```

```
    )  
    output.append(function_response)  
pprint(json.dumps(output))
```

Codice 4.15: Richiesta alle API di OpenAI usando il Function Calling

Capitolo 5

Verifica e validazione

In questo capitolo, viene descritta la fase di verifica e validazione del prodotto. Il focus principale sarà sull'addestramento di ChatGPT e sulla verifica dell'accuratezza delle risposte prodotte attraverso le conversazioni.

5.1 Verifica

La verifica si concentra soprattutto sull'accuratezza delle risposte ottenute e di conseguenza sull'addestramento del modello "gpt-3.5-turbo" prima di lasciarlo "dialogare" con l'utente. Sebbene ChatGPT sia intrinsecamente un [chatbot](#)^[8], ovvero un software addestrato per simulare una conversazione con il linguaggio naturale [5], la sfida si pone nel trasformarlo in uno strumento capace di produrre risposte strutturate. OpenAI ha introdotto la funzionalità del [function calling](#) con l'obiettivo di supportare questa esigenza [25], ma a parere personale (come evidenziato anche nella [Sezione 4.4.6](#)), tale funzionalità è ancora in uno stato prematuro.

5.1.1 Addestramento del chatbot

In ogni conversazione, viene inviato sempre un messaggio al ruolo "sistema" che spiega a ChatGPT il contesto della conversazione e ciò che si aspetta come risposta. Questo messaggio svolge un ruolo importante nell'addestramento, poiché influisce sulle risposte di ChatGPT. Di seguito sono elencate le informazioni che viene incluso in questo messaggio di addestramento:

- **Ruolo:** spiegare il ruolo del [chatbot](#) nella conversazione, ad esempio "assistente per il filtraggio di tabelle";
- **Compito:** si intende per le operazioni che verranno chieste di fare, come il filtraggio, ordinamento di una o più colonne e come deve rispondere;
- **Guida alle richieste:** anticipa le possibili richieste degli utenti, fornendo degli esempi, e quali dati deve estrapolare nella frase dell'utente, specificare inoltre gli errori di scrittura che possono fare gli utenti e come comportarsi di conseguenza;
- **Risposte non volute:** specificare, se ci sono, parole chiave, lettere e simboli non voluti, e quindi da escludere nelle risposte;

- **Data attuale:** se necessario, fornire la data e l'ora, poiché ChatGPT non ha accesso a informazioni di tempo reale;
- **Parole chiave:** elencare, se ci sono, le parole chiave e il comportamento che deve assumere il [chatbot](#) di fronte a queste parole.

5.1.2 Controllo dei Token

I token sono fondamentalmente le unità di misura delle parole o dei simboli nei modelli di linguaggio come [gpt-3.5-turbo](#) [5]. Ogni token può rappresentare una singola lettera, una parola o anche una parte di una parola, a seconda della lingua e della complessità. Nel modello [gpt-3.5-turbo](#) usato nel progetto, il limite massimo di token accettato per una singola richiesta sono 4096 (circa 11000 caratteri) [26][6].

In una chiamata [API](#) con la funzionalità di [function calling](#) di [gpt-3.5-turbo](#), i token contati includono:

- Token nei messaggi utente, quindi ogni token nel testo del messaggio utente;
- Token nei messaggi di sistema, quindi ogni i token nel testo del messaggio sistema;
- Token nell'output generato dal modello, quindi se l'output generato dal modello include testo, i token in questo testo vengono conteggiati nella lunghezza totale;
- Token nei parametri di chiamata alla funzione, quindi vengono conteggiati i parametri di chiamata alla funzione passati che contengono testo;
- Token nei nomi delle funzioni, quindi vengono conteggiati i token nei nomi delle funzioni nel [JSON schema](#) di chiamata.

In caso di superamento dei token, la richiesta verrà rifiutata e non verrà elaborata. Per questo motivo è stato implementato un controllo del numero di caratteri inseriti, non permettendo di inviare richieste in caso di superamento del limite impostato.

5.1.3 Controllo delle risposte

Sono state create funzioni specifiche per controllare e correggere i dati restituiti da ChatGPT. Queste funzioni agiscono sulla formattazione e il tipo di dati, assicurando che siano coerenti e pronti per l'uso. Alcuni esempi di queste funzioni sono illustrati nel [Codice 4.15](#). Queste funzioni svolgono diverse operazioni come la conversione da numero a stringa, il cambio in maiuscolo di una stringa o la creazione di liste. In sostanza, queste funzioni garantiscono che i dati siano presentati in modo uniforme e corretto, in modo che possano essere utilizzati nel resto del processo senza problemi.

5.2 Analisi statica e dinamica

Sono state effettuate anche l'analisi statica e dinamica del codice.

- **Analisi statica:** l'analisi statica è stata focalizzata sulla revisione del codice sorgente senza eseguirlo effettivamente. Durante questa fase, il codice è stato esaminato manualmente o utilizzando strumenti automatizzati per individuare potenziali problemi come errori di sintassi, errori di logica, potenziali vulnerabilità di sicurezza e non conformità alle linee guida di codifica. Sono stati applicati standard di programmazione e convenzioni aziendali per garantire la coerenza e la qualità del codice;

- **Analisi dinamica:** l'analisi dinamica è stata eseguita eseguendo effettivamente il codice in un ambiente controllato. I test sono stati fatti manualmente.

5.3 Validazione

La fase di validazione, che mira a garantire che il prodotto sia allineato alle aspettative e risponda ai requisiti stabiliti, è stata condotta in collaborazione con il proponente. Durante questa fase, il proponente ha confermato che le funzionalità implementate sono operative e corrispondono alle attese. L'obiettivo principale è stato quello di verificare che il software soddisfacesse i criteri di accettazione concordati. Inoltre sono stati inseriti commenti in lingua inglese all'interno del codice per facilitare una comprensione rapida e agevole del funzionamento delle varie parti del software.

Capitolo 6

Conclusioni

In questo capitolo vengono riportati il consuntivo finale delle attività, gli obiettivi raggiunti, i requisiti soddisfatti e una valutazione personale.

6.1 Consuntivo finale

Di seguito, nella [Tabella 6.1](#), viene riportato il consuntivo delle ore e attività.

Nel complesso, lo sviluppo del progetto è avanzato in linea con quanto previsto, ad eccezione delle fasi di analisi e pianificazione, le quali hanno richiesto solamente la metà del tempo stimato in quanto i casi d'uso sono risultati abbastanza chiari e facilmente individuabili.

Ore pianificate	Ore svolte	Descrizione dell'attività
92	84	Formazione sulle tecnologie
40	40	Formazione Typescript
20	20	Formazione React
16	8	Formazione servizi OpenAI
16	16	Formazione API Riskapp
40	20	Analisi e pianificazione
120	140	Sviluppo
8	8	Implementazione di uno spazio di inserimento testuale
60	72	Implementazione del prompt nei servizi di OpenAI per creazione delle query
24	28	Implementazione di uno spazio di visualizzazione e modifica dei dati elaborati per filtraggio dei dati
28	32	Utilizzo dei dati per filtraggio della tabella
48	36	Collaudo Finale
24	24	Stesura dei test
16	12	collaudo e verifica
8	0	Stesura della documentazione
300	280	Totale ore

Tabella 6.1: Distribuzione delle ore svolte

6.2 Raggiungimento degli obiettivi

Di seguito, nella [Tabella 6.2](#), viene mostrato lo stato del raggiungimento degli obiettivi fissati, ovvero tutti.

Codice	Descrizione	Stato
O01	Presenza di uno spazio di inserimento testuale per interagire con il servizio di intelligenza artificiale.	Completato
O02	Creazione del prompt da usare nei servizi di OpenAI per la creazione delle query dal testo.	Completato
O03	Possibilità di visualizzare e modificare i dati individuati che verranno utilizzati per il filtraggio.	Completato
O04	Utilizzo dei risultati di OpenAI per filtrare la tabella.	Completato
O05	Presenza degli esempi sull'utilizzo del servizio implementato.	Completato
D01	Possibilità di correzione del filtraggio precedente con una nuova ricerca testuale.	Completato
F01	Possibilità dell'inserimento vocale.	Completato

Tabella 6.2: Obiettivi raggiunti

6.3 Soddisfacimento dei requisiti

Di seguito nelle [Tabella 6.3](#) e [Tabella 6.4](#), [Tabella 6.5](#) vengono riportati i requisiti e i loro soddisfacimento.

Tabella 6.3: Tabella del soddisfacimento dei requisiti qualitativi

Requisito	Descrizione	Stato
RQD-1	Il codice deve essere ben commentato al fine di garantire la comprensione, la manutenibilità e la collaborazione all'interno del team di sviluppo. I commenti devono essere chiari, concisi e fornire spiegazioni adeguate sulle parti complesse del codice.	Soddisfatto
RQD-2	Il riconoscimento vocale deve essere implementato in modo che sia disponibile nella maggior parte dei browser moderni.	Soddisfatto

Tabella 6.4: Tabella del soddisfacimento dei requisiti di vincolo

Requisito	Descrizione	Stato
RVO-1	La parte front-end deve essere sviluppata utilizzando il linguaggio di programmazione Typescript	Soddisfatto
RVO-2	La parte backend deve essere sviluppata utilizzando il linguaggio di programmazione Python	Soddisfatto
RVO-3	Deve essere utilizzata la libreria Ant Design per una coerenza nel design e nell'aspetto dell'applicazione.	Soddisfatto

Tabella 6.5: Tabella del soddisfacimento dei requisiti funzionali

Requisito	Descrizione	Stato
RFN-1	Solo utenti autorizzati devono poter accedere al servizio di filtraggio testuale	Soddisfatto
RFN-2	Il servizio di filtraggio testuale deve essere accessibile tramite un bottone dedicato.	Soddisfatto
RFN-3	Tutte le funzionalità del servizio devono essere contenute all'interno di una finestra modale .	Soddisfatto
RFN-4	L'interfaccia deve includere un'area di testo appositamente dedicata all'inserimento della richiesta di filtraggio	Soddisfatto
RFN-5	L'area di testo per l'inserimento deve avere una lunghezza massima specificata e notificare l'utente in caso di superamento.	Soddisfatto
RFN-6	Il pulsante di invio deve essere attivo solo se è stato inserito del testo nell'area dedicata.	Soddisfatto
RFN-7	L'utente deve poter resettare i valori inseriti nell'area di testo mediante un apposito bottone.	Soddisfatto
RFN-8	All'interno della finestra modale , deve essere fornita una guida sull'uso del servizio e suggerimenti sulla struttura del testo per il filtraggio.	Soddisfatto
RFN-9	Dovrebbero essere disponibili degli esempi di testo predefiniti che l'utente può scegliere per provare il filtraggio.	Soddisfatto
RFN-10	L'utente deve poter visualizzare tutti i filtri individuati e avere il controllo per selezionare quali filtri desidera applicare.	Soddisfatto
RFD-11	L'utente deve poter ripetere la richiesta di filtraggio, apportando modifiche al testo iniziale.	Soddisfatto
RFZ-12	Possibilità di utilizzare l'inserimento vocale come alternativa per l'input testuale nel servizio.	Soddisfatto

6.4 Valutazione personale

Questa esperienza è stata davvero unica e illuminante dal mio punto di vista. Mi ha permesso di immergermi in un mondo di strumenti e risorse utili per lo sviluppo di prodotti software, che prima erano sconosciuti per me. Ho imparato come affrontare la sfida di implementare nuove funzionalità in un contesto già esistente, cogliendo l'importanza di mantenere l'omogeneità con il sistema preesistente. In particolare, ho sviluppato una maggiore consapevolezza nel gestire prodotti sviluppati da altri e nell'integrare nuove soluzioni in modo coerente.

Nel complesso sono stato molto soddisfatto del lavoro svolto, specialmente per la possibilità di sviluppare con tecnologie così innovative.

Acronimi e abbreviazioni

AI [Artificial Intelligence](#). 2, 3, 55

API [Application Program Interface](#). 2, 10, 22, 26–29, 35, 38, 46, 55

JSON [JavaScript Object Notation](#). 29, 35, 36, 42

RTK [Redux Toolkit](#). 29, 34

UI [User Interface](#). 21, 26

UML [Unified Modeling Language](#). 7, 56

VS Code [Visual Studio Code](#). 17

Glossario

AI *Artificial Intelligence*, detto *AI* è una disciplina che studia se e in che modo si possano realizzare sistemi informatici intelligenti in grado di simulare la capacità e il comportamento del pensiero umano [18]. 53

API in informatica con il termine *Application Programming Interface API* si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 53

Backend è comunemente utilizzato nel contesto dello sviluppo software e si riferisce alla parte di un'applicazione o di un sistema informatico che gestisce le operazioni non visibili agli utenti finali. Il backend è responsabile dell'elaborazione dei dati, della gestione del database, della logica dell'applicazione, della sicurezza e delle comunicazioni con il **frontend** [12]. 3, 5, 16, 26, 27, 29, 35

Chatbot è un software progettato per simulare una conversazione con un essere umano. Lo scopo principale di questi software è quello di simulare un comportamento umano; a volte sono definiti anche agenti intelligenti e vengono usati per vari scopi come per la guida in linea e per rispondere alle FAQ degli utenti che accedono a un sito [2]. 45, 46

ECMAScript è uno standard per la definizione del linguaggio di programmazione JavaScript. È un linguaggio di scripting utilizzato principalmente per sviluppare applicazioni web e, negli ultimi anni, è stato esteso anche per l'uso in applicazioni server, mobile e desktop [21]. 19

Endpoint è un punto di accesso specifico o una risorsa all'interno di un servizio web o di un'applicazione che consente agli sviluppatori di interagire con il sistema per effettuare richieste e ottenere risposte. Gli endpoint di API fungono da intermediari tra un'applicazione e il server o il servizio che ospita i dati o le funzionalità richieste [7]. 35

Frontend si riferisce alla parte di un'applicazione software o di un sito web che interagisce direttamente con l'utente, è responsabile dell'interfaccia utente, della presentazione dei contenuti e dell'interazione con l'utente [12]. 16, 20, 23, 26, 27, 29, 31, 55

Function Calling permette ai modelli gpt-4-0613 e gpt-3.5-turbo-0613 di comprendere e rispondere alle richieste degli sviluppatori attraverso una forma più strutturata e orientata alla chiamata di funzioni [14]. 29, 39, 41, 45, 46

gpt-3.5-turbo è una variante avanzata del modello di linguaggio GPT-3 sviluppato da OpenAI. Si tratta di un modello di intelligenza artificiale basato su trasformatori che è stato addestrato su una vasta quantità di testo da comprendere e generare testo naturale [5]. 38, 46

Hook sono funzioni che consentono agli sviluppatori di "agganciarsi" allo stato di React e alle caratteristiche del ciclo di vita dei componenti delle funzioni. 22, 30, 33–35

Insuretech identifica praticamente tutto ciò che è innovazione *technology – driven* in ambito assicurativo: software, applicazioni, startup, prodotti, servizi, modelli di business [17]. 1

JSON schema è uno standard per la definizione di schemi di dati in formato JSON (JavaScript Object Notation). Fornisce un modo per descrivere la struttura, le restrizioni e la validazione dei dati JSON in un formato specifico, sono utilizzati per garantire che i dati JSON siano conformi a determinate specifiche e che soddisfino i requisiti di validità [20]. 29, 39–41, 46

Mockup è una rappresentazione visiva di un prodotto o di un sistema in fase di sviluppo, spesso creata per scopi di prototipazione, progettazione o comunicazione. Sono utilizzati per visualizzare e testare l'aspetto e il funzionamento di un'idea, di un'applicazione software, di un sito web, di un prodotto fisico o di qualsiasi altro progetto, prima che venga effettivamente realizzato. 18

Modale è un componente grafico sovrapposto che appare sopra il contenuto principale della pagina, attirando l'attenzione dell'utente su un'azione specifica o su un'interazione. In questo contesto, la modale funge da contenitore per le nuove funzionalità e offre un'esperienza di utilizzo focalizzata e strutturata. 3, 7–9, 14, 15, 25–28, 31, 35, 51

Open source si riferisce a un approccio di sviluppo software in cui il codice sorgente di un'applicazione o di un programma è reso disponibile al pubblico e può essere utilizzato, studiato, modificato e distribuito liberamente da chiunque. 19–21, 33

Query è una richiesta di estrazione specifica di dati da una tabella di database o da un insieme di dati tabellari. Queste query vengono utilizzate per estrarre solo le righe di dati che soddisfano determinati criteri o condizioni specifiche, eliminando le righe che non corrispondono ai criteri [28]. 3, 4, 23, 35, 49, 50

UML in ingegneria del software *UML, Unified Modeling Language* è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 53

Riferimenti bibliografici

Siti web consultati

- [1] *Ant Design*. URL: <https://ant.design/> (cit. alle pp. 16, 21, 31).
- [2] *Chat bot*. URL: https://it.wikipedia.org/wiki/Chat_bot (cit. a p. 55).
- [3] *ChatGPT*. URL: <https://chat.openai.com/> (cit. alle pp. 3, 22).
- [4] *ChatGPT Create completion*. URL: <https://platform.openai.com/docs/api-reference/completions/create> (cit. a p. 38).
- [5] *ChatGPT Models*. URL: <https://platform.openai.com/docs/models/overview> (cit. alle pp. 38, 45, 46, 56).
- [6] *ChatGPT Models Pricing*. URL: <https://openai.com/pricing> (cit. a p. 46).
- [7] *Cos'è un endpoint API?* URL: <https://www.cloudflare.com/it-it/learning/security/api/what-is-api-endpoint/> (cit. a p. 55).
- [8] *CSS*. URL: <https://it.wikipedia.org/wiki/CSS> (cit. a p. 20).
- [9] *Difference between TypeScript and JavaScript*. URL: <https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/> (cit. a p. 31).
- [10] *ECMAScript*. URL: <https://it.wikipedia.org/wiki/ECMAScript> (cit. a p. 19).
- [11] *Figma*. URL: <https://www.figma.com/> (cit. a p. 18).
- [12] *Front-end e back-end*. URL: https://it.wikipedia.org/wiki/Front-end_e_back-end (cit. a p. 55).
- [13] *Function and Class Components*. URL: <https://legacy.reactjs.org/docs/components-and-props.html> (cit. a p. 30).
- [14] *Function Calling*. URL: <https://platform.openai.com/docs/guides/gpt/function-calling> (cit. alle pp. 26, 56).
- [15] *Function calling updates*. URL: <https://openai.com/blog/function-calling-and-other-api-updates> (cit. a p. 29).
- [16] *Github*. URL: <https://github.com/> (cit. a p. 18).
- [17] *Insurtech: cos'è e come cambierà il mondo delle assicurazioni*. URL: <https://www.insuranceup.it/it/scenari/insurtech-che-cos-e-e-quali-sono-i-suoi-pilastri/> (cit. a p. 56).

- [18] *Intelligenza artificiale*. URL: https://it.wikipedia.org/wiki/Intelligenza_artificiale (cit. a p. 55).
- [19] *JavaScript*. URL: <https://www.javascripttutorial.net/> (cit. a p. 31).
- [20] *JSON Schema*. URL: <https://json-schema.org/> (cit. a p. 56).
- [21] *Mission - ECMAScript*. URL: <https://www.ecma-international.org/mission/> (cit. a p. 55).
- [22] *NodeJS*. URL: <https://it.wikipedia.org/wiki/Node.js> (cit. a p. 21).
- [23] *OpenAI*. URL: <https://openai.com/> (cit. alle pp. 2, 4, 22).
- [24] *OpenAI Cookbook*. URL: <https://github.com/openai/openai-cookbook> (cit. alle pp. 40, 41).
- [25] *OpenAI Documentation*. URL: <https://platform.openai.com/docs/introduction/overview> (cit. alle pp. 20, 22, 28, 38, 39, 45).
- [26] *OpenAI Tokenizer*. URL: <https://platform.openai.com/tokenizer> (cit. a p. 46).
- [27] *Python*. URL: <https://www.python.org/> (cit. alle pp. 16, 20).
- [28] *Query*. URL: <https://it.wikipedia.org/wiki/Query> (cit. a p. 56).
- [29] *React*. URL: <https://react.dev/> (cit. alle pp. 20, 23, 27, 32).
- [30] *React Redux Toolkit*. URL: <https://redux-toolkit.js.org/rtk-query/overview> (cit. alle pp. 23, 29, 34).
- [31] *React Speech Recognition*. URL: <https://www.npmjs.com/package/react-speech-recognition> (cit. alle pp. 22, 33).
- [32] *React: Functional Components vs Class Components*. URL: <https://josipmisko.com/posts/react-functional-vs-class-components> (cit. a p. 30).
- [33] *RiskApp*. URL: <https://riskapp.it/> (cit. alle pp. 1, 10).
- [34] *Slack*. URL: <https://slack.com/intl/it-it> (cit. a p. 17).
- [35] *TypeScript*. URL: <https://it.wikipedia.org/wiki/TypeScript> (cit. alle pp. 16, 19, 23, 31).
- [36] *Visual Studio Code*. URL: <https://code.visualstudio.com/> (cit. a p. 17).
- [37] *WEB Speech API*. URL: <https://wicg.github.io/speech-api/> (cit. a p. 33).