Politecnico di Torino

Politecnico di Torino

Ingegneria Informatica

A.a. 2023/2024

Sessione di laurea Aprile 2024

Approccio basato su linguaggio naturale per l'esplorazione dei dati del CRM

Relatori:

Luigi De Russis

Giovanni Campolo

Alberto Francia

Candidato:

Gabriele Castelli

Ringraziamenti

Ringrazio il Professor De Russis, Giovanni Campolo e Alberto Francia, i miei relatori, per la loro disponibilità durante il percorso di tesi e li ringrazio, inoltre, per i loro contributo alla stesura di questo documento.

Esprimo la mia profonda gratitudine verso mia madre, mio padre e mia sorella per il loro incondizionato amore e sostegno nel mio percorso di vita. Senza loro non sarebbe mai stato possibile il raggiungimento di questo importante traguardo.

Sono immensamente grato ai miei nonni e zii per il calore e l'affetto che hanno sempre riversato su di me.

Ringrazio la mia ragazza per essere stata una presenza costante e per aver riempito la mia vita di allegria.

Esprimo gratitudine ai miei amici per aver condiviso con me la loro gioia di vivere durante il percorso accademico.

Questo traguardo rappresenta il culmine di un percorso non solo mio, ma condiviso con tutti voi, che mi avete accompagnato e supportato.

Indice

Elenco delle figure

Capitolo 1

Introduzione

1.1 Contesto

All'inizio degli anni '70, le valutazioni sulla soddisfazione dei clienti venivano realizzate dalle aziende tramite dei sondaggi cartacei o tramite la rete di vendita. Non esisteva una tecnologia che permettesse di acquisire e collezionare tali dati in modo differente, sebbene ne fosse chiara la potenzialità. Con gli anni, l'evoluzione tecnologica ha, dapprima, permesso l'utilizzo di fogli di calcolo per la classificazione della clientela, per esempio con Microsoft Excel, e, successivamente, fornito software e strategie sempre più complesse che richiedevano il raccoglimento di tutte le informazioni in un singolo luogo.

Nascono, così, i primi CRM (Customer Relationship Management). Tale termine non sta a indicare, esclusivamente, l'uso di un software per ottimizzare i processi aziendali, ma definisce un concetto più ampio. Un CRM è una strategia di business, che si serve della tecnologia per mettere il cliente al centro, comprenderne i bisogni e trasformarli in opportunità di mercato per l'azienda stessa.

I CRM raccolgono in un unico punto le informazioni provenienti dal sito Web dell'azienda, per esempio tramite i form di contatto e live chat, da telefonate e appuntamenti di lavoro, da email, da ricerche di mercato, dai social media della azienda e così via. I dati raccolti non vengono usati così come sono, ma necessitano di essere raffinati, ripuliti e raggruppati per essere utili. Ad esempio, quando l'azienda riceve una mail dall'esterno, il sistema CRM deve dapprima estrarre le informazioni chiave, quindi il mittente, l'oggetto, il corpo e l'eventuale allegato. Dopodiché, il corpo della mail viene ripulito dei tag HTML. Infine, le informazioni estratte vengono memorizzate all'interno del sistema CRM. In questo modo si avranno tutti i dati utili memorizzati in un'unica sede.

L'uso del CRM permette di conoscere meglio il proprio target di clienti, migliorare le relazioni con la clientela e generare offerte commerciali personalizzate per il

cliente. Molti CRM prevedono strumenti di automatizzazione del processo di vendita, che consentono di vendere in modo più rapido. Altri ancora, prevedono l'utilizzo dell'intelligenza artificiale per suggerire al venditore le prossime azioni nel processo di vendita.

Un'azienda, utilizzando un CRM, mira a mantenere i propri clienti, favorire la crescita aziendale, aumentare le percentuali di chiusura, mantenere la fidelizzazione e trasformare i propri clienti in promotori, ossia consumatori che lodano l'azienda incoraggiando altre persone a rivolgersi ad stessa. I sistemi CRM offrono una dashboard semplice e personalizzabile per accedere ai dati del cliente, permettendo di visionare, ad esempio, lo stato degli ordini del cliente stesso, lo storico delle interazioni con l'azienda e molto altro, fornendo, quindi, una visione a 360 gradi del cliente. Le precedenti funzionalità permettono ad un utente di accedere velocemente a tutto ciò che che riguarda il cliente con la finalità di suggerire proposte ad hoc. Ne consegue un aumento della produttività dovuta alla reperibilità delle informazioni in un singolo sistema e alla conseguente creazione di servizi personalizzati in base alle esigenze che emergono dai dati memorizzati.

In passato, implementare una soluzione CRM era molto oneroso per le aziende, poiché era necessario comprare un software proprietario, che era estremamente difficile da gestire e mantenere. La gestione di tali soluzioni CRM richiedeva competenze tecniche specifiche . Quindi, l'azienda necessitava di tali figure che doveva formare o assumere. Queste figure dovevano essere in grado di configurare, personalizzare e mantenere il sistema. Non tutte le aziende potevano permettersi tali figure per la gestione delle soluzioni CRM. Ciò portava all'impossibilità di utilizzare una soluzione CRM per i piccoli imprenditori. Il passaggio al Cloud può essere considerato uno degli aspetti più significativi degli ultimi anni, col vantaggio di non dover più installare software su centinaia o migliaia di computer, ma poter accedere al software e ai servizi su un ambiente online sicuro. Inoltre, il movimento verso il Cloud ha avuto anche l'effetto di rendere l'uso del CRM più accessibile a singoli imprenditori e piccoli team, che possono stipulare abbonamenti per il software in modalità SaaS (Software as a Service), lasciando l'onere della gestione al fornitore. Le soluzioni Cloud CRM vengono create, nel contesto Microsoft, tramite la piattaforma PowerPlatform che segue il paradigma del low-code. Tale paradigma offre la possibilità di implementare soluzioni scrivendo poco codice. La PowerPlatform fornisce una serie di componenti che permettono la creazione di CRM in forma di model-driven app. Dynamic365 è una collezione di model-driven app standard usate per definire sistemi CRM in contesti specifici, come ad esempio gestione dei clienti indirizzata alla vendita.

Oggigiorno, la maggior parte delle aziende utilizzano soluzioni basate su CRM per gestire le relazioni coi clienti nell'ambito della gestione di una attività commerciale, del marketing, della produzione e così via. Laddove Dynamics365 non offra una soluzione standard per l'ambito di interesse è possibile crearne una da zero attraverso

la PowerPlatform.

1.2 Obiettivo tesi

All'interno di un CRM sono contenute tutte le informazioni sul cliente che siano utili al business. Tali informazioni possono essere rappresentate in forma di report che contiene una visione di insieme su un contesto. Ad esempio, un report potrebbe fornire informazioni generali sull'andamento della chiusura delle opportunità. A partire da questo report, un utente potrebbe voler eseguire delle analisi più approfondite utilizzando le operazioni di ricerca. In base al livello di dettaglio che l'utente vuole raggiungere, si possono utilizzare tre strumenti di ricerca: ricerca di Dataverse, ricerca rapida e ricerca avanzata. La ricerca di Dataverse trova qualsiasi colonna che contiene una delle parole inserite. La ricerca rapida trova le colonne che contengono interamente l'elemento ricercato. Infine, la ricerca avanzata è lo strumento di ricerca più specifico. Ad esempio, con la ricerca rapida o di Dataverse si ricerca un elemento in un qualsiasi campo della base dati, mentre con la ricerca avanzata è possibile definire la tabella da interrogare e il valore di uno specifico campo. Quindi, la ricerca avanzata permette di approfondire l'analisi definendo specificatamente le condizioni sui campi indicati. Ad esempio, con la ricerca avanzata è possibile effettuare una ricerca per le opportunità di vendita chiuse a Roma o Torino in uno specifico intervallo temporale. I principali svantaggi di tale tipo di ricerca sono legati alla complessità della costruzione ed al tempo richiesto per definirla. Essi vengono, però, compensati dal valore che i risultati della ricerca forniscono all'analisi.

Nel contesto della ricerca sui dati l'intelligenza artificiale trova un terreno fertile in cui operare. Finora, all'interno dei sistemi CRM è stato possibile effettuare ricerche più o meno avanzate strutturando i filtri in modo classico per ottenere informazioni sulla clientela. Più nel dettaglio, le ricerche permettono di definire con precisione la tabella da interrogare, le condizioni sui campi e le relazioni con altre tabelle. Se, ad esempio, si fosse interessati a tutti i clienti con sede a Roma, la ricerca permette all'utente di selezionare la tabella di partenza, in questo caso la tabella "Cliente", selezionare il campo "Città" e porlo uguale a Roma. Questo modalità permette di ritornare risultati più specifici. Gli avanzamenti dell'intelligenza artificiale hanno aperto nuove possibilità nell'ambito dell'interrogazione sui dati. La tesi si muove sulla possibilità di utilizzare il linguaggio naturale come elemento di interrogazione del CRM. La tesi, quindi, prevede la progettazione di uno strumento di ricerca per interrogazioni puntuali, cioè che ritornino un numero ristretto di dati, su tabelle custom, cioè create dall'utente, o standard, ovvero fornite di default dal CRM, in linguaggio naturale. Tale strumento si basa sulla decodifica di un modello AI di

quanto l'utente stia chiedendo in linguaggio naturale in un formato utilizzabile per la fonte dati.

La scelta del linguaggio naturale offre i seguenti vantaggi:

- un aumento della agilità aziendale. La possibilità di rendere questo meccanismo più semplice permetterebbe un aumento del numero di operatori in grado di utilizzarlo e quindi fornire un apporto rilevante al processo di analisi;
- un risparmio in termini di tempo nella scrittura delle interrogazioni;
- delle analisi approfondite più semplici da realizzare.

Infine, il lavoro di tesi verrà contestualizzato in un caso d'uso con Teams che fornisce una interfaccia allo strumento di ricerca per l'interrogazione dei dati. L'integrazione di tale meccanismo avviene con un chatbot integrato su Teams, che fornisce una interfaccia conversazionale per eseguire le interrogazioni sui dati. La scelta di Teams come interfaccia per l'interrogazione dei dati è legata al largo utilizzo che le aziende fanno di tale piattaforma. Essa viene utilizzata ampiamente come strumento di comunicazione aziendale. In questo modo, il dipendente che necessita di interrogare il CRM può accedere a tale funzionalità in un ambiente familiare, che utilizza giornalmente, senza dover uscire da tale piattaforma. Se, ad esempio, un team di vendita esegue una videoconferenza su Teams per discutere delle indagini di mercato ed i membri necessitano di accedere a informazioni più approfondite, ciò è possibile usando la stessa piattaforma.

Ricapitolando, la progettazione si propone di definire la logica per risolvere richieste dell'utente che siano puntuali, cioè che ritornino un numero ristretto di dati. In generale, quindi, il modello AI deve interpretare le richieste dell'utente, tradurle in un formato comprensibile, interrogare la base di dati del CRM ed, infine, interpretare il risultato in linguaggio naturale. La tesi si sviluppa principalmente in tre fasi:

1. La prima fase della tesi prevede la progettazione di tale strumento di ricerca, nel caso ristretto di ricerca sui dati di una singola tabella, che sia custom o standard. Inizialmente si è progettato uno strumento di ricerca che fosse in grado di utilizzare un modello AI generativo, GPT-4-32K, per tradurre le interrogazioni dell'utente che richiedessero dati presenti su una sola tabella. Non era quindi prevista la possibilità di passare al modello AI interrogazioni più complesse che richiedessero la gestione di relazioni tra tabelle. Questa opportunità di sviluppo è stata lasciata alla fase successiva.

2. la seconda fase prevede la progettazione di una possibile estensione di tale strumento in modo tale da utilizzare il modello AI per tradurre richieste che richiedano l'utilizzo di più tabelle,che siano custom o standard. In questo

modo è possibile richiedere, ad esempio, quali siano i clienti che vivono a Roma e che hanno una macchina FIAT interrogando le tabelle "Cliente" e "Macchina'.

3. La terza e ultima fase di progettazione prevede la contestualizzazione del meccanismo di ricerca previsto dalla tesi in un caso d'uso con Teams.

Infine, la tesi si propone di realizzare una valutazione del meccanismo di traduzione del linguaggio naturale ed un confronto con la tecnologia introdotta da Microsoft a Marzo 2023, chiamata Copilot. Copilot offre varie funzionalità, ma la più interessante al nostro contesto è legata alla sua capacità di eseguire anch'egli operazioni di ricerca basate sul linguaggio naturale. Tali operazioni di ricerca su Copilot erano inizialmente possibili solo su tabelle standard, mentre la tesi si propone di abbattere questo limite. In generale, Copilot è stato introdotto in tutti gli strumenti della Microsoft PowerPlatform, con l'obiettivo di utilizzare l'AI per rendere più semplice le operazioni. Ad esempio, l'utilizzo di Copilot permette di facilitare le operazioni per:

- l'interrogazione dei dati di un CRM utilizzando il linguaggio naturale.
- la creazione di flussi automatici, cioè una serie di operazioni svolte automaticamente, utilizzando il linguaggio naturale.
- la creazione di pagine web utilizzando il linguaggio naturale.
- la creazione di suggerimenti per le risposte alle email e per le prossime azioni di vendita da intraprendere.

In questa sede, il confronto ricade unicamente sull'interrogazione dei dati tramite Copilot, che promette di raggiungere i medesimi obiettivi proposti da questa tesi, ma con i limiti che inizialmente non fosse possibile realizzare interrogazioni su tabelle custom e che richieda una configurazione manuale a più livelli. Tale configurazione permette di definire le tabelle ed i campi da utilizzare per generare la risposta alla richiesta. La tesi si propone di superare i precedenti limiti.

1.3 Struttura della tesi

Nelle prossime pagine analizzeremo nel dettaglio vari step, che prevedono l'apprendimento dei servizi per sviluppare la soluzione, la progettazione della logica e l'implementazione del meccanismo di ricerca:

- Il Capitolo 2 **"PowerPlatform, servizi Azure e Dynamics365"** presenta le tecnologie offerte da Microsoft, che seguono il paradigma del low-code per

la creazione di servizi. Esse sono state usate o studiate durante il percorso di tesi. In particolare, si analizza Dynamics365, che è la piattaforma offerta da Microsoft per definire soluzioni CRM personalizzate. Successivamente, si definiscono le metodologie con le quali è possibile interrogare i dati nei sistemi CRM classici. Infine, si parla di servizi Azure, che sono una serie di cloud based-services offerti da Microsoft per la creazione di soluzioni più potenti.

- Il Capitolo 3 **"Progettazione"** definisce quali possano essere i requisiti logici per la realizzazione dello strumento di ricerca. Definiti i requisiti, il capitolo continua fornendo un possibile processo logico che li soddisfi. Infine, viene mostrato una possibile architettura logica.

- Il Capitolo 4 **"Implementazione"** specifica i concetti chiave per l'implementazione. Definisce, quindi, quali siano i vantaggi dell'uso di Langchain, che è un framework open source usato per semplificare la creazione di applicazioni che utilizzano modelli linguistici di grandi dimensioni(LLM). Il capitolo, inoltre, definisce la sintassi e semantica della fetchXML, che è il formato per interfacciarsi al Dataverse, che è la fonte dati del CRM. Fornisce la definizione dei connettori ed infine definisce l'architettura del progetto nel dettaglio.

- Il Capitolo 5 **"Valutazione e confronto con Copilot"** delinea i risultati ottenuti dalla soluzione proposta, i limiti e futuri miglioramenti. Infine, presenta Copilot e lo confronta con la soluzione proposta da questa tesi.

- Il Capitolo 6 **"Caso d'uso: Chatbot integrato in Teams"** presenta un possibile caso d'uso della soluzione con Teams. Il capitolo, in particolare, definisce il processo logico del chatbot per l'integrazione con soluzione, le motivazioni ed i vantaggi ed, infine, specifica come è stato implementato.

- Il Capitolo 7 **"Conclusione"** contiene le conclusioni a cui si è giunti al termine del progetto di tesi.

Capitolo 2

PowerPlatform, servizi Azure e Dynamics365

Il capitolo definisce le potenzialità e le caratteristiche della PowerPlatform e di tutti i suoi componenti, di cui parleremo in breve. Viene data maggiore attenzione ai componenti utili per comprendere il contesto in cui si delinea la tesi. In particolare, si parla di Power Apps, che permette la creazione di model-driven apps. Copilot Studio, che permette la creazione di chatbots. Si descrivono, inoltre, il Dataverse, i connettori e i Servizi Azure, che sono essenziali per la creazione di applicazioni. Infine, si definisce la piattaforma Dynamics365, utile per la creazione di CRM.

2.1 PowerPlatform

Microsoft Power Platform è una piattaforma che permette alle organizzazioni di creare soluzioni personalizzate utilizzando un set di strumenti, che seguono il pattern del low-code. Questi strumenti sono PoweBI, Power Apps, Power Automate, Power Virtual Agents a Power Pages, che dettaglieremo nei prossimi paragrafi [1]. Oggigiorno, le aziende devono affrontare nuove sfide legate a:

- aumento dei costi causati da richieste che richiedono una personalizzazione del prodotto sempre più profonda. La loro creazione ed il loro mantenimento richiedono tempo e denaro;

- necessità di ridurre i tempi di attesa per la progettazione. Il team di sviluppo non ha più mesi a disposizione per la creazione di una soluzione e per il suo testing, poiché le richieste aziendali variano velocemente. Le soluzioni che richiedono tempo potrebbero non essere più utili se il tempo impiegato per averle non è consono alla velocità con cui il mercato cambia;

- necessità di scalare il lavoro al fine di aumentare l'efficienza. Se si forniscono strumenti meni complessi, per esempio seguendo il paradigma del low-code, è possibile includere nel lavoro persone mene esperte e con diverse conoscenze. In questo modo, possiamo creare gruppi ibridi di persone che lavorano al progetto, composti da esperti del settore e citizen developer, cioè sviluppatori improvvisati o alle prime armi. Seguendo questo paradigma si è in grado di incrementare l'agilità aziendale.

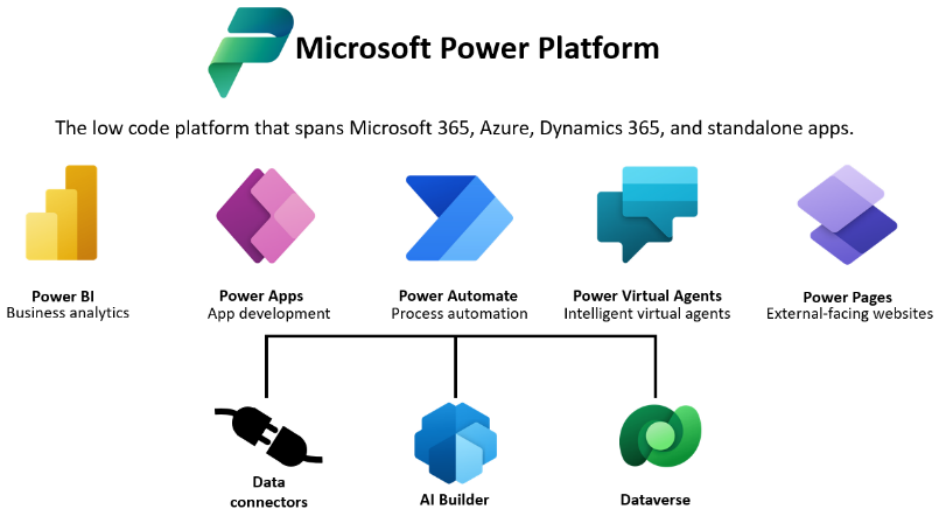

Figura 2.1: Architettura Power Platform

Power Platform si pone l'obiettivo di risolvere le precedenti problematiche, in modo che l'azienda si adattati alla velocità e alle necessità del mercato. Segue un approccio low code che permette di ridurre le tempistiche di lavoro su componenti come schermate, automatismi e definisce, inoltre, circa 900 connettori, che rendono semplice l'integrazione con sistemi interni ed esterni, riducendo i tempi.

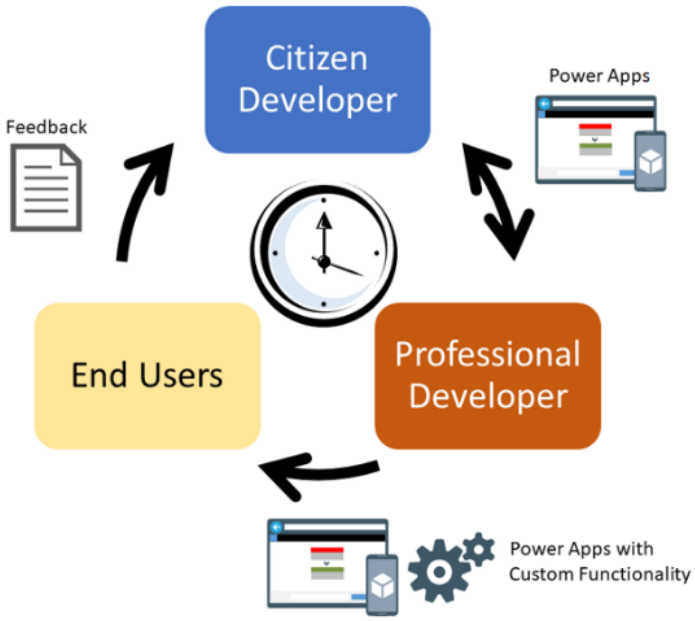

Figura 2.2: Nuovo ciclo di produzione con il pattern low-code

La figura 2.2 mostra gli attori del nuovo ciclo di produzione: l'end user, citizen developer e il professional developer.

L'end user è colui che richiede la realizzazione di una soluzione e fornisce feedback per la personalizzazione;

Il citizen developer è colui che, seguendo un paradigma low-code, è in grado di realizzare la soluzione, facendo uso dei servizi realizzati dal professional developer;

Infine, i professional developers sono coloro che si fanno carico della realizzazione dei servizi complessi, che non sono realizzabili con il pattern low-code, e che, quindi, devono essere realizzati con più attenzione e da chi ha più esperienza;

In definitiva, Power Platform consente alle organizzazioni di creare soluzioni personalizzate, che soddisfino le loro esigenze più facilmente e favorisce l'agilità aziendale per ridurre i tempi di attesa per lo sviluppo di una soluzione. La potenza di Power Platform risiede, anche, nella sua interoperabilità con gli altri strumenti Microsoft. Infatti, permette di creare soluzioni che interagiscono facilmente con Microsoft365, che è uno degli strumenti aziendali più in voga. Permette di creare facilmente applicazioni integrate su Teams, che in fase di pandemia è diventato essenziale in molte organizzazioni. Permette di interagire con le soluzioni Dynamic365 in vari modi. Ad esempio, è possibile creare le model driven application, che sono presenti nelle soluzioni Dynamic365, tramite gli strumenti di Power Platform. Le model driven applications vengono create con lo strumento Power Apps, che

permette di creare dashboard personalizzate con grafici, viste e così via. Le applicazioni, si dicono basate su modello in quanto utilizzano un modello di dati definito dall'utente che viene memorizzato in un servizio Cloud chiamato Dataverse.

Power Apps non è il solo strumento della Power Platform che interagisce con Dynamic 365. Ognuno di essi permette di integrare qualcosa all'interno di Dynamic365. Ad esempio, abbiamo che:

- Power Automate permette l'integrazione di automatismi.
- Power Virtual Agent permette l'integrazione di chatbot, che rispondono alle domande più frequenti, o che fanno uso di AI per rispondere a partire da una basi dati documentale.
- Power Pages permette di integrare siti web personalizzati, per fornire all'utente un modo per visionare i dati rilevanti per il suo business.

L'obiettivo di Power Platform è quello di aggiungere valore, creando app personalizzate, processi automatizzati e chatbot. Gli strumenti di Power Platform possono relazionarsi tra loro per la creazione di soluzioni più complicate, che negli ultimi anni, richiedono, anche, l'integrazione con servizi di intelligenza artificiale. L'AI sta cambiando i paradigmi con cui le società eseguono le attività, semplificando il modo in cui gli addetti ai lavori eseguono i task. In particolare, Microsoft Copilot sfrutta le potenzialità dell'AI per realizzare azioni che precedentemente erano possibili eseguendo una serie di azioni, e adesso sono realizzabili richiedendole in forma di linguaggio naturale. Copilot è usufruibile in tutte le componenti di Power Platform. Se, per esempio, si volesse creare un'app si potrebbe comunicare a Copilot cosa l'app dovrebbe fare e lui creerebbe, autonomamente, ciò che ritiene necessario. Questo paradigma, consente ai developer di essere più produttivi, ma permette anche ai clienti finali di utilizzare l'AI per analizzare in maniera più semplice i dati tramite semplici domande al sistema. Copilot fa uso di Azure, una piattaforma che offre centinaia di servizi Cloud per vari ambiti, che variano dall'archiviazione, alla fruizione di macchine virtuali fino ai servizi che prevedono l'uso dell' intelligenza artificiale. In definitiva, i servizi Azure vengono usati per modernizzare le vecchie soluzioni generate dalla Power Platform o per creare nuove soluzioni sfruttando servizi già implementati da altri. Infine, le seguenti citazioni mostrano esattamente le potenzialità di questa piattaforma.

"If you didn't know this was Power Pages, you might think it's a custom-developed web solution." [2] Risultato: 85% di risparmio costo con Power Pages, 30% di risparmio tempo con Power Pages

"Microsoft has done a great job of connecting Microsoft 365, [Microsoft] Teams, and Microsoft Power Platform applications." Jeff Toler, Senior Manager, Asset Protection Implementation and Support Solutions, Walgreens Boots Alliance [3]

Risultati: 75% di riduzione nel tempo di verifica dell'inventario, 50-80% di riduzione dei tempi di sviluppo

2.1.1 Low Code

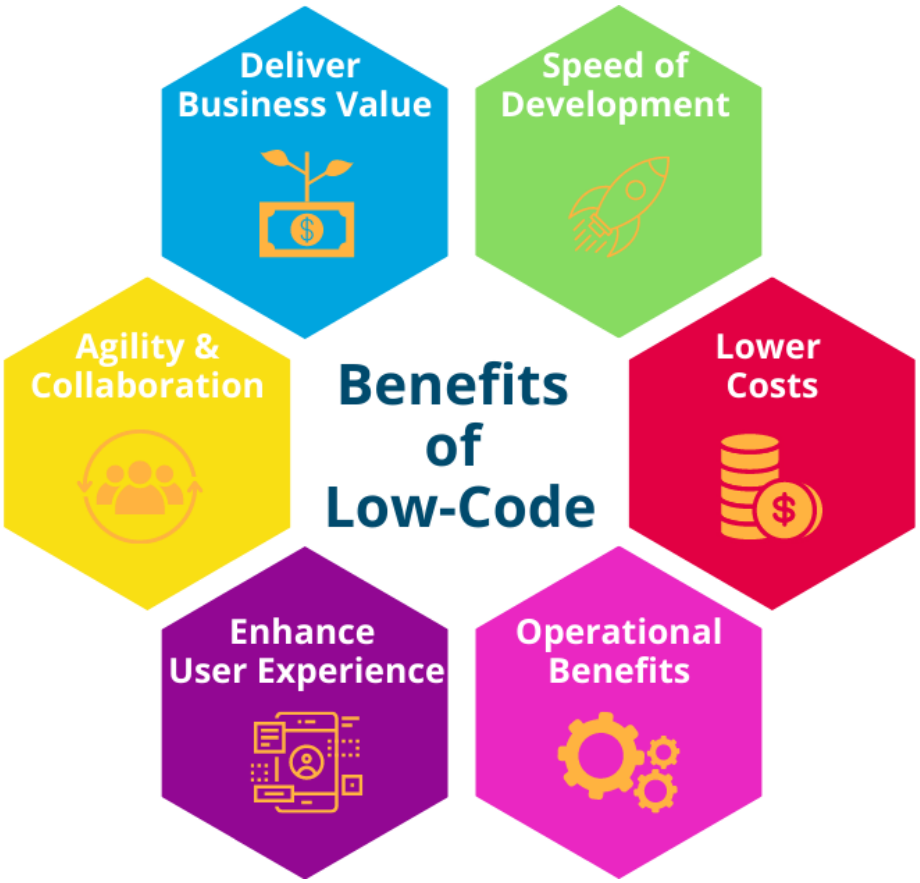

Figura 2.3: Vantaggi approccio low-code

La crescente richiesta di servizi disponibili in poco tempo, si trasforma in ambito IT nell'esigenza di accelerare le fasi di sviluppo software. Non è solo una questione di tempo, ma anche di domanda, visto che si reputa che, oggigiorno, non ci siano abbastanza sviluppatori per soddisfare le richieste del mercato. Il pattern low-code rappresenta una delle possibili soluzioni per permettere all'azienda di adattarsi e prosperare tra queste problematiche. Con una piattaforma low-code, le aziende possono realizzare soluzioni sfruttando delle semplificazioni, che permettono di risparmiare tempo, aumentare la produttività, ridurre i costi e raggiungere maggiore flessibilità [4]. È possibile interagire tramite la piattaforma ad una rappresentazione del codice più visiva, interagire con elementi grafici, eseguire operazioni "drag and

drop", l'utilizzo di modelli plug-in e widget riusabili e così via. La piattaforma si occuperà di tradurre la rappresentazione visiva, espressa in forma di low-code, in codice strutturato eseguibile.

Spesso le piattaforme low-code sono realizzate in ambiente Cloud, e così anche i suoi strumenti visivi, ciò garantisce che l'applicazione sia immediatamente utilizzabile e portabile.

Più persone possono contribuire all'implementazione di una soluzione per ridurre i tempi ed aumentare la produttività aziendale, poiché un approccio low-code non richiede particolare esperienza, chiunque ha i mezzi per sviluppare un'app in modo rapido ed efficace. [5]

Le semplificazioni permettono alle aziende di trovare più personale per aumentare la produttività, e agli sviluppatori di velocizzare alcune fasi di sviluppo e concentrarsi sugli aspetti più complessi. Non mancano le problematiche, il team IT dell'azienda è comunque coinvolto nello sviluppo, guidando gli sviluppatori meno esperti nella scrittura del codice nel pattern low-code. È inoltre molto difficile verificare tutte le soluzioni create, il ché espone a rischi di sicurezza, che possono essere superati passando alle piattaforme Cloud, che permettono di definire le policy di sicurezza dello sviluppo low-code.

Infine, uno studio condotto da una azienda di ricerca, chiamata Gartner, ha raccolto dati che testimoniano una crescita del fatturato tra il 2022 e il 2023 con l'utilizzo del pattern low-cod, pari al 19%, per un aumento totale del fatturato che sfiora i 30 miliardi di dollari. Per questo motivo, Gartner stima, che l'aumento degli investimenti delle aziende per lo sviluppo di questo approccio proseguirà sino al 2026. [6]

2.1.2 PowerApps

Power Apps è una piattaforma low code per la creazione rapida di applicazioni personalizzate, che offre un insieme di servizi, connettori e applicazioni per lo sviluppo. Le applicazioni create possono accedere ai dati contenuti nella piattaforma sottostante chiamata Dataverse, o accedere a dati nel Cloud e locali. Alcune sorgenti dati possono essere:

- Dataverse
- SharePoint
- Dynamics 365
- SQL Server and Azure SQL
- Office 365

Non è necessario sceglierne una, poiché Power Apps supporta connessioni multiple ai dati, permettendo di concentrare i dati provenienti da più fonti su una singola applicazione.

Power Apps permette di creare applicazioni web e mobile altamente configurabili per soddisfare le esigenze aziendali, con lo scopo di trasformare le operazioni aziendali manuali in processi automatizzati, scrivendo poco codice [7]. Tali applicazioni sono, inoltre, "responsive", cioè si adattano allo strumento usato per visualizzarle [1]. Power Apps permette, principalmente, la creazione di:

- canvas app
- model-driven app

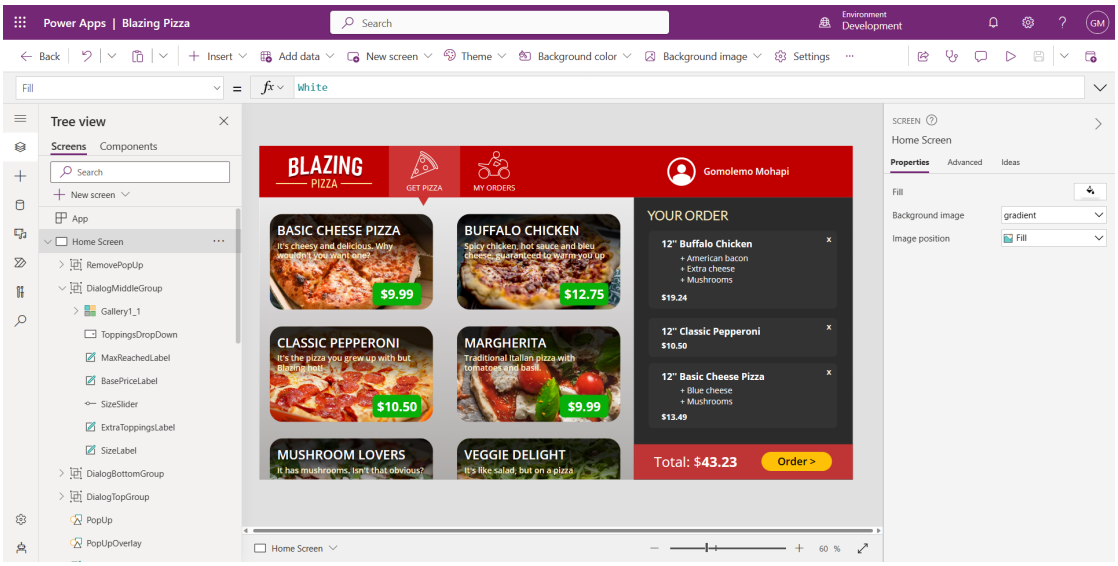

Figura 2.4: Esempio interfaccia di una Canvas App

Le canvas app offrono il maggiore grado di personalizzazione dell'interfaccia utente. La creazione dell'applicazione ha inizio da un foglio bianco, su cui possono essere inseriti una serie di elementi come bottoni, grafici, righe di testo, immagini, lasciando libero sfogo alla creatività. Le canvas app forniscono integrazione con diverse fonti di dati, decidendo come queste debbano apparire all'utente. L'uso di un linguaggio basato su formule permette di definire i comportamenti, di eseguire calcoli e di implementare le "business rules".

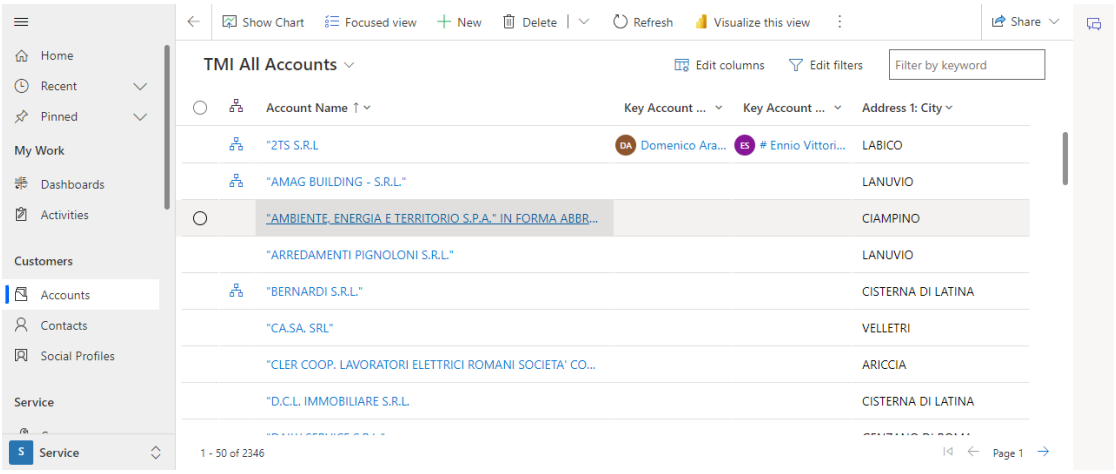

Figura 2.5: Esempio interfaccia di una Model-driven App

Le model-driven app richiedono la presenza di un Dataverse e sono strettamente legate al loro modello dati. Hanno il vantaggio di generare, automaticamente, l'interfaccia utente in base al data model. Infatti, per ogni entità avremo la creazione automatica di form, viste e dashboard, che vengono usate per popolare l'interfaccia. Tali elementi possono poi essere personalizzati, per esempio decidendo quali sono i campi più importati da mostrare, personalizzando i campi nei form e così via, ma, in generale, il livello di personalizzazione è limitato rispetto alle canvas app. A differenza di queste ultime, presentano un processo di validazione built-in dei dati. Nelle model-driven, il focus è il modello dati, quindi avremo una interfaccia utente ricca di dashboard per ogni entità e che contiene la sua rappresentazione tabellare. Vengono spesso realizzate per scenari in cui la logica di business è complicata e richiede modelli dati complessi. In questi casi la creazione automatica dell'interfaccia fornisce un vantaggio indiscutibile. Spesso vengono associate con Dynamics365, poiché possono essere utilizzate per creare applicazione CRM [8].

2.1.3 PowerAutomate

Power Automate è uno strumento della Power Platform, che permette di creare flussi di azioni per automatizzare quei processi aziendali ripetitivi. Per esempio, potrebbe essere creato un flusso, che a partire dalla ricezione di una mail, scateni la creazione di una risposta automatica per segnalare la presa in consegna della richiesta. Usare Power Automate permette di risparmiare tempo e aumentare la produttività dell'operatore aziendale. Esistono tre tipi di flow:

- automatici, questi flow vengono triggerati da un evento, per esempio la modifica di un campo. Sono i più usati;
- istantanei, sono flow che vengono eseguiti automaticamente al click di un pulsante;
- schedulati, sono flow che vengono eseguiti seguendo uno specifico programma temporale. Possiamo per esempio avere flow eseguiti giornalmente, mensilmente e così via.

Ogni flow è composto da due elementi:

- trigger, che determina quando il flusso deve essere eseguito. Per esempio, un trigger configurato alla ricezione di una mail;
- azioni, che determinano cosa il flusso deve fare. Per esempio, memorizzare il contenuto della mail in un file excel [1].

2.1.4 PowerBI

Power BI fornisce i servizi per effettuare analisi aziendali approfondite. Si compone di servizi software, applicazioni e connettori, che collaborano per trasformare collezioni di dati in elementi significativi. Collezionare dati non è abbastanza per avere informazioni utili, bisogna avere buoni dati, cioè, bisogna essere in grado di interpretarli e raccontarli. Per questo, power BI mostra i risultati dell'analisi attraverso report e dashboard, che contengono tutte le informazioni necessarie per prendere una decisione, contengono i dati più significativi, permettono di mostrare le stesse viste/dati a tutto il personale. Power BI prevede policy di sicurezza integrate in modo che il focus aziendale sia la visualizzazione dei dati e non la loro gestione.

2.1.5 Copilot Studio (Power Virtual Agents)

Oggigiorno, le aziende vogliono offrire ai clienti un supporto self-service sempre migliore e veloce, ciò è, per esempio, possibile con la creazione di chatbots. Un chatbot è un componente che fa uso di un modello AI per simulare una conversazione con il cliente. Durante tale conversazione tenta di fornire risposte ripetitive basandosi su parole chiave nella richiesta e tenendo conto delle richieste più comuni. Power Virtual Agent è lo strumento proposto da Microsoft per la creazione di chatbots, che è stato, recentemente, integrato all'interno del nuovo strumento Microsoft Copilot Studio, che ne integra le caratteristiche e abilità. Copilot Studio permette, tramite una interfaccia grafica low-code, la creazione semplice e rapida di copiloti che fanno uso dell'intelligenza artificiale. Tramite il copilota, gli utenti possono,

ad esempio, chiedere l'esecuzione di un task, che viene eseguito triggerando un flusso Power Automate. L'ambiente Copilot Studio è accessibile, con quasi le stesse funzionalità, sia tramite web app, che tramite applicazione integrata su Teams. Fa uso di un modello di comprensione del linguaggio per analizzare l'input dell'utente e selezionare il giusto argomento. Un copilota è composto da più argomenti, che corrispondono a cosa l'utente può chiedere. Nel caso in cui la richiesta dell'utente non soddisfi nessuno dei possibili argomenti, viene richiesto di formulare la domanda fino ad un massimo di due volte, dopodiché, viene avviata una conversazione con un operatore. L'opzione "Migliora conversazioni" può mitigare questa problematica. Essa sfrutta le capacità dei servizi Azure OpenAI per trovare e analizzare le informazioni contenute in uno specifico URL, tentando di fornire una risposta consona. L'uso di Copilot Studio:

- elimina la necessità di avere esperti di intelligenza artificiale nel team, poiché non richiede la necessità di effettuare un training, ma è sufficiente fornire alcuni esempi di frasi che dovrebbero avviare l'argomento;
- riduce il tempo richiesto per la creazione di un chatbot e permette a chiunque di realizzarne uno. È un servizio SaaS, che fornisce una interfaccia grafica semplice per la configurazione e la creazioni di chatbot, che sono facilmente incorporabile nella propria soluzione;
- permette di testare facilmente il prodotto senza doverlo distribuire. È, infatti, presente il tasto "Tieni traccia", con il quale è possibile seguire l'esecuzione della conversazione, evidenziando i nodi attraversati.

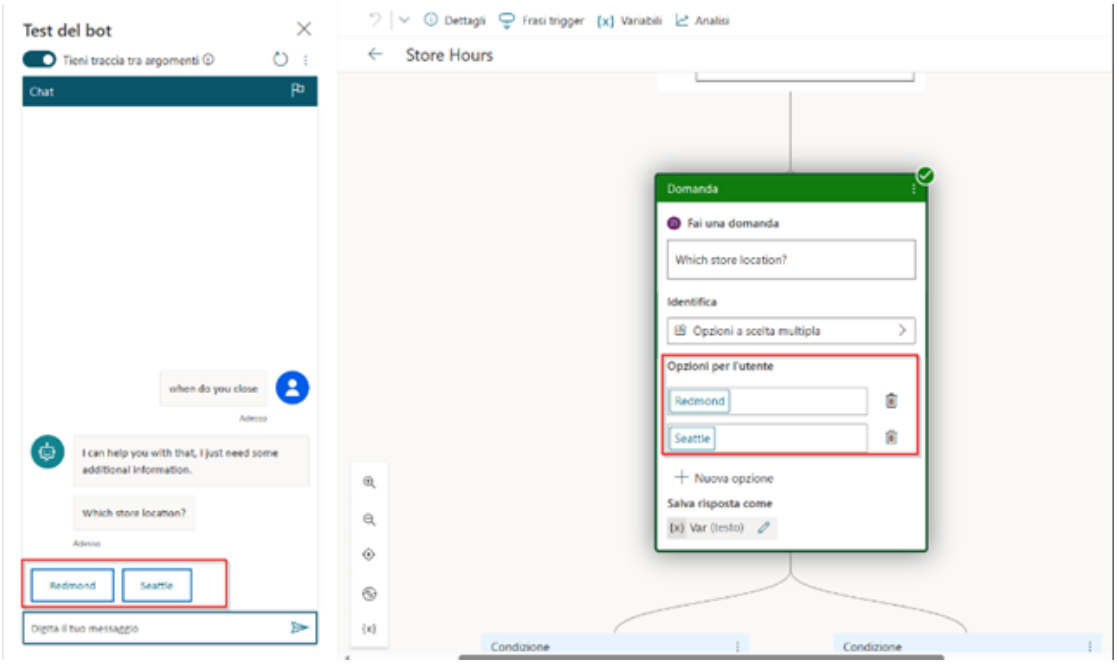

Figura 2.6: Esempio operazione di testing

Un copilota simula la conversazione naturale, grazie alla definizione di argomenti. Un argomento rappresenta un possibile percorso logico definito dal developer. Ogni argomento ha un possibile trigger, vediamone alcuni:

- Frasi: l'argomento viene avviato quando il copilota riceve un messaggio che l'intelligenza artificiale associa al flusso logico. Per far ciò, il developer fornisce al modello AI da 5 a 10 frasi, che devono triggerare l'esecuzione dell'argomento corrente. Il modello AI usa questa piccola base di dati per definire se una frase è semanticamente inerente, ed in questo caso avviare l'argomento;
- Messaggio ricevuto: l'argomento con questo trigger viene avviato ogni qualvolta l'utente invia un messaggio;
- Reindirizza: l'"argomento 2" viene triggerato quando un "argomento 1" reindirizza all'"argomento 2";
- Inattività: l'argomento viene avviato dopo un certo periodo di inattività.

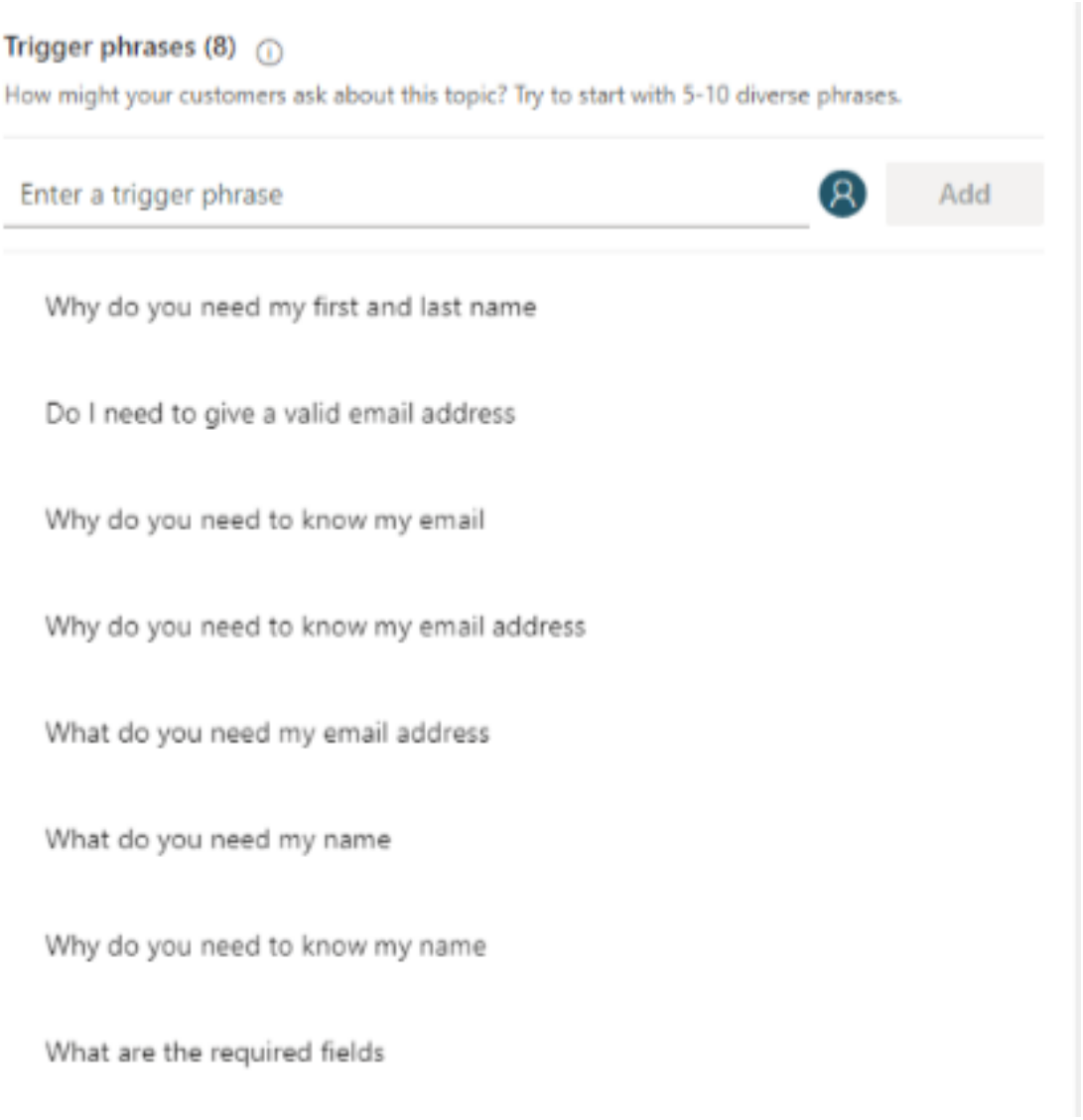

Figura 2.7: Esempio inserimento frasi trigger

È possibile creare condizioni di esecuzioni complesse tramite il nodo "Condizione". Esso permette di condizionare il codice, per esempio si potrebbe dire che l'attivazione avviene solo se il canale utilizzato è Microsoft Teams.

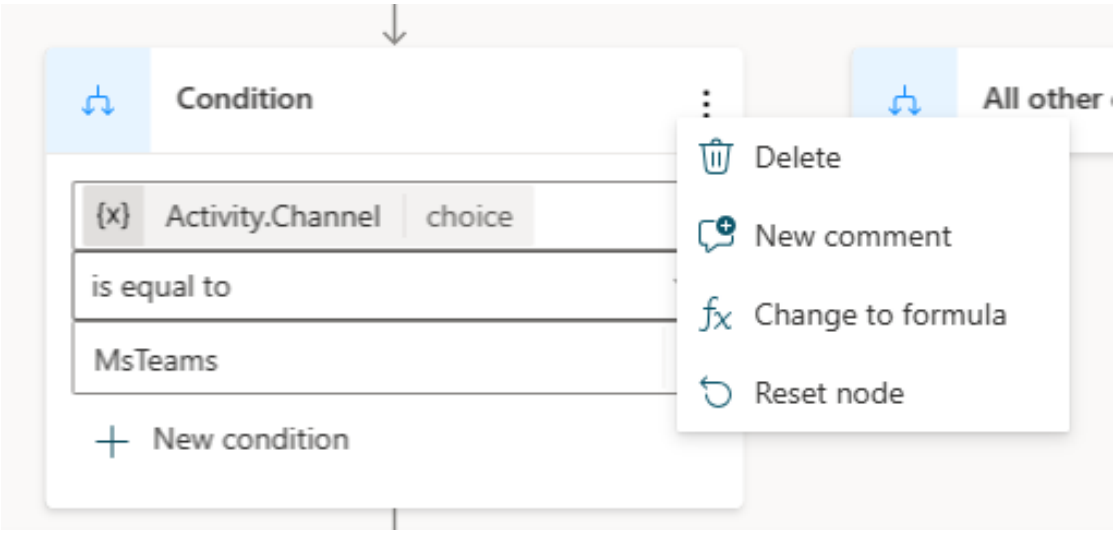

Figura 2.8: Esempio condizione

È possibile definire una priorità nei trigger, in quanto potremmo avere situazioni in cui un input ne scatena molti. L'ordine di esecuzione tiene conto del tipo di trigger, e prevede la seguente successione:

1. Impegno ricevuto

2. Messaggio/Evento/Aggiornamento della conversazione/Richiamo ricevuto

3. Frasi

Nel caso di trigger multipli dello stesso tipo è possibile assegnare esplicitamente l'ordine di esecuzione, definendo un valore numerico. Gli argomenti possono essere di due tipi, di sistema o personalizzati.

Gli argomenti di sistema definiscono i comportamenti essenziali. Per esempio, l'inizio e la fine della conversazione, ma anche i casi di errore e di escalation. Non è né possibile creare argomenti di sistema né eliminarli, ma è possibile modificarli.

Gli argomenti personalizzati costituiscono il fulcro della logica del copilota. Definiscono le azioni che il copilota deve intraprendere, e sono, quindi, creabili, modificabili ed eliminabili.

A seguito di un trigger abbiamo il cosiddetto "Percorso di conversazione", che è un percorso costituito da diversi nodi, che possono essere condizioni, domande all'utente, nodi per la gestione delle variabili, nodi per gestire il passaggio ad altri argomenti, nodi per richieste http e così via. Ogni nodo è quindi una azione effettuata dal copilota per avvicinarsi alla risposta da fornire all'utente [9].

2.1.6 PowerPages

Power Pages è una piattaforma per la creazione, l'hosting e la gestione di siti web aziendali rivolti all'esterno, compatibili con una larga serie di browser e dispositivi. Presenta un hub dell'apprendimento per la formazione contenente la documentazione, vari tutorial e video approfondimento. L'hub prevede che il riquadro di apprendimento non si sovrapponga al livello in cui avviene la creazione del canvas. Questo permette di evitare la perdita del focus durante la creazione. Il sito web creato accede ai dati archiviati nel Dataverse e ciò permette di avere siti web che sia mostrino delle informazioni sia permettano accedere ai dati dell'utente e modificarli se necessario. Prevede una vasta gamma di siti web predefiniti con un aspetto professionale e facili da usare, che una volta selezionati sono altamente modificabili per soddisfare le proprie esigenze. Power Pages utilizza Bootstrap, ciò facilità la creazione di siti web reattivi, cioè adattabili alla dimensione della finestra, così non è necessario creare varie versioni del sito per più categorie di dispositivi. È, comunque, possibile creare interfacce più spettacolari, accedendo direttamente al codice della pagina web.

2.1.7 Dataverse

Le aziende necessitano di memorizzare informazioni rilevanti in un unico luogo, per avere la possibilità di prosperare e crescere. I dati archiviati devono poter essere analizzati e rappresentati per esprimere il loro valore. Possono provenire da diverse fonti, che siano dispositivi o applicazioni. A questo scopo, Microsoft offre una tecnologia facile da usare, gestire, sicura, scalabile e accessibile a livello globale: il Dataverse. Questo è un sistema di archiviazione dati, basato su Cloud Azure, che da garanzie alle aziende che lo scelgono, in quanto fornisce scalabilità, conformità, sicurezza, e accessibilità da qualsiasi parte del mondo. Il Dataverse è, quindi, una soluzione disponibile in tutto il mondo, ma anche globalmente distribuita. Quest'ultima caratteristica è necessaria al fine di favorire la conformità con le policy di privacy sui dati, definite a livello nazionale. I dati vengono collezionati in tabelle, costituite da colonne, ognuna delle quali memorizza uno specifico tipo di dato. Ogni istanza di un Dataverse contiene una serie di tabelle, dette "standard", che rappresentano i dati utili in ogni soluzione, permettendo di ridurre il tempo per la configurazione del data model. Partendo da queste tabelle, è possibile aggiungere campi e/o definire nuove tabelle da zero, dette "custom", che richiedono la definizione dei campi utili. Ogni tabella alla sua creazione contiene un set standard di colonne per la sua gestione. Esistono, infine, un ultimo tipo di tabelle, dette "gestite", che non sono personalizzabili e sono tabelle importate nell'ambiente da una soluzione gestita. Una volta che il data model è stato definito, è possibile connetterlo alle varie applicazioni generate con Power Apps, ai flussi di Power Automate e così via. Sia i metadati che i dati archiviati nel Dataverse sono

memorizzati nel Cloud e presentano, per accedervi, meccanismi di sicurezza basati sui ruoli, in modo che l'accesso ai contenuti vari in base al ruolo nell'azienda. Il Dataverse è più di una semplice collezione di tabelle, poiché integra la sicurezza, la logica, i dati e ne permette l'accentramento. Nelle figura 2.9 vengono mostrate le

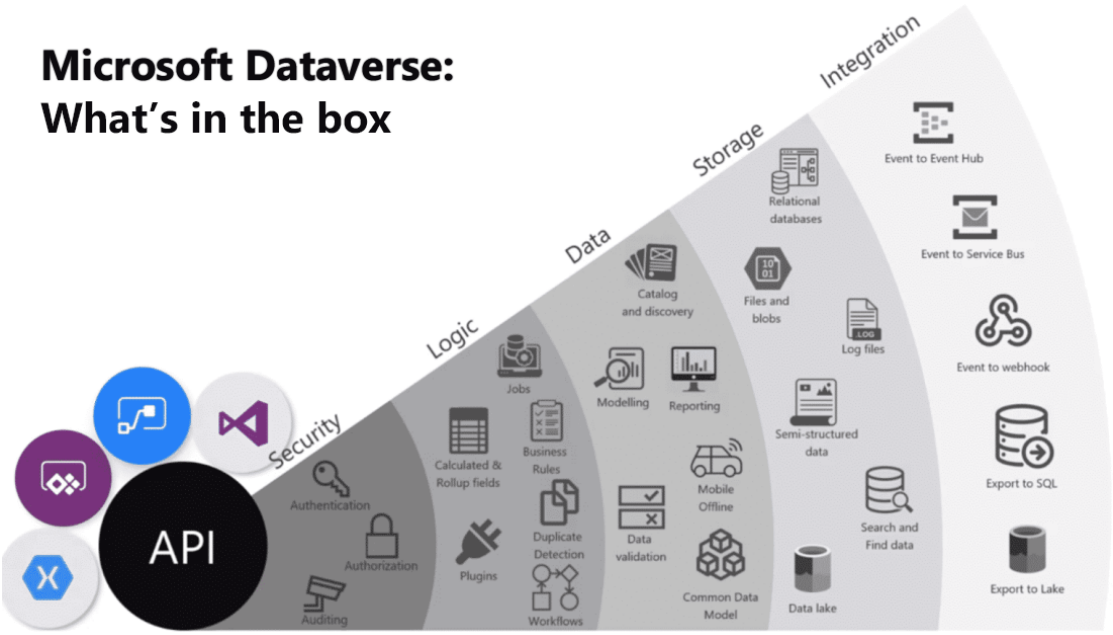

Figura 2.9: Serivizi offerti dal Dataverse

caratteristiche che fanno del Dataverse uno strumento che non memorizza solo i dati.

- Sicurezza: Il Dataverse gestisce l'autenticazione con Microsoft Entra ID per creare più livelli di visualizzazione dei dati, seguendo il principio della minima visibilità, cioè, mostrare solo i dati che sono necessari per svolgere il task. Permette di definire, quindi, vari livelli di autorizzazione, a livello di riga o colonna, fornendo capacità di controllo molto avanzate sui dati.

- Logica: le logiche di business sono facilmente integrabili nel Dataverse, e sono indipendenti da come l'utente interagisce coi dati. Tali logiche possono prevedere il rilevamento dei duplicati, la verifica delle regole di business o altro.

- Dati: i dati sono il fulcro del componente. Il Dataverse permette di modellare i dati seguendo le esigenze aziendali, di eseguire ricerche sui dati e di convalidarli.

- Archiviazione: Il Dataverse è una tecnologia basata sul Cloud. Il vantaggio è che l'azienda non deve avere più preoccupazioni sulla dimensione dei dati o su dove si trovino.
- Integrazione: è una delle caratteristiche principali del Dataverse, realizzata mediante l'uso di API, di webhook, di eventi ed esportazioni, che assicurano flessibilità al modello.

Il Dataverse permette il salvataggio nel Cloud per una moltitudine di dati, la cui quantità dipende dal tipo di licenza. Il Dataverse può contenere milioni di elementi con la possibilità di estendere la dimensione di ogni sua istanza. Per rendere la soluzione efficiente, è necessario individuare i concetti chiave, e creare per ognuno di essi una tabella. L'utilizzo di una sola tabella contenente tutti i dati sarebbe inefficiente, e renderebbe difficile la comprensione. È possibile definire relazioni tra tabelle . Le relazioni più comuni sono le "uno a molti" e le "molte a molti". Tramite Power Apps è possibile creare, modificare ed eliminare le tabelle, nonché visualizzarne il contenuto. La forza del Dataverse sta nella interoperabilità dovuta dalle centinaia di connettori, che permettono di alimentare l'archivio dati e rendere accessibili i dati alle varie componenti del mondo Microsoft, senza dover scrivere alcun codice. Quando si parla di dati è importate parlare di back-up. È possibile eseguire due tipi di back-up:

- automatici o di sistema, che vengono eseguiti automaticamente sfruttando i database SQL Azure;
- manuali, che vengono eseguiti manualmente, solitamente prima di apportare una grossa modifica [10].

In conclusione, il Dataverse è la tecnologia su cui si basano la maggior parte dei prodotti Microsoft, ed, in particolare per la nostra trattazione, Dynamics365.

2.1.8 Connettori

I connettori sono componenti essenziali all'interno del paradigma Microsoft. Una soluzione, spesso, richiede che più servizi comunichino affinché si realizzi un determinato task. L'uso dei connettori permette che questa interazione avvenga senza problemi per lo sviluppatore. Essi rappresentano un ponte tra l'origine dei dati ed il flusso che li richiede. Power platform prevede circa 900 connettori. Si ha il connettore Office 365 Outlook, che consente l'esecuzione di azioni predefinite sulla casella postale, fornendo un numero limitato di dettagli. Tra queste azioni, ad esempio, è possibile avere il download automatico degli allegati, l'invio di messaggi, etc. I connettori vengono utilizzati per interfacciarsi ad origini dati ma anche per connettersi ai servizi Azure di intelligenza artificiale.

I possibili tipi di dati a cui i connettori possono connettersi sono: origine dati tabulare e dati basati su funzioni.

Origine dati tabulare

Queste origini dati restituiscono le informazioni in forma tabellare. Tramite Power Apps è direttamente possibile visualizzare tali dati grazie all'uso di moduli e raccolte. Se, inoltre, l'origine dei dati lo permette, è direttamente possibile creare, modificare, eliminare dati . Microsoft Dataverse, SQL Server e Sharepoint, sono esempi di origini di dati tabellari.

Dati basati su funzioni

Sono un tipo di origine dati che utilizza le funzioni per relazionarsi coi dati. Una funzione può ritornare dati tabellari, ma, anche, eseguire azioni.

I connettori vengono classificati in standard e premium. Questi ultimi richiedono licenze aggiuntive. Definita l'origine dei dati e scelto il connettore, è possibile eseguire due tipologie di operazioni basate sull'origine dei dati: trigger e azioni.

Un trigger può essere scatenato a seguito di un cambiamento sui dati.

Un azione può provocare l'inserimento di nuovi dati nell'origine dei dati. Il raggiungimento di un valore potrebbe triggerare un'azione che richiede un ordine per un prodotto, che si tramuta in una riga contenente le informazioni sull'ordine [1].

2.2 Servizi Azure

Azure è una piattaforma Cloud proprietaria che fornisce una ampia gamma di servizi, che vanno dall'archiviazione sino ai servizi che sfruttano l'intelligenza artificiale. L'obiettivo di tale piattaforma è fornire gli strumenti per la creazione di applicazioni complesse e moderne, senza, però, aumentare la complessità per il developer, che può sfruttare tutta la gamma di servizi della piattaforma. I servizi Azure permettono, inoltre, di creare soluzioni scalabili, affidabili e gestibili. Alcuni di questi servizi sono:

- hosting di applicazioni. Azure permette di ospitare l'intera applicazione sui propri server. Offre varie tipologie di hosting, che vanno da servizi completamente gestiti a contenitori e macchine virtuali;

- serivizi di intelligenza artificiale. Il servizio Azure OpenAI permette l'utilizzo dei modelli AI, quali i modelli GPT-4, GPT-4 Turbo con Vision, GPT-3.5-Turbo;

- servizi basati sui contenitori. Azure offre una serie di servizi basati sui container. L'uso di container può semplificare la distribuzione e la gestione di applicazioni;
- servizi Voce di Azure AI. Servizi per trasformare la voce in testo.

Anche le applicazioni pre-esistenti possono utilizzare i servizi Azure per estendere il loro comportamento. Ad esempio, un'applicazione potrebbe utilizzare i servizi BLOB Azure per archiviare i propri dati nel Cloud, oppure, potrebbe utilizzare i servizi di "speech recognition" per sfruttare la voce come input. In generale, la fruizione dei servizi è disponibile per qualsiasi applicazione, che sia pre-esistente o in lavorazione. Azure offre varie metotologie per gestire le risorse: il portale Azure, linea di comando, usare strumenti IaC(Infrastructure as Code).

I modi per connettersi ai servizi Azure sono due: Azure SDK e API Rest di Azure.

Azure SDK

Azure SDK è la soluzione preferibile, se utilizzabile, che porta una serie di vantaggi:

- L'accesso ai servizi avviene come se si utilizzasse una semplice libreria. Si crea un oggetto di quel sevizio, i cui metodi permettono la comunicazione con la risorsa;
- Il processo di autenticazione viene integrato nella creazione dell'oggetto
- Semplificazione della programmazione. A basso livello SDK richiama le API Rest, definendo anche la logica di ripetizione per eseguire eventuali successivi tentativi.

Azure SDK è utilizzabile solo per un ristretto set di linguaggi di programmazione, quali .NET, Java, JavaScript, Python e Go.

```
13 from azure.identity import DefaultAzureCredential
14 from azure.storage.blob import BlobServiceClient
15 from azure.eventhub import EventHubProducerClient, EventData
16
17 path = Path(__file__)
18
19 cred = DefaultAzureCredential(exclude_shared_token_cache_credential=True)
20
21 blob_service_client = BlobServiceClient(
22     account_url=os.environ["BLOB_SERVICE_ENDPOINT"], credential=cred
23 )
24
25 invoices_container = blob_service_client.get_container_client(container="invoices")
26 records_container = blob_service_client.get_container_client(container="records")
```

Figura 2.10: Esempio utilizzo SDK

In genere, i servizi Azure vengono raggruppati in "resource group", cioè contenitori per i servizi di una soluzione. La suddivisione in gruppi è del tutto arbitraria. Si potrebbero raggruppare tutti i servizi in un unico "resource group", o si potrebbero raccogliere solo alcuni dei servizi, in base alla logica della organizzazione. L'uso di gruppi di risorse permette di gestire i servizi in modo più veloce, quindi utilizzarli, aggiornarli, eliminarli, tutto in un singolo step.

API Rest di Azure

API Rest di Azure è l'alternativa per soluzioni che utilizzano linguaggi di programmazioni che non siano quelli evidenziati in precedenza. Sono servizi endpoints che permettono, tramite operazioni HTTP, di creare, modificare ed eliminare risorse.

2.2.1 App Service

In generale, quando si vuole ospitare una applicazione su un server, abbiamo due alternative: "On-premise Hosting" e "Hosting on a SaaS". Con "On-premise Hosting" ci si riferisce ad un'applicazione ospitata su server di proprietà o affittati dall'azienda, su cui si ha il totale controllo, e su cui è necessaria una gestione. Col Cloud computing, il paradigma "On-premise Hosting" è sempre meno usato, poiché soluzioni come i SaaS permettono all'azienda di focalizzarsi sul business, lasciando la gestione ad altri.

L'Azure App Service è una piattaforma SaaS, basata su HTTP, che permette l'hosting di applicazioni web,usando un qualsiasi linguaggio di programmazione tra i seguenti: .NET, .NET Core, Java, Node.js, PHP, and Python. Prevede un costo legato al consumo e che dipende dalla risorse allocate, definite nel App Service Plan. App Service fornisce:

- meccanismi di sicurezza integrati;
- meccanismi di load balancing;
- scalabilità automatica;
- possibilità di sviluppo continuo del servizio da parte di Azure;
- gestione automatica;
- possibilità di "dockerizzare" l'applicazione in contenitori Windows o Linux.

Alla creazione di un App Service è necessario impostare o creare un App Service Plan, che permette di definire le risorse richieste in base al piano scelto. In particolare vengono definiti:

- OS (Windows o Linux);
- regione;
- numero di istanze di una virtual machine;
- dimensione delle precedenti istanze;
- tariffa, che determina le caratteristiche del App Service ed il costo

Ad esempio, la tariffa Free, che è quella usata in questo progetto, prevede che l'applicazione sia eseguita nella stessa virtual machine usata da altre applicazioni, che possono essere di altri utenti. Le varie app condividono le risorse, che non possono essere ri-scalate. Quindi, si ha che ogni app riceve un certo numero di minuti di esecuzione nella virtual machine condivisa. Spesso viene usata nelle fasi iniziali per testing. In generale, è molto semplice modificare le risorse affidate ad un applicazione o a un set di applicazioni, cambiando il piano tariffario nel corrispettivo App Service Plan. É possibile associare più App Service allo stesso App Service Plan quando le risorse, che verranno condivise tra le due, sono abbastanza per entrambi.

Piano Gratuito	Core	RAM	Archiviazione	In base al consumo
F1 **Gratuito**	Condivisione (60 minuti CPU/giorno)	1 GB	1,00 GB	$0

Figura 2.11: Caratteristiche piano tariffario Free

Piano di servizio Premium v3	Core	RAM	Archiviazione	In base al consumo	Piano di risparmio di 1 anno *	Piano di risparmio di 3 anni*	Riservata - 1 anno	Riservato - 3 anni
P0v3	1	4 GB	250 GB	$0,210/ora	$0,183/ora ~13% di risparmio	$0,162/ora ~23% di risparmio	$0,153/ora ~27% di risparmio	$0,120/ora ~43% di risparmio
P1v3	2	8 GB	250 GB	$0,33/ora	$0,288/ora ~13% di risparmio	$0,254/ora ~23% di risparmio	$0,248/ora ~25% di risparmio	$0,199/ora ~40% di risparmio

Figura 2.12: Caratteristiche piano tariffario a pagamento

2.2.2 OpenAI

Azure OpenAI fornisce l'accesso a vari modelli AI tra cui GPT-4, GPT-3.5 Turbo, GPT-4 Turbo e così via. È possibile utilizzare tali modelli già "trainati" per una moltitudine di operazioni, quali ad esempio, richiedere un riassunto dei contenuti in input, richiedere la creazione di snippet di codice, definire il contenuto di una immagine e tanto altro. L'accesso al servizio può avvenire tramite API Rest, Python SDK o l'interfaccia web di OpenAI Studio. Microsoft collabora attivamente con OpenAI per accedere ai modelli di AI direttamente da Azure, invece che dalle API pubbliche, fornendo, in aggiunta ai servizi OpenAI, le proprietà di sicurezza Microsoft. I vantaggi nell'utilizzare i servizi Azure OpenAI, invece che le API pubbliche, sono molteplici. Microsoft fornisce l'accesso agli stessi modelli e versioni, accessibili tramite API pubbliche, e in aggiunta offre:

- privacy: i dati inviati attraverso le API pubbliche, sono pubblici e possono essere usati per migliorare il modello. Quando si usa il servizio Azure OpenAI, i dati rimangono confidenziali in quanto il modello AI è integrato nella architettura Azure e nessun dato viene inviato verso l'esterno;
- sicurezza: Microsoft offre la sua fidata e robusta architettura. Gli sviluppatori Azure sono molto attenti alle questioni legate alla sicurezza; difatti Azure OpenAI permette l'uso della crittografia, utilizzo di reti private ed inoltre, non meno importante, un sistema di accesso globale;
- AI responsabile. Microsoft definisce delle linee guida in modo da identificare, mitigare i potenziali problemi derivanti dalla AI generativa. Per esempio, è previsto un meccanismo per evitare che l'output del modello sia potenzialmente dannoso;

Concetti Chiave

Gli utenti usufruiscono dei servizi offerti dal modello GPT tramite **richieste** che vengono fornite all'**endpoint di completamento**, che è un componente che

fornisce un'interfaccia col modello. Sebbene i modelli offrano una moltitudine di nuove possibilità, essi sono molto sensibili a come vengono formulate le richieste. La costruzione di richieste non è banale ed è più un'arte che una scienza, poiché richiede più che altro esperienza ed intuizioni [11]. Alcuni concetti da tenere in mente per generare una richiesta più chiara, sono:

- essere più specifici possibili, lasciando poco spazio all'interpretazione;
- usare analogie;
- a volte è utile ripetersi, cioè fornire la stessa istruzione prima e dopo la richiesta principale;
- l'ordine in cui definiamo la richiesta può influire sulla stessa
- definire situazioni di uscita nel caso in cui non si è in grado di rispondere alla richiesta, evitando, così, che il modello inventi informazioni o esegue scelte forzate.

La richiesta viene scomposta in **token**, che possono essere intere parole o blocchi di caratteri, in realazione alla lunghezza della parola. Ad esempio la parola "hambuger" crea i token "ham", "bur" e "ger", mentre la parola "ciao" ne genera uno solo. Il numero di token determina la latenza della risposta.

Modello GPT-4-32k

Azure OpenAI fornisce una collezione di modelli AI diversi per funzionalità e per fascia di prezzo. La famiglia GPT-4 accetta sia input testuali che immagini ed in grado di generare testo. In particolare, GPT-4 è ottimizzato per la chat permette, quindi la generazione di risposte in linguaggio naturale, che possono includere la generazione di codice. La versione utilizzata nel progetto è la 0613:

- un numero massimo di token al minuto di 9.000;
- un numero massimo di token per richiesta di 32,768
- un numero massimo di richieste al minuto di 54.
- un training dei dati che risale al settembre 2021;

2.3 Dynamics365

Dynamics365 è una suite di applicazioni model-driven in ambito CRM ed ERP costruite sulla Power Platform di Microsoft. Sebbene l'ambito della tesi si direzioni

verso i CRM, e bene conoscere anche, in breve, cosa sia un sistema ERP. Un sistema ERP è un software che permette all'azienda di organizzare le risorse operative, quindi, gestire il magazzino, la produzione, la contabilità. A differenza di un CRM che è focalizzato esclusivamente sulla vendita e le relazioni coi clienti, un sistema ERP è costituito da più moduli software, scelti in base alle necessità aziendali, ognuno dei quali gestisce una delle precedenti risorse operative enunciate.

Dynamics365 è costituito da un insieme di applicazioni aziendali con lo scopo di facilitare la gestione dell'intero processo aziendale. Le sue applicazioni mirano a facilitare uno specifico ambito (Vendite, Servizio Clienti) e sono facilmente integrabili sia tra loro, che con i sistemi pre-esistenti. Dynamics365 è la soluzione offerta da Microsoft per la gestione delle relazioni con i clienti. La soluzione aiuta a gestire e immagazzinare le informazioni sugli attuali clienti e sui poteziali, mantenendo tali informazioni in un sistema centralizzato. Il servizio clienti può accedere a queste informazioni, quando necessario, in modo semplice usando dashboard personalizzate, in cui è possibile eseguire ricerche di vario tipo. Un'azienda che non usa un CRM potrebbe non sfruttare appieno le potenzialità di crescita e perdere eventuali clienti a causa di un sistema di gestione non efficiente. Fin non troppo tempo fa le aziende memorizzavano le informazioni dei clienti e le opportunità proposte in soluzioni CRM basate su carta. La mancanza di integrazione e automazione portava all'impossibilità di accedere e condividere le informazioni su clienti, rallentando il processi di vendità e riducendo le entrate aziendali. I moderni CRM, come quelli offerti in Dynamics365, permettono di collezionare autonomamente i dati in un ambiente facilmente accessibile da tutti i team di vendita, fornendo, in aggiunta, il supporto alle varie fasi aziendali con automatismi e strumenti di AI che facilitano le azioni dei team.

In altre parole, i moderni strumenti di CRM forniscono ai team di assistenza clienti, di vendita, di marketing, di commercio, di servizio sul campo uno strumento per avere visibilità immediata a tutti i dati cruciali per lo sviluppo, il miglioramento e il mantenimento delle relazioni con i clienti [12]. Le dashboard definite nelle soluzioni CRM di Dynamics365 vengono realizzate attraverso la componente Power Apps, che permette la creazione di model-driven applications. I team possono accedere, attraverso la model-driven application, alle dashboard contenenti le informazioni utili. Le dashboard appaiono come tabelle contenenti dati, provenienti dal Dataverse, su cui è possibile eseguire delle operazioni di ricerca. I team di vendita possono accedere a specifici dati tramite i sistemi di ricerca offerti dall'ambiente. I dati possono essere interrogati attraverso una delle seguenti modalità: ricerca di Dataverse, ricerca rapida e ricerca avanzata.

Ricerca di Dataverse

La ricerca di Dataverse è un tipo di ricerca abilitata di default. Permette di ricercare un dato su più tabelle, fornendo le corrispondenze in un elenco ordinato (Figura 2.13), utilizzando un sistema di ricerca esterno. I risultati possono, anche, essere mostrati in una finestra ricapitolativa, suddivisa per tabelle, in cui è possibile selezionare le corrispondenze definendo dei filtri (Figura 2.14). Il comportamento di ricerca è il seguente: ricerca corrispondenze in qualsiasi colonna di una tabella, che contenga testo nella forma di riga singola, righe multiple, lookups e set di opzioni. Non supporta la ricerca su tipi numerici o date. Questo tipo di ricerca offre i seguenti vantaggi:

- migliori prestazioni di ricerca, grazie all'indicizzazione esterna e all'uso della ricerca Azure;
- trova le corrispondenze con qualsiasi parola inserita nella ricerca in qualsiasi colonna, a differenza di quanto avviene nella "ricerca rapida" che necessita che tutte le parole inserite nel termine di ricerca siano presenti all'interno del singolo campo della colonna.
- Le corrispondenze possono includere forme flessive di una parola, ad esempio con la parola "flusso" vengono restituite, anche, le corrispondenze "flussometro" e "flussimetria".
- Il risultato viene fornito in un elenco ordinato per pertinenza. Cioè se, per esempio, abbiamo due dati da cercare, i dati presenti su due campi, avranno maggiore pertinenza rispetto agli stessi dati posti nello stesso campo a distanza di righe di testo.
- Evidenzia le corrispondenze nell'elenco dei risultati.
- Le corrispondenze sono divise per tabelle.
- Vengono visualizzati, anche, corrispondenze con un carattere errato. Se, per esempio, si cerca "rosa" verranno mostrate anche le corrispondenze con "rossa".

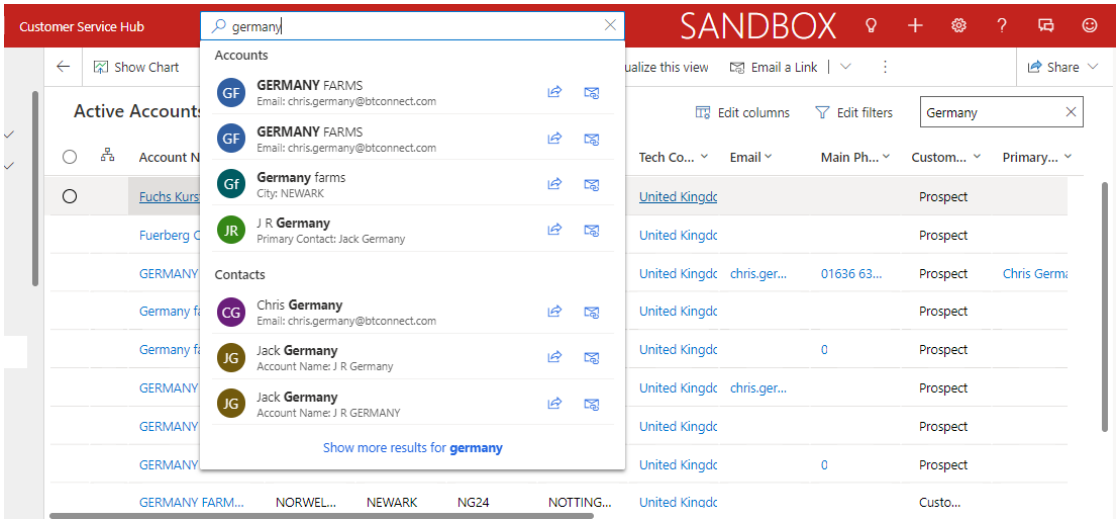

Figura 2.13: Ricerca di Dataverse

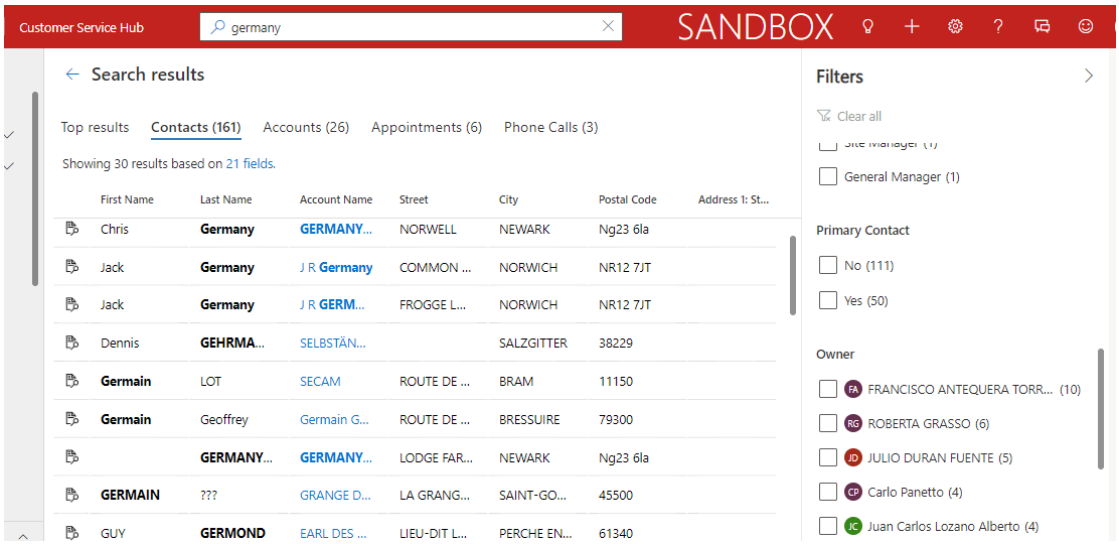

Figura 2.14: Ricerca di Dataverse: riepilogo ricerca

Ricerca rapida

La ricerca rapida è un tipo di ricerca che presenta due sfaccettature: ricerca su griglia e ricerca categorizzata. La prima indica la possibilità di ricerca sulla singola tabella, ed è abilitata di default, la seconda indica la possibilità di ricercare su più tabelle. Al fine di abilitare quest'ultima un amministratore di sistema deve disattivare la ricerca di dataverse. Il risultato della ricerca viene mostrato all'interno

della vista nel caso di ricerca su griglia, mentre nel caso di ricerca categorizzata, il risultato viene raggruppato per tabelle e viene calcolato su un massimo di 10 di loro. Il comportamento di ricerca è il seguente: cerca le corrispondenze di tutte le parole inserite nella barra di ricerca presenti in una colonna di una tabella. La ricerca è effettuata su tutti i campi ricercabili selezionati nelle configurazioni. La ricerca rapida fornisce la possibilità di trovare corrispondenze in una colonna della tabella o su più tabelle. La ricerca su più tabelle è detta "ricerca categorizzata", che viene effettuata su massimo di 10 tabelle, fornendo risultati suddivisi per tabella. Questa opzione di ricerca è disponibile in una vista.

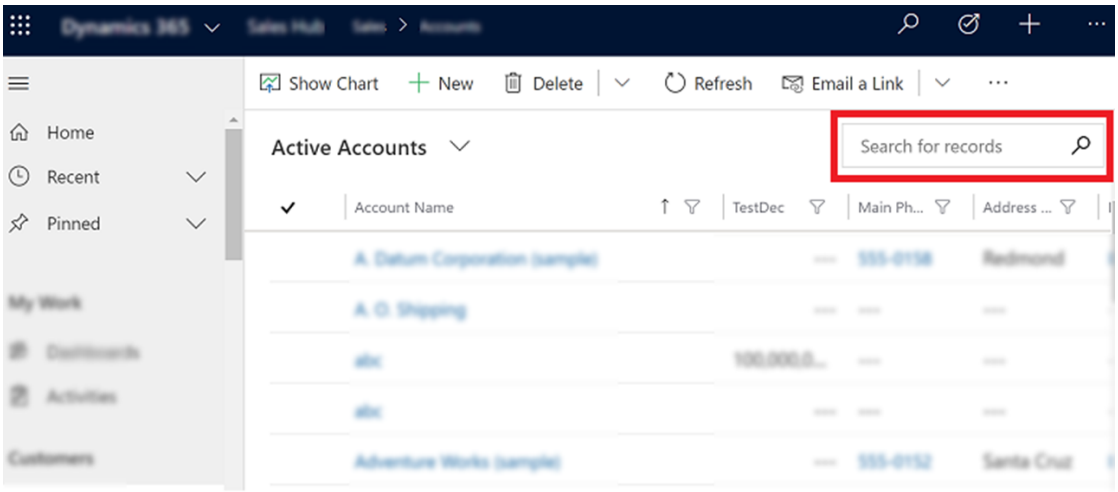

Figura 2.15: Ricerca rapida

Ricerca avanzata

La ricerca avanzata è un tipo di ricerca abilitata di default. Permette di effettuare ricerche accurate su tabelle singole o multiple. È possibile:

- definire precisamente il valore di un certo campo;
- definire condizioni di ordinamento;
- definire condizioni complesse di ricerca;
- definire relazioni tra tabelle;
- usare una GUI intuitiva e veloce;
- cercare su tutti i campi ricercabili;

L'utente usa la GUI fornita dal sistema per effettuare la ricerca. Se questa richiede la visualizzazione di campi che non sono visibili nella vista corrente, l'utente può accede alla sezione "Modifica Colonne", che permette di gestire quali colonne devono essere visualizzate. La struttura di composizione della ricerca ricorda quella prevista dal linguaggio SQL. L'interrogazione, inserita tramite la GUI, viene poi passata ad un costruttore di interrogazioni, che la traduce in formato "fetchXML". Con tale formato è possibile richiedere i dati al Dataverse e verrà approfondito nel prossimo capitolo.

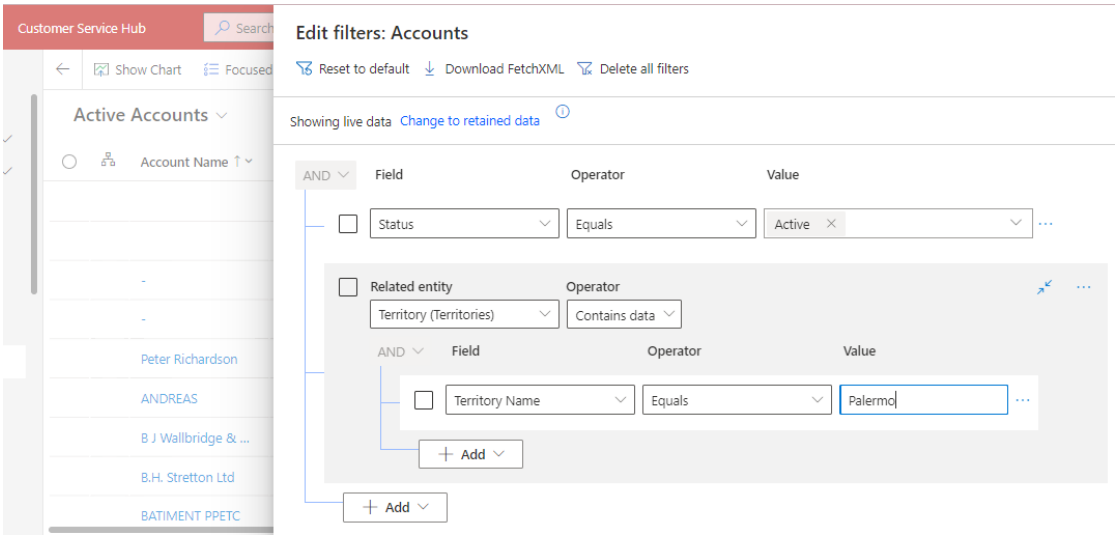

Figura 2.16: Ricerca Avanzata

Capitolo 3

Progettazione

Il capitolo presenta, innanzitutto, il contesto in cui avviene la progettazione, inteso come l'ambiente in cui si vuole fornire valore con la proposta di tesi. Successivamente, definisce quali siano le necessità che la proposta cerca di risolvere ed infine, fornisce un possibile modello logico per realizzare i requisiti emersi dallo studio del contesto.

3.1 Funzionalità richieste e contesto

Nei moderni sistemi CRM, i team di vendita, del servizio clienti ed in generale di tutti i reparti, possono accedere alle informazioni sui clienti, o sui potenziali, attraverso una interfaccia personalizzabile, che mette in risalto le caratteristiche principali necessarie per lo sviluppo, il miglioramento e il mantenimento della clientela. I team potrebbero avere, in aggiunta, la necessità di effettuare ricerche sulla fonte dati per estrarre informazioni chiave per i loro task. Tali indagini sono possibili tramite le operazioni di ricerca, che sono state definite sul finire dello scorso capitolo. Esse sono la ricerca di Dataverse, la ricerca rapida e la ricerca avanzata.

Tra queste, la ricerca avanzata è quella che permette di realizzare interrogazioni puntuali e ben definite, sfruttando una struttura standard per la creazione. La complessità ed il tempo richiesto per la sua creazione sono i principali svantaggi di questo tipo di ricerca, che vengono, però, compensati dal valore che i suoi risultati forniscono.

Appare chiaro, quindi, come la possibilità di rendere più veloce la realizzazione di queste ricerche possa portare ad avere risvolti molto positivi all'interno dei team. Tali vantaggi includono:

- un aumento della agilità aziendale. La possibilità di rendere questo meccanismo più semplice permetterebbe un aumento del numero di operatori in grado di fornire un apporto rilevante al processo;
- un risparmio in termini di tempo nella scrittura delle interrogazioni;
- delle analisi approfondite più semplici da realizzare.

La necessità di rendere le interrogazioni dei team più semplici potrebbe essere soddisfatta dallo sviluppo di un sistema che permette di creare interrogazioni puntigliose sui dati contenuti in tabelle custom o di default per mezzo del linguaggio naturale. La realizzazione di un sistema, che sia capace di fare ciò, è, oggigiorno possibile grazie allo sviluppo, che negli ultimi anni, molte aziende hanno supportato con ingenti quantità di denaro. Microsoft, ad esempio, ha concretizzato le sue ricerche sui modelli AI introducendo, nel Marzo 2023, un nuovo strumento, chiamato Copilot, che permette di rendere più semplici certe azioni, sostanzialmente, definendole in forma di linguaggio naturale.

La tesi è stata eseguita all'interno della Business Unit di Cluster Reply DCX, che offre la creazione di soluzioni personalizzate in ambito Automotive. Tra le soluzioni, spiccano le applicazioni model-driven utilizzate all'interno di Dynamics365. Per il contesto in cui Reply lavora, è stato considerato come modello dati una copia di quello di una azienda leader nel settore delle macchine e dei servizi nel mondo degli operatori agricoli.

Il modello dati proposto nasce dall'esigenza di questa azienda di realizzare una piattaforma CRM su scala globale. In generale, tale modello dati è stato creato affinché venisse utilizzato dalle concessionarie e dai team nazionali di mercato di un'azienda, che si muove nel campo agricolo, per gestire il processo di ricerca, trattativa e vendita con il cliente finale.

3.2 Architettura

Il paragrafo definisce i blocchi logici necessari per la progettazione di un modello che permetta la creazione di ricerche in linguaggio naturale sui dati del CRM.

Durante le numerose riunioni, organizzate precedentemente e durante la fase di progettazione, sono stati definiti gli obiettivi che il progetto dovesse raggiungere. Tali requisiti di progettazione prevedono:

- la necessità di interrogare dati in tabelle custom o standard;
- la necessità di eseguire interrogazioni che richiedono operazioni aggregate;
- la necessità di rendere fruibile il servizio globalmente;

- la necessità di non aggiornare continuamente le informazioni sul modello dati;
- la necessità di verificare la sintassi dell'interrogazione e ripetere la sua creazione se necessaria;
- la necessità di fornire una integrazione sul sistema Teams;
- la possibilità di fornire risposte a domande puntuali, cioè che ritornano un numero limitato di record.

Durante la progettazione sono stati definiti i vari blocchi logici, mostrati in Figura 3.1, per la realizzazione di quanto chiesto. Essi comprendono:

- un blocco che, basandosi su un modello AI, permette di convertire le necessità espresse in forma di linguaggio naturale in un formato comprensibile per la fonte dati;
- un blocco che verifichi la correttezza sintattica dell'interrogazione nel formato scelto e la effettui. Se la correttezza sintattica non è rispettata deve essere previsto un meccanismo ciclico di tentativi;
- un blocco che, basandosi su un modello AI, interpreti il risultato dell'interrogazione fornendo una risposta in linguaggio naturale;
- un blocco che permetta la disponibilità globale del servizio;
- un blocco che permetta di memorizzare le informazioni sul modello dati.

3.2.1 Descrizione Modello

In questo paragrafo si descrive il comportamento atteso in base ai requisiti definiti dal gruppo Cluster Reply. L'obiettivo è permettere all'utente di interrogare i dati del CRM in modo più semplice. La Figura 3.1 rappresenta un possibile modello per realizzare quanto chiesto. L'utente inserisce sotto forma di testo in linguaggio naturale la richiesta con cui vorrebbe ottenere delle informazioni. Un modello AI la riceve, la analizza e, basandosi sulle informazioni che ha precedentemente recuperato sul modello dati, crea una richiesta. L'analisi permette di interpretare la richiesta e trasformarla in un formato, che in questa sede denominiamo "formato di ricerca", che sia compatibile con la fonte dati. Il "formato di ricerca" viene analizzato sintatticamente da un blocco di verifica, che, in caso di errori sintattici, prevede una richiesta di ripetizione dell'interpretazione dell'input dell'utente. In caso di successo, invece, il "formato di ricerca" viene utilizzato per interrogare la fonte dati, la quale restituisce il risultato ad un altro modello AI. Quest'ultimo interpreta questi dati e la richiesta dell'utente per generare una risposta all'utente

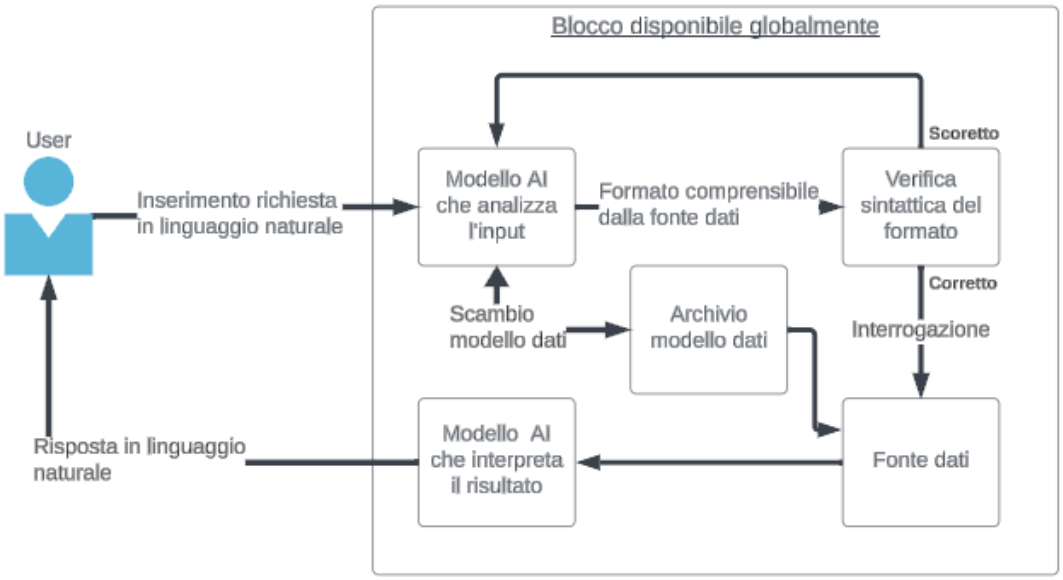

Figura 3.1: Modello progettuale

in linguaggio naturale. A questo punto, l'utente ottiene i dati che, però, poiché non è prevista una forma di verifica semantica, potrebbero essere il risultato di una mal interpretazione del primo modello AI.

3.3 Processo Progettuale

La progettazione ha seguito due fasi principali, la seconda che ha esteso le funzionalità della prima. In particolare, le due fasi sono le seguenti:

1. Una prima fase di progettazione del sistema di interrogazione in linguaggio naturale su più tabelle, ma con la limitazione che la richiesta richieda il coinvolgimento di al più una di esse;

2. Una seconda fase che prevede l'estensione della precedente per la creazione di interrogazioni più complicate che possono richiedere il coinvolgimento di più tabelle e su cui sono possibili operazioni di max, min e count;

Nel prossimo capitolo, approfondiremo le scelte implementative effettuate per ognuna di queste fasi.

Capitolo 4

Implementazione

Il capitolo definisce, innanzitutto, alcune scelte implementative, definendone le motivazioni e i valori. Successivamente, definisce l'architettura dettagliata del progetto e, per ogni suo componente, viene trattata la logica di funzionamento. Laddovè si ritiene opportuno, viene mostrato il codice, per una descrizione più accurata. Inoltre, per ogni componente si evidenzia l'evoluzione lungo le fasi descritte nel finire dello scorso capitolo.

4.1 Langchain

LangChain è un framework open source per la creazione di applicazioni che utilizzano modelli linguistici di grandi dimensioni(LLM). Gli LLM sono dei modelli AI, che vengono pre-addestrati su grandi quantità di dati. Tali dati vengono scelti in base all'ambito in cui il modello AI vuole essere usato. Inizialmente, i modelli AI avevano la necessità di avere un grande e ben etichettato dataset per ogni contesto di utilizzo. Se per questo contesto, il dataset non esisteva, si richiedevano migliaia di ore per collezionare i componenti testuali, le immagini, etc al fine di costruirlo. Successivamente, il modello imparava basandosi sulle informazioni del dataset e, infine, poteva essere applicato al contesto scelto. Il futuro dell'AI spinge verso modelli flessibili e riusabili, in modo che un modello addestrato su certi dati possa essere applicato anche in altri contesti. Tutto ciò ha portato a rimpiazzare i modelli AI specifici per un task con i cosidetti modelli di fondazione. Questi vengono addestrati su un vasto insieme di dati non etichettati, che richiedono un minimo affinamento e possono poi essere utilizzati nei vari contesti. Gli LLM non sono onniscenti, spesso è richiesto che accedano a fonti dati esterne o a API per usufruire di qualche servizio. LangChain permette di semplificare notevolmente le precedenti attività. Offre un approccio basato su moduli, che permette agli sviluppatori di poter confrontare i vari prompt e modelli di fondazione senza dover riscrivere l'intero

codice [13]. In generale, l'uso degli strumenti forniti da LangChain permette di migliorare la personalizzazione, l'accuratezza e la pertinenza delle risposte fornite dal modello AI. LangChain fornisce una serie di astrazioni in python o javascript per permettere anche all'utente meno esperto di interfacciarsi al mondo dell'AI generativa. Molti modelli sono importabili su LangChain e la classe LLM ne fornisce una interfaccia standard. Vediamo alcuni vantaggi dell'utilizzo di langchain:

- possibilità di riutilizzare LLM di contesti specifici senza necessità di riaddestramento;
- metodologie per estrarre informazioni da documenti e, così, incrementare l'esperienza dell'utente;
- semplificazione dello sviluppo dell'intelligenza artificiale;
- strumenti di supporto alla connessione dei modelli con origini dati esterne
- uso gratuito e ampiamente supportato.
- rende l'AI generativa più accessibile agli appassionati

4.1.1 Moduli

I principali moduli presenti in LangChain sono [14]:

- Prompts (4.1.1);
- Models (4.1.1);
- Memory (4.1.1);
- Agents (4.1.1);
- Callbacks (4.1.1);
- Chains (4.1.1);

Prompts

È il modulo che si occupa della gestione dei prompts. I prompt sono le indicazioni che si forniscono al LLM, in modo da fornirgli il contesto in cui è richiesto il suo utilizzo. Un prompt in generale conterrà qualcosa del tipo "Fai questo", "Non fare quello", può contenere una serie di esempi per indicare il formato della risposta, o, ancora, può contenere una domanda a cui rispondere. La classe ChatPromptTemplate viene utilizzata per creare modelli di prompt flessibili per i modelli di chat. Il metodo "from_messages" accetta una lista di messaggi in specifici formati. Nell'esempio

viene fornita una lista di messaggi di chat espressi in forma di tupla, il cui primo elemento indica il tipo, ed il secondo contiene il template. Il template può contenere la definizione di variabili, che ne aumentano la flessibilità. Al momento della chiamata, le variabili vengono sostituite con il loro valore.

```
1  from langchain_core.prompts import ChatPromptTemplate
2
3  template = ChatPromptTemplate.from_messages([
4      ("system", "You are a helpful AI bot. Your name is {name}."),
5      ("human", "Hello, how are you doing?"),
6      ("ai", "I'm doing well, thanks!"),
7      ("human", "{user_input}"),
8  ])
9  messages = template.format_messages(
10     name="Bob",
11     user_input="What is your name?"
12 )
```

Un oggetto ChatPromptTemplate può ricevere in input, oltre al formato di tupla, anche altri, ad esempio, può ricevere una istanza della classe MessagePromptTemplate.

Models

Questo modulo prevede due tipologie di modelli: "Language Models" e "Text Embedding Models".

Per quanto riguarda il primo, si ha una ulteriore divisione:

- I "Language Models" sono il componente principale di LangChain. LangChain fornisce una interfaccia standard per comunicare con vari LLM, utilizzando una API "key". Questa categoria di modelli prendono in input e restituiscono in output del testo;
- I "Chat Models" utilizzano di base gli LLMs, ma prevedono "chat messages" sia come input che come output, il che li rende adatti all'uso in applicazioni che richiedono una conversazione;

I "Text Embedding Models" vengono usati quando si ha la necessità di associare il testo ad una controparte numerica. Tali numeri saranno compresi e processati dai machine learning models. "Text embeddings" definisce una rappresentazione vettoriale di un testo, che può essere utile se si vuole pensare al testo nello spazio vettoriale per eseguire operazioni come la ricerca semantica. Essa permette di trovare pezzi di testo più simili presenti nello spazio vettoriale. I "Text Embedding Models" prendono in input un testo e ritornano in output una lista di numeri con la virgola.

Memory

Il modulo fornisce gli strumenti per mantenere lo stato durante la conversazione, in modo da contestualizzare le future richieste dell'utente. Non è possibile passare l'intera conversazione ai modelli per motivi legati al limite sul numero di tokens. La classe ConversationBufferMemory permette di salvare i messaggi scambiati e successivamente estrarli. La figura sottostante mostra un esempio di utilizzo.

```
1     from langchain.memory import ConversationBufferMemory
2
3     memory = ConversationBufferMemory()
4     # saving conversation
5     memory.save_context({"input": "Describe LSTM"}, {
6                     "output": "LSTM is a type of recurrent neural
    network architecture that is widely used for sequential and
    time series data processing."})
7     # retrieving conversation from a memory
8     memory.load_memory_variables({})
```

Una classe alternativa è la ConversationBufferWindowMemory, che permette di memorizzare una finestra contenente gli ultimi k messaggi.

Agents

A differenza di una Chain, che è definita da una serie di passaggi "hardcoded" nel codice ed eseguiti in un ordine predefinito, un Agente è un modello linguistico che può prendere decisioni sulle azioni da intraprendere. Ad esempio, un agente potrebbe decidere se rispondere ad una domanda, eseguire una ricerca o fare una chiamata ad una API. Sono tre le fasi principali per la definizione di un Agent:

- creazione dell'istanza con l'input dell'utente;
- creazione degli strumenti associati all'agente, che esso può utilizzare
- esecuzione dell'agente, che sfruttando il modello linguistico associato decide quali azioni intraprendere ed in quale ordine

```
1 # Importare le librerie necessarie
2 from langchain import Agent, Memory, Tool
3
4 # Definire l'input dell'utente
5 user_input = "Qual è il tempo oggi?"
6
7 # Creare un'istanza dell'agente
8 agent = Agent(input=user_input)
9
```

```
# Definire gli strumenti che l'agente può utilizzare
tool_weather = Tool(name="weather", function=get_weather)
# get_weather è una funzione definita dall'utente
agent.add_tool(tool_weather)

# Fornire all'agente una memoria iniziale
memory = Memory(data={"location": "Roma"})
agent.set_memory(memory)

# Eseguire l'agente
agent.run()
```

Nell'esempio viene istanziato un Agent con l'input dell'utente, dopodiché si definiscono i suoi strumenti, li si associa all'istanza e ,infine, lo si esegue. L'Agent deciderà quale strumento utilizzare ed in che ordine usarlo per rispondere alla domanda dell'utente.

Callbacks

Il modulo Callbacks fornisce allo sviluppatore la capacità di reagire e certi eventi che possono verificarsi durante l'esecuzione di una chain. In questo modo è possibile il monitoraggio e la registrazione di ciò che avviene nel sistema e la creazione di applicazioni LLM più complesse e personalizzate.

```
    chain = LLMChain(llm=llm, prompt=prompt, callbacks=[handler])
```

Nella figura è mostrato come è possibile associare una callback ad una chain. FileCallbackHandler è esempio di classe che implementa un gestore di callback per scrivere l'output in un file. Questo può essere utile per il logging e il debug.

Chains

Il componente principale in LangChain è la "chain", che permette di combinare più componenti insieme al fine di creare una unica applicazione. Questa possibilità di scelta rende lo strumento versatile, cioè utilizzabile in una grande varietà di contesti. Ad esempio, la catena LLMChain permette di legare il modello AI e quello dei prompt, passandoli come inputs. Al fine di eseguire la catena appena creata viene chiamato il metodo run della classe LLMChain, che si aspetta in input le variabili necessarie al prompt. [15]

4.2 FetchXML

La fetchXML è un formato proprietario di Microsoft, che viene utilizzato all'interno di Dynamics365 per interrogare il Dataverse. Può essere usato per farsi ritornare i dati contenuti ma, anche, per inserire, eliminare informazioni nel Dataverse. Come suggerisce il nome, si basa sul formato XML e permette di eseguire interrogazioni che:

- prevedono la definizione di condizioni più o meno complesse sulle varie entità del modello dati;
- presentano operazioni su aggregati. Ad esempio, è possibile definire il conteggio, la somma, la media, il minimo e il massimo;
- prevedono un ordinamento;
- prevedono il join tra più entità;

Il formato fetchXML viene usato all'interno di Dynamics365 tramite l'operazione di ricerca avanzata, per generare le viste , o, ancora, all'interno dei flussi definiti con Power Automate. Offre strumenti di interrogazioni simili al linguaggio SQL, a differenza del quale non presenta operatori di unione, intersezione e differenza, inoltre, non è possibile creare sottoquery. Viene usato per interrogare i dati utilizzando l'API Web o l'SDK per ".NET". Considerando l'SDK, l'esecuzione di una interrogazione nella forma di fetchXML avviene, dapprima, definendo la fetchXML come una stringa, e ,successivamente, si utilizza l'interfaccia "IOrganizationService", che fornisce i metodi per accedere ai dati e ai metadati del Dataverse. Quando il Dataverse riceve una fetchXML, ne esegue la traduzione in una query SQL e provvede alla sua esecuzione sul database.

Si veda adesso il seguente esempio:

```
1  <fetch mapping='logical'>
2     <entity name='account'>
3        <attribute name='accountid'/>
4        <attribute name='name'/>
5        <link-entity name='systemuser' to='owninguser'>
6           <filter type='and'>
7              <condition attribute='lastname' operator='ne' value='
       Cannon' />
8           </filter>
9        </link-entity>
10    </entity>
11 </fetch>
```

La prima riga contiene l'intestazione, dove si specificano le caratteristiche dell'interrogazione. Ad esempio, si potrebbe avere mapping='logical', che indica al Dataverse che i "name" rappresentano i nomi logici degli attributi/entità. Si potrebbe avere distinct='false', per indicare che si accettano dati uguali, oppure si potrebbe avere aggregate='true' per indicare che l'interrogazione contiene una operazione aggregata. Subito dopo l'intestazione, si trova l'entità, cioè la tabella. Indentati al suo interno, si trovano gli attributi che si richiedono indietro e le entità di collegamento(link-entity). Esse definiscono una operazione di join, verso la tabella definita nella proprietà name, con l'attributo specificato dalla proprietà "to". All'interno della "link-entity" si trovano gli attributi appartenenti alla nuova entità che vogliono essere ritornati. Sugli attributi è possibile eseguire operazioni di uguaglianza, come quella mostrata a riga 7, che devono essere contenuti all'interno di un tag "filter" nel caso ci siano più condizioni. Tale tag indica come le condizioni siano legate. [16]

4.3 Architettura

Di seguito viene mostrata una possibile architettura che implementa il meccanismo di interrogazione in linguaggio naturale dei dati del CRM. I principali attori sono:

- lo user, che è la persona che decide quali domande porre, e le passa al componente Control Logic Unit;
- la Control Logic Unit, che è l'elemento che si occupa di orchestrare la logica di funzionamento;
- la AI Interface Unit, che è l'elemento che permette di interagire con il modello AI offerto da Azure OpenAI;
- il Dataverse, una base dati, di cui si è già parlato, che contiene i dati di interesse;

4.3.1 Logica di funzionamento

Il paragrafo entra nei dettagli della logica di funzionamento, che è stata mostrata nel capitolo sulla progettazione, definendo a basso livello tutte le interazioni.

Lo User definisce la domanda da porre al sistema e la invia al componente Control Logic Unit. In questa prima fase, dove ancora non è stata implementata una interfaccia per l'inserimento dei dati, l'inserimento e l'invio avvengono tramite PostMan. Esso è una piattaforma per costruire e utilizzare API. La Control Logic Unit espone un servizio, tramite API, raggiungibile dallo User, che, così, passa

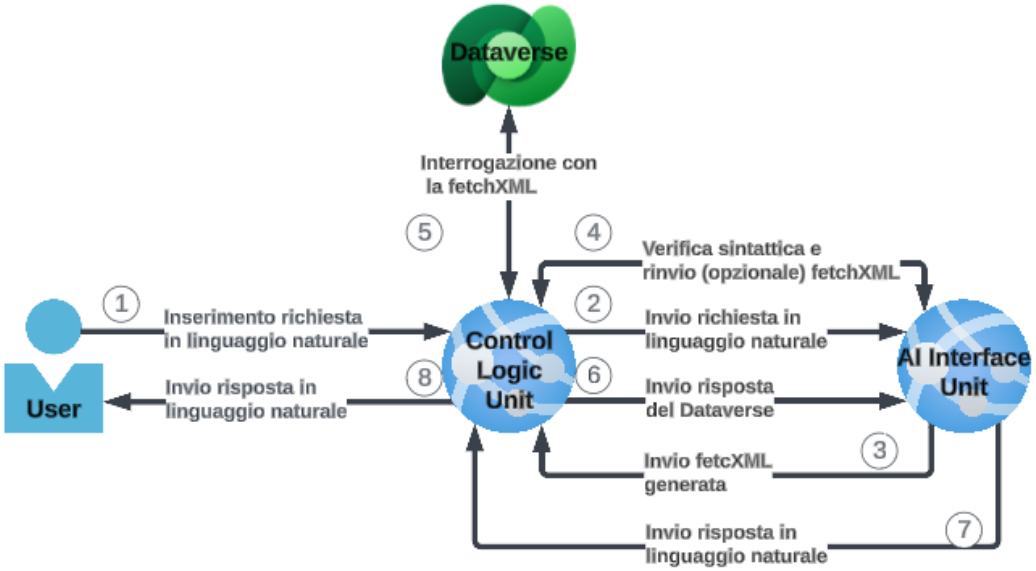

Figura 4.1: Modello architetturale

la sua richiesta. La Control Logic Unit deve ottenere le informazioni sul modello dati, che sono ottenibili interrogando il Dataverse. Dopodiché, vengono passate alla AI Interface Unit, tramite una opportuna API. L'invio di questi metadati avviene sotto precisa richiesta dell'utente, o al passare di un certo limite di tempo. Successivamente, la Control Logic Unit utilizza una API resa disponibile dalla AI Interface Unit per passargli la richiesta dell'utente. L'AI Interface Unit inizia un processo di selezione dei metadati utili per formulare la richiesta dell'utente, interagendo, tramite l'astrazione fornita da LangChain, col modello GPT-4-32k offerto dal servizio Azure OpenAI. La richiesta viene convertita nella forma di una fetchXML, che viene inviata, tramite una opportuna API, alla Control Logic Unit. Questa unità esegue una verifica sintattica della fetchXML, interrogando il Dataverse. In base al risultato prevede due strade. In caso di verifica fallita, la Control Logic Unit contatta nuovamente l'AI Interface Unit per richiedere la generazione di una nuova fetchXML, fornendo il motivo di errore delle precedenti. Questo meccanismo prevede un numero di 5 tentativi, al termine dei quali viene ritornata la lista di errori sintattici che si sono verificati. Altrimenti, se la verifica fornisce risultato positivo, il Dataverse ritorna i dati richiesti dalla fetchXML. Tale risposta viene fornita insieme alla domanda al AI Interface Unit, che effettua, interfacciandosi con il modello GPT, un processo di interpretazione. Tale processo richiede la conoscenza dei metadati e l'interrogazione dell'utente. La risposta, così generata, viene propagata, prima alla Control Logic Unit, e, successivamente, all'utente.

Nei prossimi paragrafi si evidenziano i dettagli implementativi distinguendoli in base alle fasi alle fasi implementative, che qui si ricordano essere:

1. Una prima fase di progettazione del sistema di interrogazione in linguaggio naturale su più tabelle, ma con la limitazione che la richiesta richieda il coinvolgimento di al più una di esse;

2. Una seconda fase che prevede l'estensione della precedente per la creazione di interrogazioni più complicate che possono richiedere il coinvolgimento di più tabelle e su cui sono possibili operazioni di max, min e count;

4.3.2 Control Logic Unit

È il componente che si occupa di orchestrare tutto il meccanismo, che permette di interrogare i dati del CRM, ospitando tale logica all'interno di un App Service Azure. La struttura di tale componente prevede la presenza di:

- un file Program.cs che contiene le API e la logica di interazione con gli altri componenti;

- un file GetEntitySchema.cs che contiene la logica di richiesta dei metadati;

- una cartella EntityMetadata contenente i metadati sulle tabelle e sui loro campi;

- una cartella Response contenente la risposta all'interrogazione del Dataverse;

Program.cs - Interrogazione

Il file Program.cs contiene una serie di API richiamabili esternamente. L'endpoint "\" permette all'utente di avviare il meccanismo di traduzione, passando la richiesta di interrogazione. Per prima cosa, viene generata una connection string, che è un elemento costituito da una serie di coppie chiave valore, utilizzato per stabilire la connessione con il Dataverse col ruolo di amministratore.

```
1         string connectionString = $"AuthType=ClientSecret;Url={url
    };ClientId={clientId};ClientSecret={clientSecret};Authority={
    authority}";
```

In tale estratto di codice:

- l'url indica l'URL dell'istanza del dataverse;

- clientId contiene, in questo caso, l'applicationId, cioè un identificativo dell'applicazione, che insieme al segreto contenuto nel ClientSecret, permette di accedere col ruolo di amministratore;

- Infine, Authority indica l'organizzazione o il servizio che gestisce e controlla l'accesso all'applicazione.

Successivamente, l'endpoint prevede un meccanismo ciclico di chiamate "Post" verso l'AI Interface Unit. Il corpo contiene un oggetto UserInput, che definisce la richiesta dell'utente, e i messaggi di errore scaturiti da un tentativo precedente. Il valore di ritorno della Post all'AI Intereface Unit contiene la fetchXML ed una variabile di stato che ne indica il successo della creazione. All'interno del ciclo, che si ripete fino ad un massimo di cinque volte, ad ogni iterazione avviene la verifica sintattica tramite un tentativo di interrogazione sul Dataverse con la funzione "QueryDataverse". Tale funzione è eseguita all'interno di un contesto "try-catch" che esce dal ciclo in caso positivo, altrimenti, lancia una eccezione catturata dal blocco "catch", che concatena l'errore alla variabile nell'oggetto UserInput che ne tiene traccia. In questo modo, si fornisce l'informazione all'AI Interface Unit di ogni errore che si verifica nel corso dei 5 tentativi. Il blocco "catch" si occupa inoltre di intercettare possibili errori legati al raggiungimento del limite sul numero di record su cui è possibile eseguire operazioni aggregate. Tale limite è di 50.000 record ed è scelto al fine di mantenere le prestazioni e l'affidabilità del sistema. In questo caso, è previsto un messaggio che notifica all'utente la necessità di essere più specifico nella richiesta. Infine, i dati che sono ritornati dal Dataverse vengono inviati tramite un ulteriore "Post" all'AI Interface Unit, che ne interpreta il contenuto fornendo la risposta in linguaggio naturale, che a sua volta viene passata all'utente come valore di ritorno.

```
1  app.MapPost("/", async (UserInput userInput) =>
2  {
3
4      //Creo connection string
5      string url = "url"; //l'URL della istanza
6      string clientId = "Application Id"; //Application Id
7      string clientSecret = "Secret"; //Secret
8      string authority = "authority"; //Tenant Id
9      string connectionString = $"AuthType=ClientSecret;Url={url};
       ClientId={clientId};ClientSecret={clientSecret};Authority={
       authority}";
10
11     string errorMessagesIterations = "";
12     var postData = new UserInput
13     {
14         Input = userInput.Input,
15         ErrorMessagesIterations = errorMessagesIterations
```

```
16      };
17
18      HttpClient httpClient = new HttpClient
19      {
20          //Inserire Uri secondo web service
21          BaseAddress = new Uri(baseAddress)
22      };
23      string endpointFetchXML = "inputservice";
24
25      for (int i = 0; i < 5; i++)
26      {
27          Console.WriteLine($"Iterazione n {i}");
28          string fetchResponse = await PostAsyncFetch(httpClient,
    postData, endpointFetchXML);
29          JObject jsonObject = JObject.Parse(fetchResponse);
30          string status = jsonObject["status"].ToString();
31          string message = jsonObject["message"].ToString();
32
33          //Verifica se:
34          //- The model was not able to find any table of interest
    for the user input;
35          //- The tables of interest for the user input are not all
    related.There is at least a table with no relationship
36          if (status == "error")
37          {
38              return Results.Ok(message);
39          }
40          Console.WriteLine("FetchXML:" + message);
41
42          try
43          {
44              string response = QueryDataverse(message,
    connectionString);
45
46              string endpointSaveResponseFlask = "saveResponse";
47              await PostAsyncResponse(httpClient, response,
    endpointSaveResponseFlask);
48              break;
49          }
50          catch (Exception ex)
51          {
52
53              if (ex.Message.Contains("AggregateQueryRecordLimit"))
54              {
55                  return Results.Ok($"Please try to formulate a more
     specific request. An outdated aggregation limit has been
    launched");
56              }
```

```
57              Console.WriteLine($"Entered in the catch expression on
    {i} iteration with error code {ex.Message}");
58              errorMessagesIterations = errorMessagesIterations + $"
    - {ex.Message}\n";
59              postData.ErrorMessagesIterations =
    errorMessagesIterations;
60              Console.WriteLine(postData.ErrorMessagesIterations);
61              if (i== 4) {
62
63                  Console.WriteLine($"Errors occurred in the main
    MapPost due to post status code.Error Messages: \n{
    errorMessagesIterations}");
64                  return Results.Ok($"An error occurred in the main
    MapPost due to post status code.Error Message: {ex.Message}");
65
66              }
67          }
68      }
69      string endpointFinal = "responseResult";
70      string result = await PostAsyncFinal(httpClient, postData,
    endpointFinal);
71
72      return Results.Ok(result);
73  }
74  );
```

La funzione QueryDataverse accetta come parametri la connection string, descritta precedentemente, e la fetchXML. Innanzitutto, crea una istanza della classe ServiceClient a partire dalla connectionString. ServiceClient è una classe che permette l'autenticazione utilizzando la Microsoft Authentication Library (MSAL). Successivamente, si utilizza l'interfaccia IOrganizationService, che fornisce i metodi per interrogare il Dataverse. In particolare, si utilizza il metodo "RetrieveMultiple", che prende in input la fetchXML e ritorna un oggetto della classe EntityCollection contenente una raccolta di istanze di entità. Infine questo oggetto viene serializzato e salvato come un file json nella cartella "Response".

Program.cs - Richiesta metadati

Il sistema prevede due metodologie per richidere i metadati, una temporizzata ed un'altra che richiede un input dell'utente. Entrambe seguono la stessa logica, che prevede:

- la creazione di una connection string;
- la definizione hardcode delle tabelle per cui richiedere i metadati;

- l'esecuzione del metodo SaveEntitySchema;
- invio dei metadati all'AI Interface Unit;

La funzione SaveEntitySchema accetta come parametri la connectionString ed il vettore di stringhe contenente il nome logico delle tabelle. Per prima cosa defisce un oggetto ServiceClient a partire dalla stringa e, successivamente, istanzia un oggetto della classe GetEntitySchema. Per ogni tabella chiama il metodo Execute di quest'ultima classe, che prevede in input la tabella corrente e il vettore di tabelle. La classe GetEntitySchema è definita nel file col medesimo nome e prevede:

- un attributo privato contenente la connectionString;
- un metodo Execute pubblico;
- una serie di metodi privati;

Il metodo Execute richiama i due metodi privati GetEntityInfoSchema e GetEntityAttributesSchema, che richiedono i metadati dello schema della tabella e degli attributi. Una volta, ottenuti, questi vengono compattati e inviati all'AI Interface Unit.

GetEntityInfoSchema utilizza una istanza della classe RetrieveEntityRequest, in cui si specificano le proprietà per definire quali dati il Dataverse deve ritornare.

```
1 var request = new RetrieveEntityRequest
2 {
3     EntityFilters = EntityFilters.Attributes,
4     LogicalName = entityLogicalName
5 };
6 var response = (RetrieveEntityResponse)service.Execute(request);
```

Ad esempio, nel nostro caso si è interessati ai soli attributi dell'entità indicata nella variabile "entityLogicalName", che vengono richiesti tramite la proprietà "EntityFilters.Attributes". Tale proprietà ritorna le informazioni sull'entità più le informazioni sui suoi attributi. L'esecuzione del metodo Execute sulla richiesta appena descritta restituisce una istanza della classe RetrieveEntityResponse, che contiene le informazioni richieste. La proprietà EntityMetadata della classe RetrieveEntityResponse permette di accedere a tali dati fornendo una istanza della classe EntityMetadata. A questo punto, si definisce una classe, AttributeDetails, contenete tutte le informazioni di cui si è interessati. Tramite l'istanza della classe EntityMetadata, è possibile accedere al metodo Attributes, che ritorna un array di attributi. Per ognuno di essi estrapoliamo:

- il labelName dell'attributo, cioè l'alias;

- il logicalName dell'attributo, il nome tecnico dell'attributo;
- la descrizione;
- il tipo;
- la proprieta Searchable;

La proprietà Searchable indica la possibilità di applicare condizioni sul valore dell'attributo, che, nel caso risulti falsa, può comunque essere richiesto per essere mostrato. Se un attributo è di tipo PickList si estraggono le opzioni disponibili e le si memorizza in una lista di oggetti PicklistOption. Ogni oggetto PicklistOption contiene un'etichetta (il testo visualizzato per l'opzione) e un valore.

```
1  foreach (var attribute in entityMetadata.Attributes)
2  {
3      AttributeDetails details = new AttributeDetails
4      {
5          LabelName = attribute.DisplayName?.UserLocalizedLabel?.
       Label,
6          LogicalName = attribute.LogicalName,
7          Description = attribute.Description?.UserLocalizedLabel?.
       Label,
8          AttributeType = attribute.AttributeType.ToString(),
9          Searchable = attribute.IsValidForAdvancedFind.Value
10     };
11
12     if (attribute.AttributeType == AttributeTypeCode.Picklist)
13     {
14         // Handle picklist attributes and their options
15         PicklistAttributeMetadata picklistAttribute = attribute as
       PicklistAttributeMetadata;
16         if (picklistAttribute != null)
17         {
18             details.AttributeType = "PicklistType";
19             details.PicklistOptions = new List<PicklistOption>();
20
21             foreach (OptionMetadata option in picklistAttribute.
       OptionSet.Options)
22             {
23                 details.PicklistOptions.Add(new PicklistOption
24                 {
25                     Label = option.Label?.UserLocalizedLabel?.
       Label ?? "",
26                     Value = (int)option.Value
27                 });
28             }
29         }
30     }
```

```
31
32      attributeDetails[attribute.LogicalName] = details;
33  }
```

Tali informazioni vengono, poi, definite in formato JSON e salvate nella cartella EntityMetadata. Tale cartella viene, infine, inviata all'AI Interface Unit.

GetInfoSchema esegue un qualcosa di molto simile a quello enunciato in precedenza, cambiano, esclusivamente, le informazioni richieste. Si richiedono:

- il labelName dell'entità;
- il logical name dell'entità;
- la descrizione;
- i campi obbligatori;
- i campi Searchable;
- le relazioni "many to one", "many to many" e " one to many", esclusivamente con le altre tabelle scelte;

```
1     SchemaDetails details = new SchemaDetails
2     {
3         LabelName = entityMetadata.DisplayName?.UserLocalizedLabel
      ?.Label,
4         LogicalName = entityMetadata.LogicalName,
5         Description = entityMetadata.Description?.
      UserLocalizedLabel?.Label,
6         MandatoryFieldsOnCreate = entityMetadata.Attributes.Where(a
       => a.RequiredLevel.Value == AttributeRequiredLevel.
      ApplicationRequired || a.RequiredLevel.Value ==
      AttributeRequiredLevel.SystemRequired).Select(a => a.
      LogicalName).ToList(),
7         SearchableFields = entityMetadata.Attributes.Where(a => a.
      IsValidForAdvancedFind.Value).Select(a => a.LogicalName).ToList
      (),
8         ManyToManyRelations = entityMetadata.
      ManyToManyRelationships.Where(r => tablesCopy.Contains(r.
      Entity1LogicalName) || tablesCopy.Contains(r.Entity2LogicalName
      ))
9             .Select(r => new Microsoft.Xrm.Sdk.Metadata.
      ManyToManyRelationshipMetadata
10            {
11                IntersectEntityName = r.IntersectEntityName,
12                Entity1LogicalName = r.Entity1LogicalName,
13                Entity2LogicalName = r.Entity2LogicalName,
```

```
14                  Entity1IntersectAttribute = r.
    Entity1IntersectAttribute,
15                  Entity2IntersectAttribute = r.
    Entity2IntersectAttribute
16              }).ToArray(),
17          ManyToOneRelations = entityMetadata.ManyToOneRelationships.
    Where(r => tablesCopy.Contains(r.ReferencedEntity) ||
    tablesCopy.Contains(r.ReferencingEntity))
18              .Select(r => new Microsoft.Xrm.Sdk.Metadata.
    OneToManyRelationshipMetadata
19              {
20                  ReferencedEntity = r.ReferencedEntity,
21                  ReferencingEntity = r.ReferencingEntity,
22                  ReferencedAttribute = r.ReferencedAttribute,
23                  ReferencingAttribute = r.ReferencingAttribute
24              }).ToArray(),
25          OneToManyRelations = entityMetadata.OneToManyRelationships.
    Where(r => tablesCopy.Contains(r.ReferencedEntity) ||
    tablesCopy.Contains(r.ReferencingEntity))
26              .Select(r => new Microsoft.Xrm.Sdk.Metadata.
    OneToManyRelationshipMetadata
27              {
28                  ReferencedEntity = r.ReferencedEntity,
29                  ReferencingEntity = r.ReferencingEntity,
30                  ReferencedAttribute = r.ReferencedAttribute,
31                  ReferencingAttribute = r.ReferencingAttribute
32              }).ToArray()
33      };
```

La gestione delle informazioni per le relazione è stata realizzata solo successivamente, per la fase due, che prevedeva la possibilità di richiedere interrogazioni su più tabelle. Per ogni relazione si definisco quali sono le entità in gioco e su quali attributi avviene la relazione. Queste informazioni sono disponibili attraverso la proprieta ManyToManyRelationships, o simili, che ritorna tutte le relazioni "many to many" dell'entità corrente. Di queste, si è interessati solo a quelle che si riferiscono alle altre tabelle selezionate all'inizio.

4.3.3 AI Interface Unit

L'AI Interface Unit è il componente che fa da tramite tra la Control Logic Unit e il modello AI offerto da Azure OpenAI. LangChain, descritto all'inizio di questo capitolo, è il framework utilizzato per interfacciarsi con il modello AI al fine di ottenere la traduzione del linguaggio naturale in una interrogazione in formato fetchXML. Tale componenente offre una serie di API per:

- ricevere l'input dell'utente e avviare il meccanismo di traduzione;
- memorizzare la risposta del Dataverse;
- richiedere una interpretazione della risposta precedente;
- memorizzare i metadati;

La trattazione segue le due fasi di implementazione.

4.3.4 Implementazione fase 1

La fase 1 è la fase iniziale in cui si è verifica la fattibilità della traduzione dell'interrogazione in linguaggio naturale. In questa prima fase, l'interrogazione deve richiedere l'utilizzo di una sola tabella, tra quelle definite all'inizio, per effettuare la traduzione. Il metodo principale è "fetch_xml_chain". Il servizio API che richiama tale metodo, per prima cosa crea un oggetto, usando la classe AzureChatOpenAI, che permette di interagire con l'istanza del servizio OpenAI ospitata in Azure. L'oggetto "chat" viene istanziato con i seguenti campi:

- openai_api_base= contiene l'URL dell'istanza del servizio AzureOpenAI;
- openai_api_version= indica la versione dell'OpenAI API utilizzata;
- deployment_name= è il nome assegnato alla distribuzione su Azure, che fornisce l'endpoint ai modelli OpenAI. Nel caso del progetto si fa riferimento ad una distribuzione che utilizza il modello GTP-4-32k;
- openai_api_key= definisce la OpenAI API key per accedere al sevizio;
- openai_api_type= definisce il tipo di OpenAI API usata, nel nostro caso Azure;

```
1 chat = AzureChatOpenAI(
2     openai_api_base=os.environ.get("OPENAI_API_BASE"),
3     openai_api_version="2023-03-15-preview",
4     deployment_name=os.environ.get("
    AZURE_OPENAI_GPT4_DEPLOYMENT_SAGO"),
5     openai_api_key=os.environ.get("OPENAI_API_KEY"),
6     openai_api_type=os.environ.get("OPENAI_API_TYPE"),
7 )
```

A questo punto, si presenta il contenuto della "fetch_xml_chain". Essa prevede la definizione di una stringa che verrà usata per generare un prompt di sistema al fine di fornire il contesto generale. In particolare contiene il seguente testo:

```
1 system_prompt_template = """System Prompt:
2 You are part of a multi-agent LLM chain designed to facilitate
     querying a Dataverse database using natural language.
3 You have access to the following tables:
4  - Table logical name : new_issue (label name : Issue).
5  - Table logical name : new_document (label name : Document).
6  - Table logical name : new_program (label name : Program).
7
8 Given the user input "{user_input}", your goal is to
     collaboratively process the query. You will receive the output
     of the previous agent's action in each prompt. Provide
     instructions for the next agent based on the previous output.
9 """
```

Il sistema viene avvisato della presenza di più catene, ognuna delle quali svolge un micro task. Si fa riferimento a tali catene col nome di "agent", che però non va inteso come una istanza del componente Agent di Langchain, ma come un elemento di interazione col modello AI che non ha la capacità di prendere decisioni o agire autonomamente. Tali agenti collaborano dividendo l'obiettivo in sotto task e forniscono informazioni al prossimo agente tramite un prompt. Vengono, inoltre, definite quali siano le tabelle e per ognuna di esse si indica il "label" e "logical" name. Infine, viene fornito in modo dinamico, tramite una variabile, l'interrogazione dell'utente. Dopodiché, vengono definiti i compiti dei vari agenti tramite altre stringhe, che diventeranno anche loro prompts. Vediamo, adesso, quali siano tali prompts e cosa definiscano:

- l'agent 1 analizza l'input e determina la tabella da interrogare e restituisce solo il suo nome logico. Si fornisce, inoltre, un esempio di formato di risposta: "example answer: [new_issue], [new_document], [new_program]";
- l'agent 2 ha il compito di selezionare le colonne più rilevanti della tabella selezionata in base all'interrogazione dell'utente. Richiede di ritornare il solo logical name nel formato "[new_name, new_description, new_originalissuetypecode]";
- l'agent 3 compone una query fetchXML per recuperare i dati dal database in base alla richiesta dell'utente. Gli vengono specificate delle istruzioni da eseguire, quali:
- ricordati di non usare mai il label name della tabella,;
- per i campi Picklist usa il valore intero associato,;
- usare la tabella e le colonne selezionate precedentemente;

- di restituire la sola interrogazione FetchXML, fornendone un esempio:

  <fetch version="1.0" output-format="xml-platform" mapping="logical" distinct="false">.

  Al fine di creare una corretta fetchXML vengono fornite le informazioni sul tipo di attributi e, se di tipo Picklist, i corrispettivi valori;

Le precedenti azioni dei vari agents, sono inizialmente dichiarate come semplici stringhe e, poi, vengono trasformate in oggetti "AIMessagePromptTemplate" attraverso il metodo from_template che accetta una stringa in input.

La classe AIMessagePromptTemplate viene utilizzata,nel nostro contesto, per fornire suggerimenti al modello AI per generare risposte che siano pertinenti a un particolare argomento o compito. Il modello genererà quindi una risposta basata sul messaggio AI e sul suo training dei dati. La stringa di sistema viene utilizzata, invece, per la creazione di un "SystemMessagePromptTemplate", che rappresenta il contesto e le informazioni generali.

A partire da questi PromptTemplate creati, vengono generati tre ChatPromptTemplate, uno per ogni interazione col modello, che permettono di creare prompt per un modello AI di chat.

```
1 chat_prompt_1 = ChatPromptTemplate.from_messages(
2         [system_message_prompt, agent_1_message_prompt])
3 chat_prompt_2 = ChatPromptTemplate.from_messages([
    agent_2_message_prompt])
4 chat_prompt_3 = ChatPromptTemplate.from_messages([
    agent_3_message_prompt])
```

Dopodiché, vengono definite le chain tramite il costrutto LLMChain, che permette di combinare il modello AI con i ChatPromptTemplate. Ogni chain viene, poi, eseguita tramite il metodo run, a cui vengono passati i valori dinamici sotto forma di variabili.

```
1 agent_1_chain = LLMChain(llm=chat, prompt=chat_prompt_1, verbose=
    verbose)
2 answer = agent_1_chain.run(
3     user_input=user_input
4 )
```

L'agent_1 seleziona la tabella, dopodiché, si verifica che la tabella selezionata sia una di quelle di cui si possiedono i metadati, ed, in caso positivo, si ritorna la descrizione della tabella e i campi "Searchable". Viene creata una nuova chain, che ha l'obiettivo di scegliere in base al nome degli attributi, quali siano i più utili.

```
agent_2_chain = LLMChain(llm=chat, prompt=chat_prompt_2, verbose=
    verbose)
answer = agent_2_chain.run(
    user_input=user_input,
    table_name=table_name,
    table_description=table_description
)
```

Infine, per gli attributi selezionati si ritornano tutte le informazioni, compresi il tipo e le opzioni, se presenti, in modo da generare la fetchXML. Vengono fornite anche informazioni hard coded di contesto come la data, lo userID e lo userName

```
agent_3_chain = LLMChain(llm=chat, prompt=chat_prompt_3, verbose=
    verbose)
answer = agent_3_chain.run(
    user_name="Gabri",
    current_date=datetime.datetime.now().strftime("%Y-%m-%d %H:%M
    :%S"),
    user_id="123456789",
    user_input=user_input,
    table_description=table_description,
    selected_columns_description=selected_columns_description
)
```

L'agent_3 con tali informazioni è in grado di generare una fetchXML, che viene ritornata alla Control Logic Unit. Si noti come in questa prima versione, si sia cercato di verificare le capacità di comprensione del modello sui nomi delle entità/attributi. Difatti, le scelte sulle tabelle e sugli attributi vengono svolte esclusivamente in base al loro nome. Ciò limita l'affidabilità del sistema ma ne aumenta la scalabilità.

4.3.5 Implementazione fase 2

La seconda fase di implementazione ha l'obiettivo di estendere il funzionamento su più tabelle e rendere possibili operazioni aggregate di conteggio, minimo e massimo. Al fine di fare ciò, si sono modificati gli AI prompt nel seguente modo:

- all'agent_1 viene detto di analizzare l'input e scegliere, stavolta, anche più tabelle tra quelle che possono essere interessate dall'interrogazione. A differenza di prima, gli viene detto che la scelta deve essere fatta in base alla descrizione delle tabelle e che, se nessuna tabella viene individuata, il modello deve ritornare una stringa costante "No tables". Viene, inoltre, indicato di ritornare

il logical name della tabella e viene fornito un esempio di output nel caso di singola o multipla scelta.

- all'agent_2 viene fornito l'input dell'utente e richiesto, stavolta, di selezionare le colonne più rilevanti tra le tabelle scelte. Gli viene indicato che le colonne rilevanti possono essere presenti in tabelle selezionate distinte e che la scelta deve essere effettuata in base alla descrizione della colonna. Gli viene chiesto di ritornare solo il logical name e che questo sia concatenato con il logical name della tabella di cui fa parte. Vengono, inoltre forniti degli esempi di risposta:

```
1 example answer single entity: ["tableName-accountid", "
    tableName-accountnumber", "tableName-address1_addressid"]
    where all belong to account table
2 example answer multiple entity: ["tableName-accountid", "
    tableName-accountnumber", "tableName-address1_addressid", "
    tableName-clu_accountid"] where clu_accountid belongs to "
    clu_customervehicle" and the others to "account"
```

- all'agent_3 viene detto di comporre la fetchXML, dato l'input dell'utente. Inoltre, gli vengono passate le tabelle, le colonne precedentemente selezionate, una serie di indicazioni ed, in aggiunta a prima, le relazioni tra le tabelle selezionate;

Le indicazioni fornite all'agent_3 sono le seguenti:

- Ricorda che la query fetchXML deve sempre restituire solo i primi dieci record. Questo perché il progetto mira a rispondere a interrogazioni puntuali;
- Ricorda di utilizzare sono relazioni inner, e quindi avere link-type=“inner”;
- Ricorda che l'attributo nel “from” di un tag link-entity deve appartenere all'entità che appare nell'attributo name del tag;
- È possibile inserire un tag link-entity in un altro solo se esiste una relazione tra di loro, che puoi verificare con le relazioni passate;
- Se si utilizza l'operatore 'in', valori multipli dovrebbero essere specificati come elementi separati;
- Quando l'input di un utente contiene una richiesta per un conteggio, un max e un min, è importante utilizzare la funzione di aggregazione nella query FetchXML;

- Quando si utilizza un operatore aggregato, ricordarsi di specificare l'operazione aggregata nell'intestazione della fetchXML;
- Se l'input dell'utente richiede l'aggregazione su alcuni attributi, usare 'group by' su questi attributi;
- Quando l'input di un utente contiene una richiesta di max, è importante usare l'operatore max.
- Quando l'input di un utente contiene una richiesta per un min, è importante usare l'operatore min.
- Il tag order deve contenere un alias se la fetchXML è una query aggregata.
- Un alias non può essere specificato per una clausola d'ordine in una query senza aggregati.
- Qualsiasi attributo non utilizzato in una funzione aggregata come max, min, count dovrebbe essere incluso in una clausola groupby per essere ritornato come valore da visualizzare;
- Un input dell'utente che richiede il più basso di qualcosa dovrebbe usare la funzione aggregata min;
- Se un attributo viene utilizzato come condizione in una query, è generalmente una buona pratica selezionarlo per l'output. Questo perché l'attributo può contenere informazioni rilevanti che aiutano a comprendere i risultati della query.
- Ricorda che l'alias dovrebbe essere in formato "snake case", non usare mai lo spazio.
- Considera di non ripetere gli errori di esecuzione precedenti. Qui vengono passati i messaggi di errori di possibili esecuzioni precedenti su quella interrogazione
- L'intestazione della fetchXML deve essere come la seguente: <fetch version="1.0" output-format="xml-platform" mapping="logical" distinct="false" top="10">
- Ricorda la data odierna. Questa viene passata nelle informazioni di contesto dell'agent_3;
- Restituisci solo la query FetchXML;

La natura di queste indicazioni nasce dagli errori comuni che il modello realizza, al fine di limitarne la casistica.

Si noti come in questa seconda implementazione, le scelte sulle tabelle e le colonne vengano eseguite sulla base delle loro descrizioni, rendendo il modello più affidabile nelle scelte, ma, anche, meno scalabile.

4.3.6 Interpretazione della risposta(Comune ad entrambe le fasi)

Continuiamo la trattazione con lo studio del meccanismo di interpretazione dei dati per costruire una risposta in linguaggio naturale. Esso è lo stesso per entrambe le fasi di implementazione. Una volta che la Control Logic Unit riceve la risposta dal Dataverse, questa viene inviata all'AI Interface Unit per essere tradotta in linguaggio naturale. La funzione che si occupa di fare ciò è la "processing_response_chain". Essa è costituita da un prompt di sistema e un AI prompt, realizzati come prima. A partire da questi, si definisce un ChatPromptTemplate, che viene usato per istanziare una LLMChain, chiamata agent_1_chain. Il prompt di sistema fornisce il contesto al modello AI, indicando la presenza di una catena LLM single-agent progettata per creare una risposta di linguaggio naturale. Si indica al sistema che ha accesso a un file contenente le informazioni per formulare la risposta in linguaggio naturale ed alla domanda in linguaggio naturale dell'utente. Per quanto riguarda l'AI prompt, esso contiene una serie di indicazioni:

- È necessario utilizzare le informazioni nel file e la domanda per creare una risposta dettagliata;
- È necessario utilizzare tutte le informazioni nel file per costruire la risposta;
- È necessario visualizzare le informazioni in una tabella markdown con diverse colonne se il file contiene più di due entità;
- Non è necessario fare riferimento al file nella risposta dettagliata;
- È necessario utilizzare le descrizioni delle colonne per comporre la risposta;
- È necessario restituire solo la risposta;

Affinché, si ottengano le descrizioni dei campi è stato implementato un meccanismo che per ogni attributo ritornato nella risposta, appartenente ad una certa entità, venga ritornata la sua descrizione a partire dai metadati memorizzati nella cartella "EntityMetadata". Se la risposta non contiene nessun elemento, per esempio non esiste un record per la richiesta fatta, allora, verrà notificata tale situazione all'utente

```
agent_1_chain = LLMChain(llm=chat, prompt=chat_prompt_1, verbose=
    verbose)
jsonResponse=json_get_response()
columns_description = get_tables_and_fields(jsonResponse)
if columns_description==-1:
    return "No data available for the request"


answer = agent_1_chain.run(
    question = user_input,
    jsonResponse = jsonResponse,
    columns_description = columns_description
)
```

Le righe di codice riportano la creazione della LLMChain ed i dati dinamici che alimentano l'AI prompt passati all'interno del metodo "run".

Capitolo 5

Valutazione e confronto con Copilot

Il capitolo si propone si descrivere i risultati ottenuti dai test effettuati. In particolar modo, si evidenziano i casi di successo ed insuccesso e, per quest'ultimi, si valutano le motivazioni. Viene poi trattato Copilot, descrivendone le capacità nell'ambito della ricerca sui dati. Successivamente, si descrivono i potenziali miglioramenti del progetto. Infine, viene realizzato un confronto con Copilot e vengono espresse le riflessioni finali.

5.1 Risultati ottenuti

5.1.1 Procedura di testing

Si desidera porre l'attenzione sui risultati ottenuti nei test. Inizialmente, i test effettuati avevano lo scopo di verificare che il modello possedesse tutti i dati necessari al fine di tradurre la richiesta dell'utente. Terminate queste verifiche preliminari ed, una volta effettuate le correzioni necessarie, le operazioni di test si sono mosse verso la verifica che il modello AI eseguisse le giuste scelte sintattiche. Qualora da un test sia emerso un nuovo comportamento che conduce ad errori sintattici, si sono utilizzati gli AI prompt per tentare di correggerli. Gli AI prompt sono stati utilizzati, quindi, per condizionare le scelte del modello AI in modo da indirizzarlo alla creazione di fetchXML sintatticamente corrette. Per la realizzazione dei suggerimenti sono state seguite le indicazioni della cosiddetta ingegneria del prompt, e sono state eseguite una moltitudine di test su ognuno dei singoli suggerimenti in modo da verificarne la comprensione. Questa metodologia ha permesso col tempo di ridurre il numero di errori e di poter eseguire operazioni

che richiedevano un supporto sintattico, quali la creazione di relazioni e operazioni aggregate.

5.1.2 Test rilevanti

In questa sezioni si mostrano i test più significativi realizzati durante il percorso di tesi. Ognuno di essi ha lo scopo di mostrare un nuovo comportamento del sistema o un suo difetto.Per ogni test si definisce l'indice di correttezza, che indica in numero di volte in cui la query creata è quella corretta(semanticamnete e sintatticamente) su una finestra di cinque esecuzioni successive.

Test 1

Nel test 1, l'utente richiede al sistema di mostrargli il nome, l'indirizzo e il numero di telefono dell'account con un certo ID. Il modello crea, generalmente, la seguente fetchXML:

```
<fetch version="1.0" output-format="xml-platform" mapping="logical
    " distinct="false" top="10">
  <entity name='account'>
    <attribute name='accountid' />
    <attribute name='address1_composite' />
    <attribute name='telephone1' />
    <filter>
      <condition attribute='accountid' operator='eq' value='4
    af7d09e-a1ae-ea11-a813-000d3a654ce0' />
    </filter>
  </entity>
</fetch>
```

La richiesta è molto semplice, si tratta di una interrogazione su singola tabella con una condizione. La fetchXML generata è corretta e presenta un indice di correttezza di 5, che mostra come il modello AI sia sempre in grado di rispondere a domande di questo tipo.

Test 2

Il test 2 presenta una richiesta dell'utente che chiede di mostrargli le informazioni sui veicoli di uno specifico brand. Il modello, generalmente, non riesce a creare nessuna fetchXML poiché le tabelle che vengono scelte per la sua creazione non hanno alcuna relazione. Il sistema rileva ciò e lo segnala all'utente bloccando la generazione della fetchXML. L'errore in questa interrogazione è legata alla scelta delle tabelle da utilizzare, che avviene in base alla loro descrizione. In particolare, l'attributo

"brand" della tabella "customer_vehicle" su cui vogliamo eseguire la condizione è semanticamente simile al contenuto informativo di una tabella "productBrand". Questo porta il modello AI a selezionare entrambe, e poiché non esiste una relazione che le lega, viene lanciata una eccezione. In questo caso l'indice di correttezza è 0.

Test 3

Nel test 3, l'utente richiede di mostrare le informazioni degli account che hanno un certo "technical type id". Al fine di generare tale interrogazione, il modello deve essere capace di gestire le relazioni fra tabelle e di saper gestire "technical type id", che è un attributo di lookup. L'interrogazione ritornata è la seguente:

```
1  <fetch version="1.0" output-format="xml-platform" mapping="logical
       " distinct="false" top="10">
2    <entity name="account">
3      <attribute name="accountid" />
4      <link-entity name="clu_customervehicle" from="clu_accountid"
       to="accountid">
5        <link-entity name="clu_vehicle" from="clu_vehicleid" to="
       clu_vehicleid">
6          <filter>
7            <condition attribute="clu_technicaltypeid" operator="eq"
        value="510334000" />
8          </filter>
9        </link-entity>
10     </link-entity>
11   </entity>
12 </fetch>
```

Tale interrogazione è parzialmente corretta. La gestione delle relazioni e della scelta degli attributi è corretta, mentre non è corretta la gestione degli attributi di lookup, che sarebbe dovuta avvenire in questo modo:

```
1  <condition attribute="clu_technicaltypeid" operator="eq" value="{3
       b9f8ae6-6a7c-40cc-935c-fd691f11b2b6}" uiname="510334000" uitype
       ="clu_technicaltype"/>
```

Dove value è l'ID della tupla di riferimento ed "uiname" è il valore del campo name di quella tupla. Il campo name viene generato automaticamente alla creazione di una nuova tupla, ed indica il nome del record. Quando l'utente richiede una condizione su un attributo di lookup, non può conoscere il suo l'ID, ma per semplicità, deve conoscere il name della tupla. Inoltre, se l'utente visualizza l'attributo di lookup nella tabella, esso contiene il valore del campo name della tupla. Ad esempio, nel caso sovrastante:

- clu_technicaltypeid è l'attributo di lookup che viene visualizzato nella tabella con il valore del campo name;
- name è l'attibuto rappresentativo della tupla a cui la lookup fa riferimento;

Quindi, affinché l'interrogazione sia corretta è necessario che il modello a partire dal contenuto di name conosca in qualche modo l'ID che identifica la tupla corrispondente. Il modello deve essere informato di questa casistica e deve ottenere tutte le informazioni utili per gestire gli attributi di lookup. L'indice di correttezza, anche in questo caso, è 0. Se l'utente passasse il reale valore dell'attributo di lookup, cioè l'ID, e non del suo riferimento, l'indice di correttezza salirebbe a 3.

Test 4

Il test 4 presenta una richiesta dell'utente che chiede di mostrargli "hotness", "opportunity id" e "probability" dell'opportunità con il valore più basso di probabilità di chiusura. La particolarità di questa interrogazione sta nella necessità del modello di generare una operazione aggregata di minimo. Di seguito la fetchXML generata:

```
1 <fetch version="1.0" output-format="xml-platform" mapping="logical
     " distinct="false" top="10" aggregate="true">
2   <entity name="opportunity">
3     <attribute name="clu_hotnesscode" alias="hotness" groupby="
     true" />
4     <attribute name="clu_opportunitynumber" alias="opportunity_id"
      groupby="true" />
5     <attribute name="clu_timingofpurchasecode" alias="
     postponing_timing" groupby="true" />
6     <attribute name="closeprobability" alias="
     probability_to_close_opportunity" aggregate="min" />
7     <order alias="probability_to_close_opportunity" descending="
     false" />
8   </entity>
9 </fetch>
```

La fetchXML è corretta. In particolare tutti gli attributi richiesti nella visualizzazione sono stati correttamente inclusi nella clausola "group by". La clausola order è stata arricchita di un alias, come richiesto dalla sintassi, e il tag aggregate è stato inserito nel giusto attributo. L'indice di correttezza è di 3.

5.2 Potenziali miglioramenti

Questo paragrafo mira a definire i principali limiti, emersi nella sperimentazione, ed alcuni possibili miglioramenti che potrebbero essere adottati in versioni future.

I principali limiti possono essere riassunti in:

- scelta errata della tabella legata ad attributi semanticamente vicini al contenuto di altre tabelle. Il modello all'inizio sceglierà le tabelle più affini semanticamente;
- incapacità di eseguire operazioni su attributi di lookup;
- mancanza di un meccanismo di memoria all'interno della conversazione;
- mancanza di un meccanismo di verifica semantica;

A partire da queste problematiche, si cerca di definire, astrattamente, un possibile miglioramento della soluzione. Un meccanismo di verifica semantica permetterebbe all'utente di avere la sicurezza che ciò che sia stato generato sia non solo sintatticamente corretto, ma coerente alla richiesta. La verifica semantica è difficilmente realizzabile, per questo motivo una possibile mitigazione potrebbe consistere nel richiedere l'ausilio dell'utente. Il sistema potrebbe fornire parallelamente alla risposta in linguaggio naturale, anche, la richiesta in formato fetchXML che il modello ha generato. A tal punto, un utente conscio può verificarne la correttezza semantica. Questa è, per esempio, la scelta intrapresa da Microsoft per il prodotto Copilot. Tale meccanismo richiede, però, che l'utente conosca il formato fetchXML e che ci sia un tempo di verifica, che potrebbe essere inconveniente.

Per il problema riguardo le operazioni sugli attributi di lookup è necessario, dapprima, analizzare la problematica. Per realizzare operazioni sugli attributi di lookup è necessario conoscere il loro valore. Tale valore individua univocamente la tupla a cui fa riferimento. Quindi, è necessario conoscere tale identificativo univoco per creare una fetchXML. L'utente non conosce tale identificativo, quindi, nella richiesta inserirà il valore corrispondente all'ID. Le strada più semplice e veloce per superare questa mancanza è definire una link-entity, cioè una relazione, che colleghi la tabella con l'attributo di lookup con la tabella in cui è presente l'attributo a cui fa riferimento. Nella relazione si imposta l'attributo di "from" con il nome dell'attributo di lookup, mentre si imposta nel "to" l'attributo a cui fa riferimento. A questo punto, all'interno della link-entity si inserisce un tag condition sull'attributo name, e si fornisce nella condizione il valore inserito dall'utente. Per fare maggiore chiarezza si mostra come l'interrogazione iniziale, la prima in ordine, venga trasformata nella seconda.

```
1 <fetch mapping="logical" output-format="xml-platform" version="1.0
    " no-lock="false" distinct="true" top="10">
2 <entity name="account">
3 <attribute name="name"/>
4 <order attribute="name" descending="false"/>
5 <attribute name="clu_relationshiptypecode"/>
6 <attribute name="clu_sourcesoftwarecode"/>
```

```
<attribute name="accountid"/>
<link-entity name="clu_customervehicle" alias="ab" link-type="
    inner" from="clu_accountid" to="accountid">
     <link-entity name="clu_vehicle" alias="ac" link-type="inner"
         from="clu_vehicleid" to="clu_vehicleid">
         <filter type="and">
             <condition attribute="clu_technicaltypeid"
                 operator="eq"
                 value=" {3b9f8ae6-6a7c-40cc-935c-fd691f11b2b6}"
                 uiname="510334000" uitype="clu_technicaltype"/>
         </filter>
   </link-entity>
</link-entity>
</entity>
</fetch>
```

```
<fetch mapping="logical" output-format="xml-platform" version="1.0
    " no-lock="false" distinct="true" top="10">
<entity name="account">
<attribute name="name"/><attribute name="address1_city"/><
    attribute name="clu_marketcode"/>
<order attribute="name" descending="false"/>
<attribute name="accountid"/>
<link-entity name="clu_customervehicle" alias="aa" link-type="
    inner" from="clu_accountid" to="accountid">
     <link-entity name="clu_vehicle" alias="ab" link-type="inner"
     from="clu_vehicleid" to="clu_vehicleid">
         <link-entity name="clu_technicaltype" alias="ac"
         link-type="inner" from="clu_technicaltypeid"
         to="clu_technicaltypeid">
             <filter type="and">
                 <condition attribute="clu_name" operator="eq"
                 value="510334000"/>
             </filter>
         </link-entity>
     </link-entity>
</link-entity>
</entity>
</fetch>
```

Affinché possa essere gestita questa trasformazione è necessario che il modello sia informato tramite gli AI prompt. Si richiede, inoltre, che vengano recuperate le informazioni sulle relazioni che legano l'attributo al suo riferimento e vengano passate al modello AI.

L'implementazione di un meccanismo che tenga memoria dei messaggi precedentemente scambiati durante la conversazione, è integrabile grazie ai costrutti forniti dal framework LangChain. Tramite questi strumenti è possibile tenere in memoria il solo messaggio precedente o una finestra contenente gli ultimi k messaggi.

Per quanto riguarda, invece, la scelta errata della tabella, a causa della richiesta di un'interrogazione su elementi semanticamente vicini al contenuto di altre tabelle, è un problema arduo da superare. Se il numero di attributi che hanno questa caratteristica fosse limitato, si potrebbe fornire una indicazione specifica al momento della prima scelta tra le tabelle. Tale indicazione potrebbe essere qualcosa del tipo: "Se la richiesta richiede l'uso di questi attributi [(attributo1,descrizione),(attributo2,descrizione),... (attributoN,descrizione)] di questa tabella, allora considera di utilizzare questa tabella. Altre tabelle possono essere comunque utilizzate per tradurre altre condizioni". Altrimenti, un'altra possibilità potrebbe essere gestire un meccanismo che, in caso non si trovi la colonna che soddisfa la richiesta, riesegua la scelta tra le tabelle tenendo a mente la nuova informazione.

5.3 Copilot

Copilot è uno strumento di assistenza, basato sull'intelligenza artificiale, progettato da Microsoft per aumentare la produttività. I suoi utilizzi spaziano dagli ambiti lavoratiti a quelli dell'istruzione, fino all'utilizzo nella vita quotidiana. Utilizza il modello GPT-4 sviluppato da OpenAI per fornire assistenza all'utente [17]. La sua integrazione è avvenuta in diverse piattaforme grazie alla versatilità di utilizzo. Ad esempio, è stato, recentemente, integrato all'interno di Windows 11 per fornire supporto all'utente per la gestione delle impostazioni del PC, ma, anche, come strumento veloce per ricevere risposte alle proprie domande. Il suo utilizzo all'interno di Windows 11 non si limita qui. Microsoft365 [18].

5.3.1 Copilot per Dynamics365

L'utilizzo di Copilot in Dynamics365 permette di sfruttare le potenzialità di un assistente AI nelle applicazioni aziendali CRM. In particolare, l'assistente fornisce valore incrementando la produttività e l'efficienza dei team di vendita, gestione del cliente, marketing e così via [19]. All'interno di un CRM si hanno spesso compiti che richiedono tempo, come l'inserimento manuale dei dati e la generazione di contenuti. Lo scopo di Copilot è di evitare che i team spendano tempo in attività ripetitive, sfruttando l'AI per automatizzarle [20]. Copilot è integrato in tutti i prodotti Dynamics365, ognuno dei quali si occupa di uno specifico ambito di gestione. In generale, Copilot può trasformare qualsiasi Model-Driven App, che siano applicazioni predefinite come Sales o Customer Service o personalizzate, in

un'applicazione intelligente. Nel seguente paragrafo si considerano le funzionalità introdotte da Copilot nello scenario delle vendite.

Microsoft Dynamics365 Sales

"Microsoft Dynamics for Sales" è un'applicazione model-driven con lo scopo di migliorare l'efficienza delle vendite e soddisfare le richieste dei clienti."Copilot for Sales" è un assistente AI progettato per semplificare le operazioni intraprese dal team di vendita per andare avanti in una vendita e concluderla. Può essere integrato sia ad un "Saleforce Sales Cloud" che ad "Microsoft Dynamics for Sales". "Copilot for Sales" integra le funzionalità di "Copilot for Microsoft365" in modo tale da sfruttarne le applicazioni. "Microsoft Dynamics for Sales" utilizza "Copilot for Sales" per offrire una serie di funzionalità, che semplificano le vendite. Tali funzionalità sono:

- permettere di mantenere i dati aggiornati e accurati in modo efficiente sfruttando l'Intelligenza artificiale. In particolare, offre supporto alla creazione di contatti, all'aggiornamento dei record di un CRM in Outlook e Teams e alla sincronizzazione dei dati. Se si apre il riquadro di "Copilot for Sales", ad esempio, su una mail ricevuta in Outlook, Copilot è in grado di precompilare i dettagli del contatto in base alla firma del mittente e di evidenziare i campi che popola, al fine di inserire questo nuovo contatto nel CRM;
- generare riepiloghi sulle informazioni dei record del CRM direttamente su Outlook e Teams;
- sfruttare Copilot in Microsoft Word che offre la creazione più veloce di documenti fornendo una lista di contenuti ed elementi consigliati, che sono semanticamente correlati. Questo consente ai venditori di creare report dettagliati prima di una riunione e di condividerli con i membri del proprio team;
- offrire supporto alla comprensione delle esigenze dei clienti al fine di creare relazioni più forti e personalizzate;
- permettere la generazione automatica di bozze di risposta ad email basate sulle email scambiate e sui dati del CRM;
- permettere di visualizzare i riepiloghi di una riunione fornendo una vista sulle parole chiave ed una analisi della conversazione;

Queste sono solo alcune delle operazioni su cui Copilot for Sales offre supporto. Nel contesto della tesi, si è interessati particolarmente all'interrogazione del CRM, operazione che viene mostrata nella figura 5.1 sottostante.

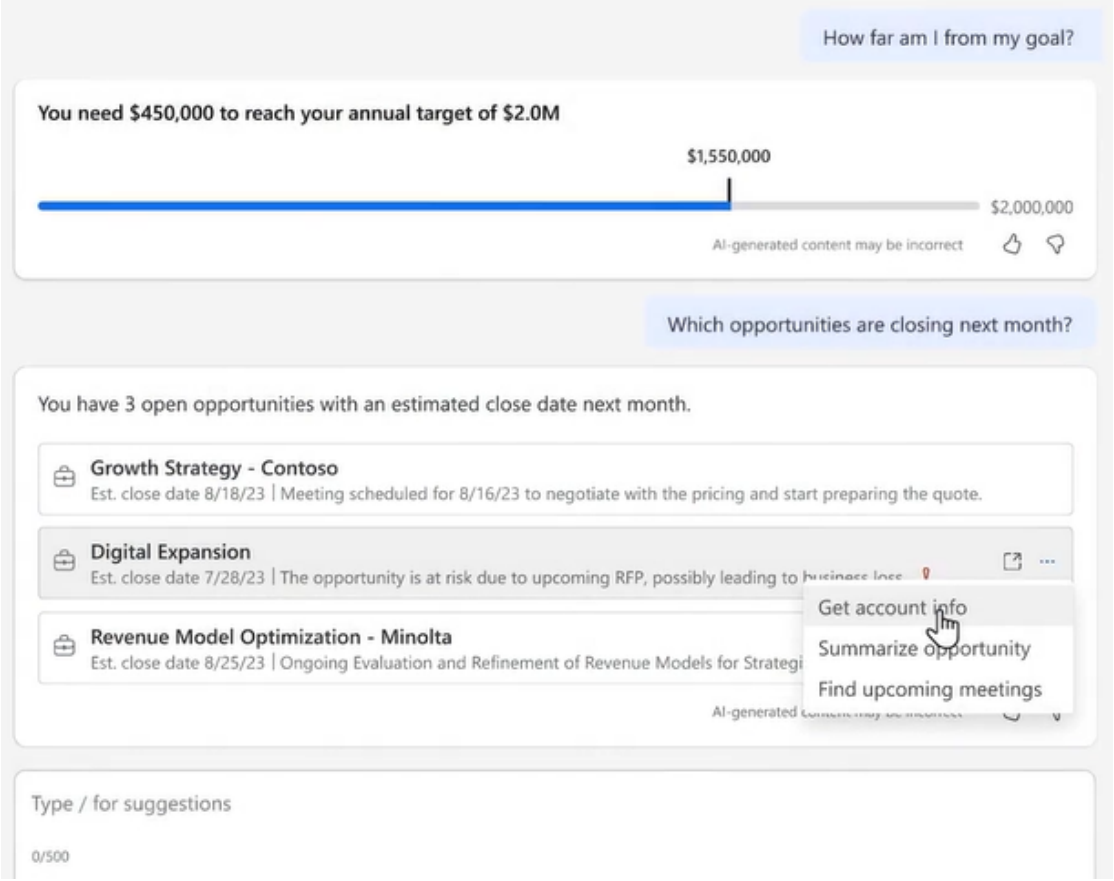

Figura 5.1: Copilot for Sales: interrogazione CRM

La figura mostra le potenzialità avanzate di ricerca sui dati del CRM. I team possono richiedere informazioni, ad esempio, sulle opportunità in chiusura nel prossimo mese. Copilot interroga il CRM e restituisce i valori. Tali valori sono cliccabili e permettono di ottenere degli "insight" su tali dati.

Copilot per le Model-Driven App

Come si è già detto, Copilot può trasformare qualsiasi Model-Driven App in un app intelligente. L'utente, tramite un'esperienza conversazionale, può interrogare i dati della Model-Driven app con domande in linguaggio naturale. Al fine di fare ciò, è necessario seguire due step di configurazione, il primo che prevede modifiche al livello di ambiente, mentre il secondo prevede cambiamenti al livello di tabelle e colonne. Il secondo step evidenzia come chi effettua la configurazione decida quali tabelle e, soprattutto, quali colonne abilitare alla ricerca. Di default la maggior parte delle tabelle standard è abilita all'uso di Copilot, mentre nel caso di tabelle

custom, le opzioni "Track changes" e "Appear in search results table" devono essere abilitate. Una volta che una tabella è abilitata, si procede con aggiungere le colonne sulle quali effettuare le ricerche accedendo alla "Quick Find View" e modificando le colonne all'interno della lista "Find by".

Funzionalità di ricerca

Copilot permette all'utente sia di eseguire interrogazioni sui dati sia di navigare lungo l'applicazione. É possibile fornire richieste che richiederebbero la definizioni di filtri/ ordinamento sui dati. É possibile utilizzare comandi speciali per richiedere particolari comportamenti. Ad esempio, la parola chiave "List" inserita come prima parola nella richiesta indica a Copilot che deve generare una opzione di navigazione su una pagina contenente una vista sulla lista richiesta. É, anche, possibile definire interrogazione che richiedono l'aggregazione su qualche attributo. La parola chiave "Summarize" posta all'inzio della richiesta permette di richiedere un riepilogo su dei dati.

5.4 Riflessioni finali e confronto

Al fine di verificare le potenzialità del progetto sono state definite delle possibili interrogazioni sui dati. La scelta di tali dati è avvenuta congiuntamente con l'organizzazione Reply Cluster, che ha definito quali fossero i più interessanti del modello dati. Su un numero di sedici interrogazioni, con richieste di natura differente, sono stati realizzate, su ognuno di essi, cinque tentativi di creazione di fetchXML per un totale di ottanta esecuzioni. Di tali esecuzioni, cinquantuno tentativi hanno prodotto una fetchXML valida sia sintatticamente che semanticamente. Tale risultato può essere considerato un buon punto di partenza per future migliorie sull'affidabilità. Durante la fase di progettazione sono stati definiti i seguenti requisiti:

1. la necessità di interrogare dati in tabelle custom o standard;
2. la necessità di eseguire interrogazioni che richiedono operazioni aggregate;
3. la necessità di rendere fruibile il servizio globalmente;
4. la necessità di non aggiornare continuamente le informazioni sul modello dati;
5. la necessità di verificare la sintassi dell'interrogazione e ripetere la sua creazione se necessaria;
6. la necessità di fornire una integrazione sul sistema Teams;

7. la possibilità di fornire risposte a domande puntuali, cioè che ritornano un numero limitato di record.

Copilot, per l'interrogazione dei dati del CRM, soddisfa pienamente i seguenti requisiti:

1. la necessità di interrogare dati in tabelle custom o standard;

2. la necessità di eseguire interrogazioni che richiedono operazioni aggregate;

3. la necessità di rendere fruibile il servizio globalmente;

5. la necessità di verificare la sintassi dell'interrogazione e ripetere la sua creazione se necessaria;

7. la possibilità di fornire risposte a domande puntuali, cioè che ritornano un numero limitato di record.

Il primo requisito era inizialmente solo parzialmente soddisfatto in quanto Copilot permetteva di interrogare solo le tabelle standard. Tale limite è stato col tempo colmato. Per quanto riguarda il settimo requisito, Copilot estende il tipo di interrogazioni a quelle non necessariamente puntuali.

Copilot soddisfa parzialmente i seguenti requisiti:

4. la necessità di non aggiornare continuamente le informazioni sul modello dati;

Copilot lo soddisfa solo in parte in quanto l'aggiornamento dell'indicizzazione in un sistema CRM può variare a seconda del sistema specifico e delle impostazioni.

Infine, Copilot non soddisfa completamente i seguenti requisiti:

6. la necessità di fornire una integrazione sul sistema Teams;

Copilot non prevede la possibilità di interrogare i dati di un CRM direttamente dall'applicazione Teams, ma esclusivamente all'interno di una model driven app.

Il progetto di tesi soddisfa appieno i seguenti requisiti:

1. la necessità di interrogare dati in tabelle custom o standard;

3. la necessità di rendere fruibile il servizio globalmente;

4. la necessità di non aggiornare continuamente le informazioni sul modello dati;

5. la necessità di verificare la sintassi dell'interrogazione e ripetere la sua creazione se necessaria;

6. la necessità di fornire una integrazione sul sistema Teams;

7. la possibilità di fornire risposte a domande puntuali, cioè che ritornano un numero limitato di record.

La tesi soddisfa appieno il terzo requisito attraverso il servizio Cloud App Service, soddisfa il quarto in quanto l'aggiornamento viene temporizzato o effettuato su diretta richiesta dell'utente. Il quinto requisito viene soddisfatto dalla tesi in quanto è previsto un blocco ciclico di cinque tentativi di costruzione della fetchXML in modo che sia sintatticamente corretta. La tesi soddisfa,inoltre, la necessità di una integrazione dello strumento su Teams, prevista dal sesto requisito, e la possibilità di definire interrogazioni puntuali.

Mentre, il progetto di tesi soddisfa solo parzialmente i seguenti requisiti:

2. la necessità di eseguire interrogazioni che richiedono operazioni aggregate;

La soluzione proposta da tale tesi fornisce supporto esclusivamente alle interrogazioni che richiedono delle operazioni aggregate quali: min, max, count.

La soluzione proposta, a differenza di Copilot non richiede una configurazione dell'ambiente e delle tabelle, ma richiede l'utilizzo di descrizioni chiare per le tabelle e colonne. Mentre, il principale limite della soluzione mostrata nella tesi è legato all'utilizzo degli attributi di lookup, che non sono interrogabili a meno di non conoscere il loro ID.

A differenza di questa soluzione, Copilot permette di:

- utilizzare comandi per offrire varie funzionalità, tra le quali la richiesta di un riepilogo su certe entità del CRM;
- eseguire interrogazioni in minor tempo grazie all'indicizzazione dei dati;
- navigare nella Model-Driven app;

Difatti, quindi, la soluzione Copilot prevede maggiori vantaggi in termini di funzionalità e tempistiche rispetto alla soluzione qui descritta, che invece ha il vantaggio di non richiedere nessuna configurazione e la possibilità di essere integrata in Teams. Infine,quindi, la soluzione di tesi può essere considerata un punto di partenza per soluzioni più competitive.

Capitolo 6

Caso d'uso: Chatbot integrato in Teams

In quest'ultimo capitolo vedremo la contestualizzazione del progetto di tesi in un caso d'uso con Teams. Inoltre,definiremo le motivazioni ed i vantaggi che portano all'integrazione del sistema in un chatbot su Teams. Si definiscono, inoltre, le fasi di progettazione e implementazione.

6.1 Motivazione

L'ultima fase della tesi ha previsto l'applicazione della soluzione appena creata in uno strumento di lavoro, quale Teams. La scelta di Teams come interfaccia per l'interrogazione dei dati è legata al largo utilizzo che le aziende fanno di tale piattaforma. Essa viene utilizzata ampiamente come strumento di comunicazione aziendale. In questo modo, il dipendente che necessità di interrogare il CRM può accedere a tale funzionalità in un ambiente familiare, che utilizza giornalmente, senza dover uscire da tale piattaforma. Se, ad esempio, un team di vendita esegue una videoconferenza su Teams per discutere delle indagini di mercato ed i membri necessitano di accedere a informazioni più approfondite, ciò è possibile usando la stessa piattaforma. L'integrazione della soluzione su Teams, fornisce in questo progetto l'opportunità di realizzare una interfaccia grafica per interagire col sistema creato.

6.2 Progettazione

Questo paragrafo illustra la logica di funzionamento del chatbot. Per prima cosa, si analizzano quali debbano essere le sue funzionalità. Il chatbot deve essere,

principalmente, in grado di connettersi al servizio, ma deve, anche, fornire la parvenza di una conversazione naturale. I principali elementi che lo costituiscono devono essere:

- un blocco che si occupi di dare il benvenuto all'utente e spiegare il contesto di utilizzo;
- un blocco in grado di interrogare il servizio sviluppato;
- un blocco che richieda all'utente se ha terminato;
- un blocco di terminazione;
- un blocco che gestisce il caso di errori nella comunicazione col servizio;

Lo schema proposto è quello mostrato in figura 6.1.

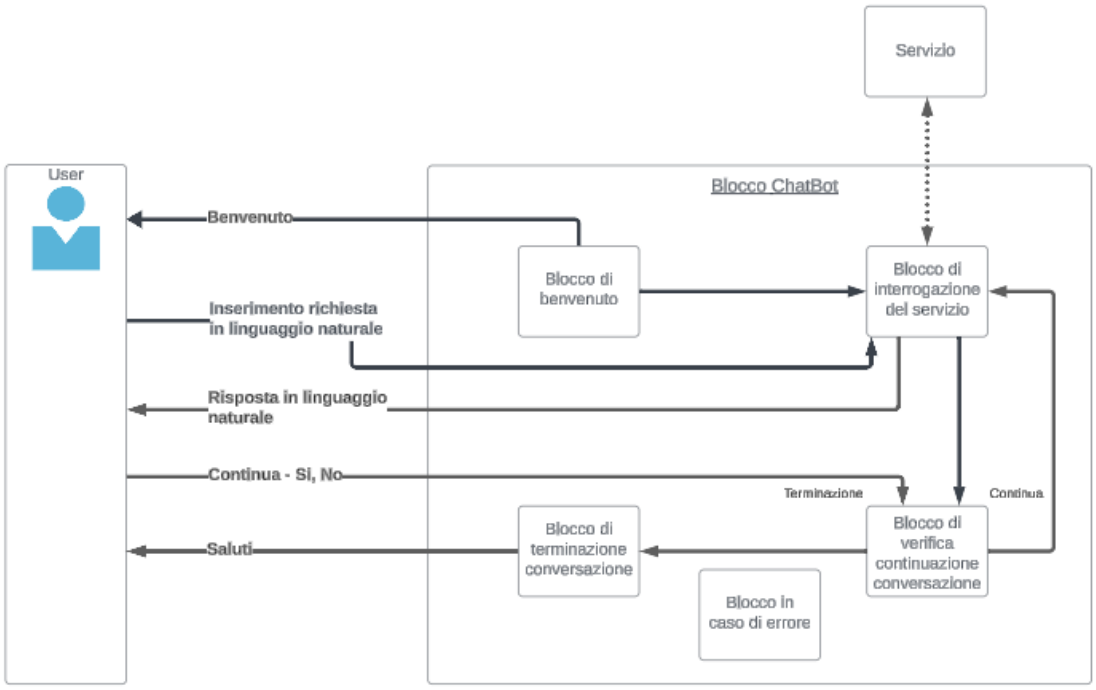

Figura 6.1: Modello progettuale chatbot

Quando l'utente avvia il chatbot, riceve un saluto ed una presentazione del chatbot. Il chatbot richiede all'utente di inserire la sua richiesta, che viene inoltrata al blocco che in figura si chiama "blocco di interrogazione del servizio". Tale blocco si occupa di ricevere l'interrogazione ed inoltrarla al servizio. In un certo istante, riceverà la risposta e la fornirà all'utente. A questo punto l'utente deve poter scegliere se continuare la conversazione o se terminarla. La continuazione richiede

che l'utente invii una nuova richiesta al "blocco di interrogazione del servizio", in un loop che termina quando l'utente decide di chiudere la conversazione. Il "blocco di terminazione conversazione" si occupa di chiudere la conversazione salutando l'utente. Infine, si ha un blocco che gestisce i casi di errore di comunicazione col servizio.

6.3 Implementazione

Questo paragrafo si pone l'obiettivo di realizzare quanto discusso prima, definendo nel dettaglio le interazioni tra gli argomenti, che rappresentano quelli che, precedentemente, si sono definiti come blocchi. L'implementazione è avvenuta tramite l'applicazione web Copilot Studio, che come si è visto in precedenza, permette la semplice creazione di chatbot che possiedono funzionalità AI. Nel caso della tesi non si è utilizzata appieno la reale forza di questo strumento, in quanto la parte di AI è stata già trattata diversamente. Ad esempio, Copilot Studio permette di definire l'argomento da scatenare in base a delle parole chiave. Ciò, significa che quando l'utente inserisce una parola, il chatbot verificherà se esiste un argomento triggerato. Se ci sono più trigger dello stesso tipo, è necessario specificare un ordine.

In questa implementazione non si è sfruttata tale funzionalità. Quando la conversazione inizia, il sistema chatbot esegue l'argomento di sistema "Conversation Start".

Start Conversation

Start Conversation è un argomento di sistema che viene triggerato nel momento in cui la conversazione ha inizio. Viene richiamato all'Inizio della conversazione, quando il chatbot viene inizializzato, e quando l'utente decide di terminare la corrente conversazione. Il chatbot, dunque, rimane sempre attivo ed in attesa di un input. Gli argomenti di sistema non sono eliminabili, ma possono, invece, essere modificati a piacimento. Nel nostro caso si prevede che sia costituito da due soli elementi:

- un messaggio di benvenuto;
- un reindirizzamento all'argomento "post Azure"

Post Azure

Tale argomento è il blocco che si occupa di ricevere l'input ed interrogare il servizio sviluppato in questo progetto di tesi. L'argomento presenta un trigger di reindirizzamento, che si triggera se viene richiamato esplicitamente da un altro argomento. Gli argomenti che lo richiamano sono:

- Start Conversation, che lo richiama all'Inizio della conversazione, quando il chatbot viene attivato;
- VerifyIfFinisced, che lo richiama nel caso in cui l'utente voglia eseguire più richieste;
- Fallback, che lo richiama nel caso in cui si esca dal meccanismo ciclico di richieste;

Una volta che tale argomento viene triggerato, mostra all'utente un messaggio in cui richiede come possa aiutarlo. Se la variabile che contiene la richiesta è vuota, allora, la richiesta dell'utente viene salvata in una variabile globale. Altrimenti questa interazione viene saltata. Il meccanismo di richiesta viene realizzato attraverso il blocco "Question" messo a disposizione da Copilot Studio.

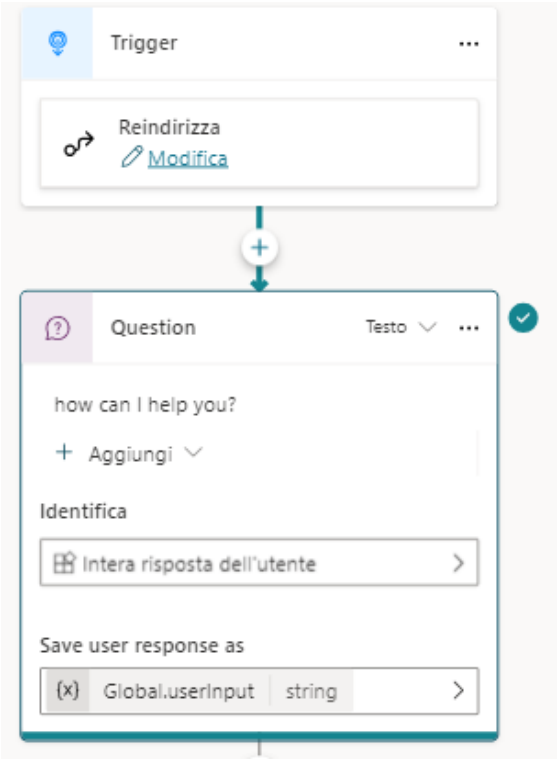

Figura 6.2: Argomento Post Azure 1

Copilot Studio offre strumenti avanzati, che non richiedono la necessità di codice di integrazione tramite flusso Power Automate. Ad esempio, per il nostro scopo, fornisce la possibilità di definire richieste HTTP e di gestirne le impostazioni, come ad esempio quella di timeout, direttamente nella piattaforma. La richiesta HTTP, mostrata sotto, è una richiesta Post verso l'endpoint del servizio. Il corpo

della richiesta viene popolato, usando il linguaggio powerFx, con l'input inserito dall'utente. Il sistema chatbot viene impostato in modo da aspettare fino ad un massimo di 3 minuti, dopodiché, viene segnalato un errore che scatena l'esecuzione dell'argomento di sistema FallBack. Se la risposta del server, in cui è ospitato il servizio, fornisce una risposta prima della fine del timeout, allora, la risposta viene salvata nella variabile indicata.

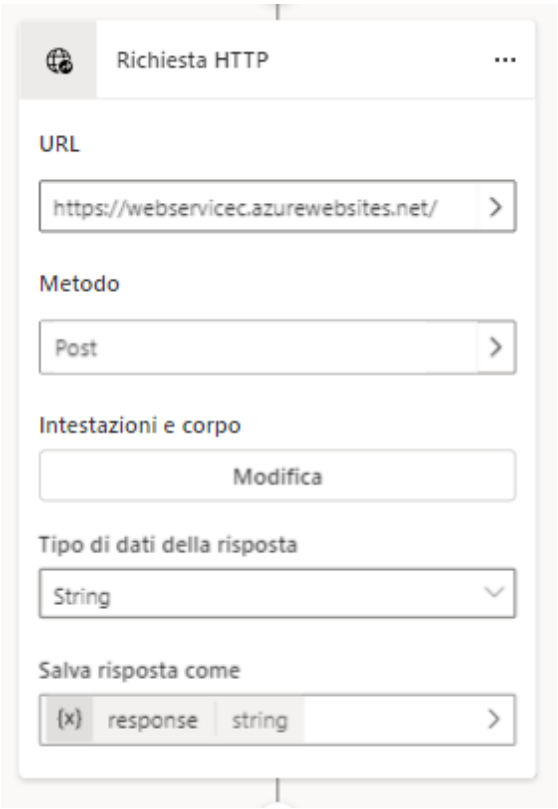

Figura 6.3: Argomento Post Azure 2

Successivamente, la risposta viene fornita all'utente tramite il blocco "Message". Dopodiché, avviene la cancellazione delle variabili impostate precedentemente al fine di poter ripetere l'operazione. Il flusso termina con la chiamata all'argomento verifyIfFinished, un argomento personalizzato che gestisce la logica di ripetizione nel caso in cui l'utente sia interessato a definire altre richieste.

verifyIfFinished

L'argomento verifyIfFinished si occupa di gestire la capacità del chatbot di ricontattare il servizio di traduzione, nel caso in cui l'utente voglia richiedere più informazioni sui dati del Dataverse. L'argomento viene triggerato in seguito a una

chiamata esplicita proveniente dall'argomento "Post Azure". Per prima cosa, viene posta una domanda, tramite il blocco "Question", all'utente riguardante la sua volontà di continuare con l'inserimento di altre richieste. Il sistema fornisce due possibilità di risposta cliccabili, che sono:

- End Conversation, opzione che indica la volontà dell'utente di terminare la conversazione;
- Continue, opzione che indica la volontà dell'utente di eseguire una ulteriore interrogazione;

La scelta viene salvata all'interno della variabile "choice". Si è progettata, inoltre, la possibilità di saltare questa interazione, permettendo all'utente, al momento della richiesta di continuare, di inserire direttamente la nuova richiesta.

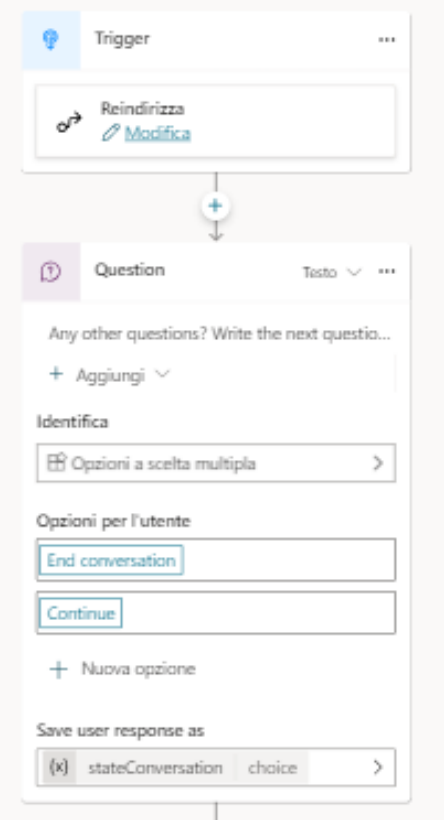

Figura 6.4: Argomento verifyIfFinished 1

Se il valore di "choice" è "End conversation" allora il flusso reindirizza l'utente alle azioni previste dall'argomento End Conversation, che si dettaglia più avanti. Altrimenti, se il valore è qualsiasi altro, imposto la variabile che definisce l'input

dell'utente all'ultimo messaggio inserito dall'utente. Se tale valore è "continue", allora, il sistema cancella il valore delle variabili in modo tale che venga rieseguita la richiesta di inserimento dell'interrogazione della "Post Azure". Se invece, il valore è qualsiasi altro viene triggerata direttamente la "Post Azure", che conterrà già la richiesta da passare al servizio. In questo modo la richiesta di inserimento dell'interrogazione viene saltata perchè già salvata.

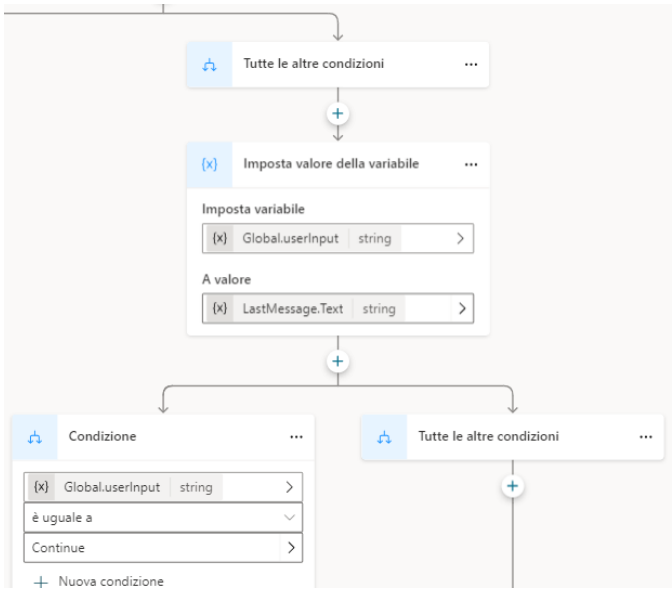

Figura 6.5: Argomento verifyIfFinished 2

Figura 6.6: Argomento verifyIfFinished 3

Fallback

L'argomento di Fallback viene eseguito quando l'input inserito non triggera nessun argomento definito nel sistema [21]. Nel nostro caso viene utilizzato come meccanismo per forzare il sistema a rientrare nel ciclo di richieste se, per un qualche motivo, ne si esce. Consiste esclusivamente di una chiamata all'argomento "Post Azure".

End Conversation

L'argomento "End Conversation" viene chiamato quando l'utente non ha più altro da chiedere e seleziona la sua volontà di terminare la conversazione. Viene eseguito al momento della scelta dell'opzione "End Conversation" quando il sistema richiede se si vuole continuare. Contiene una chiamata all'argomento "Start Conversation" per terminare la corrente conversazione. Da questo momento il chatbot rimarrà in attesa di ricevere un nuovo input. Quando, quindi, l'utente richiede la chiusura della conversazione non avviene la terminazione del chatbot, che rimane in attesa tornando allo stato iniziale.

On Error

L'argomento "On Error" viene utilizzato per gestire le casistiche di errore [22]. Una qualsiasi azione può scatenare un errore, che viene segnalato. Nel nostro caso, la richiesta HTTP in caso di timeout, lancia un evento di errore che viene catturato da questo argomento. L'argomento è personalizzabile ma non eliminabile in quanto

argomento di sistema. Tale argomento, quindi, definisce la logica da seguire nel momento in cui un errore viene segnalato. Nel nostro caso, al momento dello scadere del timeout, il chatbot indirizza l'utente all'argomento "verifyIfFinished" in modo tale che possa decidere se è il caso di ritentare l'interrogazione o terminare la conversazione.

6.4 Integrazione e Pubblicazione

Come già accennato, Copilot Studio è stato lo strumento scelto per implementare la logica del chatbot. Tale scelta viene motivata dalla facilità di integrazione che la piattaforma fornisce coi principali sistemi, che siano Microsoft o meno. Copilot Studio prevede la possibilità di integrare un chatbot su diversi canali, quali Skype, Cortana, Slack, siti web e tanti altri. Per prima cosa, lo sviluppatore deve occuparsi di pubblicare il chatbot, dopodiché, il chatbot può essere collegato a più canali. Ogni nuova modifica può essere propagata su tutti i canali, semplicemente, effettuando una nuova pubblicazione. Per coloro che stanno usando in quel momento l'applicazione, la propagazione dei cambiamenti avviene quando si inizia una nuova conversazione. Quando la pubblicazione viene eseguita, si può rendere l'applicazione disponibile ai membri del team o all'intera organizzazione [23]. La pubblicazione sul canale Teams permette di [24]:

- decidere come personalizzare l'aspetto del chatbot;
- installare il chatbot solo per te;
- condividere il collegamento di istallazione ad un sotto insieme di utenti;
- mostrare il chatbot a tutti gli utenti della tua organizzazione, una volta che sia stato approvato dall'amministratore;
- aggiungere il chatbot ad un canale Teams;

Capitolo 7

Conclusioni

Questo capitolo conclusivo definisce, dapprima, quali siano i risultati ottenuti rispetto agli obiettivi che la tesi si prefissava, e, successivamente, definisce i principali limiti ed i potenziali miglioramenti della proposta di tesi.

Innanzitutto, l'obiettivo generale della tesi prevede la progettazione di uno strumento di ricerca avanzata che permetta all'utente di realizzare interrogazioni col linguaggio naturale sui dati contenuti nel Dataverse associato ad un CRM. Microsoft si è posto lo stesso obiettivo e ,nel Marzo del 2023, ha rilasciato la sua soluzione, Copilot. Tale prodotto presentava alcune limitazioni. Innanzitutto, richiede una configurazione time-consuming a più livelli, di tabella e di colonna. Inoltre, Copilot prevedeva inizialmente la possibilità di interrogare dati contenuti nelle sole tabelle standard, cioè le tabelle offerte di default. Più nello specifico, quindi, la nostra tesi si propone di offrire lo strumento di ricerca descritto precedentemente che, a differenza di Copilot, si concentri nel risolvere le richieste puntuali, cioè che ritornano un numero limitato di record, e che superi i limiti di Copilot precedentemente descritti. L'obiettivo della proposta di tesi si ritiene raggiunto. L'utente è in grado di interrogare i dati di qualsiasi tabella, che sia custom o standard, senza la necessità di effettuare alcuna configurazione.

Al termine dell'implementazione sono stati effettuati una serie di test che hanno mostrato le potenzialità della soluzione nel raggiungere l'obiettivo proposto. La scelta dei dati da interrogare è avvenuta congiuntamente con l'organizzazione Reply Cluster, che ha definito quali fossero i più interessanti del modello dati. Inoltre, l'analisi sui risultati dei test ha permesso di evidenziare le principali limitazioni del progetto di tesi. Il primo limite è relativo alla presenza di attributi semanticamente vicini al contenuto di altre tabelle. In questo caso, lo strumento di ricerca non è in grado di selezionare la corretta tabella e la ricerca termina. Un ulteriore limite è l'incapacità di eseguire operazioni sugli attributi di lookup.

La tesi propone delle possibili soluzioni ai precedenti limiti al fine di introdurre un miglioramento nella soluzione. Il primo limite, quello legato alla scelta sbagliata

della tabella, è difficilmente risolvibile. Il problema potrebbe essere mitigato con l'aggiunta di informazioni specifiche nella descrizione della tabella che suggeriscano quando è il caso di selezionare una tabella rispetto ad un'altra. Per quanto riguarda il limite sulle operazioni con gli attributi di lookup, si è previsto un meccanismo che traduce la condizione su tali attributi in un blocco. Tale blocco è composto da una relazione con la tabella in cui è presente l'attributo a cui punta la lookup e da una condizione espressa su tale attributo puntato.

La soluzione qui proposta sarà utilizzata come base per future evoluzioni all'interno di Cluster Reply.

Il suo sviluppo è stato formativo e pone le basi per la creazione di nuove soluzioni all'interno del contesto aziendale. Le abilità acquisite in questo contesto sulla PowerPlatform e sui meccanismi per interfacciarsi con un modello AI, saranno sicuramente utili e spendibili all'interno dell'azienda e, in generale, nel mondo del lavoro.

Bibliografia

[1] Microsoft. *Descrizione del valore aziendale di Power Platform*. URL: https://learn.microsoft.com/it-it/training/modules/introduction-power-platform/3-describe-business-value-power-platform (cit. alle pp. 7, 13, 15, 23).

[2] Dushyanth Chandramouli. *Storie Clienti*. Technology Director, PwC. Ott. 2011 (cit. a p. 10).

[3] Jeff Toler. *Storie Clienti*. Senior Manager, Asset Protection Implementation and Support Solutions, Walgreens Boots Alliance. Feb. 2021 (cit. a p. 10).

[4] Microsoft. *Perché lo sviluppo con poco codice è importante*. Dic. 2023. URL: https://powerapps.microsoft.com/it-it/what-is-low-code/ (cit. a p. 11).

[5] Microsoft. *Che cos'è una piattaforma di sviluppo low-code?* Dic. 2023. URL: https://powerapps.microsoft.com/it-it/low-code-platform/ (cit. a p. 12).

[6] Giorgio Fusari. *Low Code: cos'è, quali vantaggi apporta alle aziende*. Mar. 2023. URL: https://www.zerounoweb.it/software/cose-lo-sviluppo-low-code-e-quali-sono-le-piattaforme-disponibili/ (cit. a p. 12).

[7] Microsoft. *What is Power Apps?* Mar. 2023. URL: https://learn.microsoft.com/en-us/power-apps/powerapps-overview (cit. a p. 13).

[8] Shish Singh. *Canvas Apps vs. Model-Driven Apps: Making the Right Choice*. Set. 2023. URL: https://dev.to/shishsingh/canvas-apps-vs-model-driven-apps-making-the-right-choice-2834 (cit. a p. 14).

[9] GitikaG Iaan D'Souza-Wiltshire Winona Azure. *Panoramica su Microsoft Copilot Studio*. Dic. 2023. URL: https://learn.microsoft.com/it-it/microsoft-copilot-studio/fundamentals-what-is-copilot-studio (cit. a p. 19).

[10] Olprod M. Mercuri M. O'Rourke. *Perché scegliere Microsoft Dataverse?* Apr. 2023. URL: https://learn.microsoft.com/it-it/power-apps/maker/data-platform/why-dataverse-overview (cit. a p. 22).

[11] Microsoft. *Che cos'è il Servizio OpenAI di Azure?* Feb. 2024. URL: https://learn.microsoft.com/it-it/azure/ai-services/openai/overview (cit. a p. 28).

[12] Microsoft. *What is CRM?* URL: https://dynamics.microsoft.com/en-us/crm/what-is-crm/ (cit. a p. 29).

[13] IBM. *Che cos'è LangChain?* URL: https://www.ibm.com/it-it/topics/langchain (cit. a p. 39).

[14] Iryna Kondrashchenko. *First Steps in LangChain: The Ultimate Guide for Beginners*. Giu. 2023. URL: https://medium.com/@iryna230520/first-steps-in-langchain-the-ultimate-guide-for-beginners-part-1-2baf5a4e1b81 (cit. a p. 39).

[15] Amazon. *Che cos'è LangChain?* URL: https://aws.amazon.com/it/what-is/langchain/ (cit. a p. 42).

[16] Danny Rodriguez. *FetchXML, an overview of this essential syntax*. Apr. 2020. URL: https://dynamics-chronicles.com/article/fetchxml-overview-essential-syntax (cit. a p. 44).

[17] Microsoft. *Cos'è e come funziona Copilot*. Mar. 2024. URL: https://www.aranzulla.it/cose-e-come-funziona-copilot-1578021.html (cit. a p. 68).

[18] Microsoft. *Copilot per Microsoft 365*. Gen. 2024. URL: https://learn.microsoft.com/it-it/office365/servicedescriptions/office-365-platform-service-description/microsoft-365-copilot (cit. a p. 68).

[19] Microsoft. *Copilot for Dynamics 365*. Nov. 2023. URL: https://learn.microsoft.com/en-us/microsoft-cloud/dev/copilot/copilot-for-dynamics365 (cit. a p. 68).

[20] Microsoft. *Introducing Microsoft Dynamics 365 Copilot, the world's first copilot in both CRM and ERP, that brings next-generation AI to every line of business*. Mar. 2023. URL: https://blogs.microsoft.com/blog/2023/03/06/introducing-microsoft-dynamics-365-copilot/ (cit. a p. 68).

[21] Microsoft. *Uso dell'argomento di fallback*. Nov. 2023. URL: https://learn.microsoft.com/it-it/microsoft-copilot-studio/guidance/fallback-topic (cit. a p. 81).

[22] Microsoft. *Codici di errore: Microsoft Copilot Studio*. Nov. 2023. URL: https://learn.microsoft.com/it-it/microsoft-copilot-studio/error-codes?tabs=webApp (cit. a p. 81).

[23] Microsoft. *Concetti chiave: pubblicare il tuo bot*. Dic. 2023. URL: https://learn.microsoft.com/it-it/microsoft-copilot-studio/publication-fundamentals-publish-channels?tabs=teams#publish-the-latest-content (cit. a p. 82).

[24] Microsoft. *Aggiungere un chatbot a Microsoft Teams*. Mag. 2023. URL: https://learn.microsoft.com/it-it/microsoft-copilot-studio/publication-add-bot-to-microsoft-teams (cit. a p. 82).