Progetto di gruppo Assistente Sanitario LLM

Richard Berardi, Federico Capoferri, Gabriele Cipriani
2057533, 2045199, 2047776
Università La Sapienza di Roma

Abstract

Questo documento descrive l'architettura e l'implementazione di un assistente sanitario virtuale basato su Large Language Model (LLM). Il sistema permette l'interazione tra pazienti, medici e amministratori attraverso un'interfaccia web moderna, gestendo prenotazioni, consultazioni e analisi dei sintomi.

1 Divisione delle cartelle

Le tre cartelle principali del progetto sono:

• 2 Approfondimento delle cartelle

2.1 Backend

Nel backend troveremo il `Dockerfile` e la cartella `src` quest'ultima a sua volta divisa in:

- `data`, dove ci vanno a finire le mappe [ancora da implementare]
- `databse`, ci sono i file python che interagiscono con il nostro database
- `models`, alcuni modelli pydantic per la gestione della chat con LLM

Successivamente nella sottocartella `backend` abbiamo:

- `llm_interaction`, sono presenti i file che gestiscono tutto il flow di lavoro con LLM
- `login_admin`, tutte le funzioni i metodi per la gestione del login degli admin e accettazzione dei medici
- `login_back_utenti`, tutte le funzioni e i metodi per la gestione dell'iscrzione e del login degli utenti
- `login_iscrizione_medici`, tutte le funzioni e i metodi per la gestione dell'iscrizione, login, ges- tione agenda dei medici, 'e presente anche una cartella dove vengono salvati i tesserini caricati dai medici
- `server.py`, il file effettivamente lanciato da uvicorn in cui ci sono tutti metodi FAST API, abbiamo usato un file unico per centralizzare il tutto

2.2 Frontend

Nel frontend è presente una cartella `src` che contiene:

- `login_front`, sono presenti i file che gestiscono la comunicazione con il backend per quanto riguarda gli utenti, e contiene anche tutti i file .html e .css delle pagine degli utenti
- `login_medici_front`, sono presenti i file che gestiscono la comunicazione con il backend per quanto riguarda i medici, e contiene anche tutti i file .html e .css delle pagine dei medici
- `login_admin_front`, sono presenti i file che gestiscono la comunicazione con il backend per quanto riguarda gli admin , e contiene anche tutti i file .html e .css delle pagine degli admin
- `frontend.py`, il file effettivamente lanciato da uvicorn in cui ci sono tutti metodi FAST API, abbiamo usato un file unico per centralizzare il tutto

3 Database

Tutte le procedure di inserimento vengono gestite **internamente al database** mediante l'uso di apposite **stored procedure** che permettono una maggiore pulizia ed ordine logico

del codice. Inoltre l'eliminazione dei clienti e dei medici viene anch'essa gestita da **stored procedure** apposite che semplicemente ne cambiano il **flag di stato**, in modo da "oscurarli".

Questa scelta è stata presa al fine di gestire correttamente le informazioni del database in modo che risultino **sempre presenti** ma con diversi livello di leggibilità per il sistema e gli utenti finali.

- `Admin` Contiene i dati per l'autenticazione degli amministratori.
  - **id** – chiave primaria
  - **email** – unica e non nulla
  - **password** – hash sicuro
- `Cliente` Informazioni personali dei pazienti.
  - **id, nome, cognome, email** – identificativi
  - **eta, sesso, peso, altezza**
  - **stato** – enum: Attivo, Eliminato
  - Vincoli: email unica, check su eta, sesso
- `IntolleranzaAlimentare` Associa clienti a intolleranze specifiche.
  - **id, id cliente, intolleranza, stato**
- `CondizioniPatologichePregresse` Patologie passate dichiarate dai clienti.
  - **id, id cliente, condizione_preg, stato**
- `CondizioniPatologicheFamiliari` Patologie presenti nella famiglia del cliente.
  - **id, id cliente, condizione_fam, stato**
- `Medico` Dati anagrafici e professionali del medico.
  - **id, nome, cognome, email, telefono**
  - **codice_fiscale, numero_albo, citta_ordine, url_sito, stato, verificato**
- `Specializzazione` Specializzazioni mediche con geolocalizzazione.
  - **id, id_medico, specializzazione, ranking, stato, indirizzo latitudine, longitudine**
- `Agenda` Appuntamenti prenotati per ciascun medico.
  - **id, id_medico, appuntamento, id_cliente, stato**
- `Appuntamento` Dettagli della visita tra medico e cliente.
  - **id, id_cliente, id_medico, data_registrazione, data_appuntamento patologia_individuata**
  - **stato**: Prenotato, Effettuato, Eliminato
  - Vincoli: patologia obbligatoria solo se Effettuato
- `RankingAppuntamento` Valutazione del servizio medico da parte del cliente.
  - **id, id_appuntamento, id_medico, voto, commento, data_ranking, stato**
  - Un ranking per ogni appuntamento
- `Chat` Collegamento tra cliente e messaggistica.
  - **id, id_cliente, stato**

Figure 1: Struttura database

4 Interazione con l'LLM

L'interazione è pensata su un sistema di fasi successive, per cui l'llm può trovarsi a dover rispondere a domande del tipo *'analisi sintomi'*, *'ricerca degli specialisti'*, *'prenotare una visita'*. Data la variabilità di una conversazione naturale, l'obiettivo (non ancora raggiunto) sarebbe quello di permettere al sistema di non dover seguire l'ordine logico in maniera rigorosa, ma di essere flessibile sulle richieste. Per tale obiettivo il funzionamento è il seguente:

1. dal frontend un utente potrà scegliere se aprire una chat già avviata o iniziarne una nuova

2. scegliendo di iniziare una nuova chat si inizializza una conversazione, ovvero si istruisce un oggetto, modellato con una classe, VirtualAssistant, che rappresenta il nostro assistente, descrivendogli quale è il suo ruolo nel servizio e quali sono i compiti a cui risponde

3. dal frontend viene preso un messaggio e mandato a un endpoint 'llm_interact' che gestisce l'intero processamento, restituendo la risposta

4. a ogni messaggio l'oggetto viene ricreato con passaggio della storia fino a quel momento, presa dal database

5. ogni messaggio viene prima classificato, dallo stesso assistente; la scelta di non usare un classificatore separato è stata fatta per aiutare la classificazione con le informazioni della storia della chat, cosa che congetturiamo si possa vedere meglio con un modello più potente

6. il messaggio classificato viene scartato dalla storia della chat in quanto non riguarda l'interazione con l'utente

7. dopo la classificazione il messaggio viene passato ad un prompt che wrappa il messaggio, fornendo indicazioni su cosa fare e passando dati opportuni da cui attingere per la risposta: per esemipo, se la domanda riguarda la volontà di fare una prenotazione, ci sarà un prompt che circonda la domanda, arricchendola di indicazioni per l'assistente e dati presi dal database specifici per la fase in cui si trova, in questo caso una lista di medici con cui poter prenotare

8. correntemente l'assistente segue lo schema rappresentato nell'immagine 'Interazione Utente LLM' solo dall'alto al basso, mentre può crashare se si segue un ordine scorretto (il crash è voluto per ragioni di debug, così da intervenire più visibilmente nello sviluppo successivo)

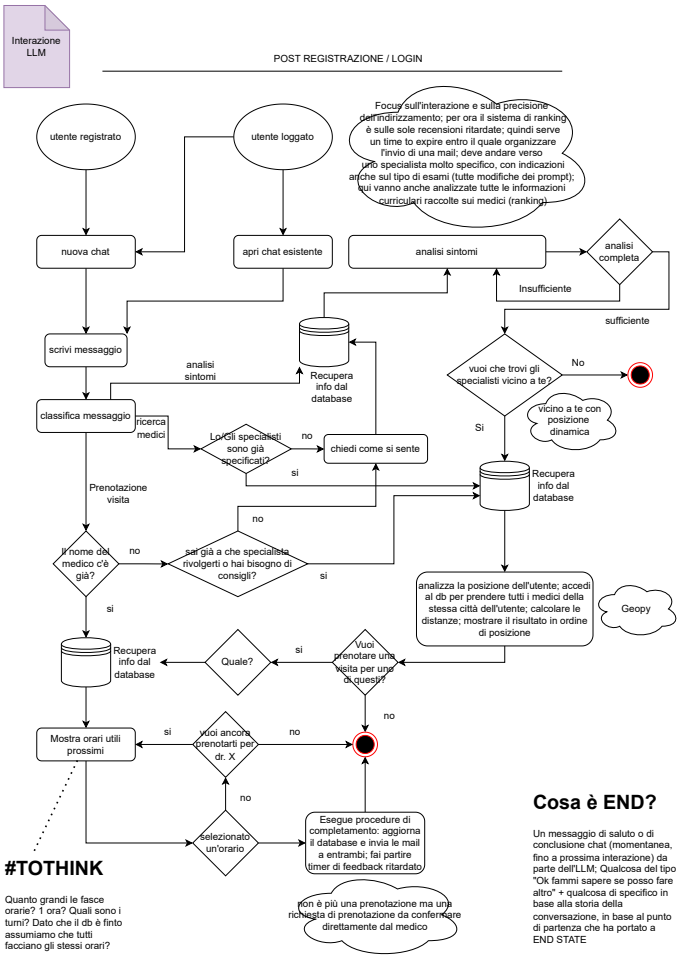

Figure 2: Interazione Utente-LLM

5 Implementazioni successive

All'interno dei vari file presenti nel progetto vi sono le implementazio di feature ulteriori che permettono al progetto di risultare più gradevole all'uso per l'utente e più professionale in generale.

Lato Database è già implementato un sistema di recensione mediante voto e commento degli appuntamenti correttamente terminati, e quindi con diagnosi effettuata, che permette poi la stima di un punteggio medio dei vari medici nelle loro specializzazione d'interesse. Ciò ci permetterebbe di aggiungere un cirterio di scelte ulteriore alla mera distanza minima cliente-medico.

Inoltre risulta già creata l'apposita tabella `Agenda` necessaria per gestire e render

disponibili i vari slot orari per gli appuntamenti del singolo medico. La sua implementazione andrebbe gestita come per le altre da una stored procedure ad hoc e la modifica di quella che effettua le prenotazioni dei clienti.

Lato Frontend l'implementazione di una mappa interattiva, mediante un apposita libreria python, che permette a colpo d'occio di avere informazioni sulle varie soluzioni proposte ed il loro collocamento nella città dell'utente.