Progetto Finale di Laboratorio di Ingegneria Informatica

Alessandro Caruso 1944592 **Gabriele Di Perna 1867288**

1 Introduzione

Il progetto ha come obiettivo l'implementazione di un sistema web-based per l'interrogazione di database tramite Large Language Models (LLM). La piattaforma consente sia l'aggiunta di nuove informazioni sia la consultazione dei dati già presenti, supportando sia query SQL dirette sia richieste in linguaggio naturale.

2 Organizzazione del database

Il database implementato si chiama ***moviesdb*** ed è strutturato per gestire informazioni sul mondo del cinema. Lo schema comprende quattro tabelle principali:

- ***regista***
  Contiene informazioni relative ai registi, con chiave primaria *idR*
  Attributi: nome(univoco),età

- ***piattaforma***
  Raccoglie le piattaforme di distribuzione o streaming, con chiave primaria *idP*
  Attributi: nome(univoco)

- ***movies (film)***
  Contiene informazioni relative ai film, con chiave primaria *idF*
  Attributi: titolo/name(univoco),anno,genere.
  Contiene la *foreign key* (movies.idR -> regista.idR) che la mette in relazione con la tabella *regista*

- ***dove vederlo***
  Specifica la disponibilità di ciascun film sulle piattaforme, ha come chiave primaria *idF* (collegata a *movies.idF*). Può contenere fino a due piattaforme per film (*idP1,idP2*) tramite *foreign key* verso *piattaforma.idP*. La presenza di un vincolo *check* impedisce alle due piattaforme di coincidere.

3 Organizzazione del Codice

Il progetto è stato sviluppato seguendo un'architettura modulare che separa chiaramente la logica applicativa dal livello di accesso ai dati e dalla definizione degli endpoint.

3.1 Backend

Il backend è organizzato nei seguenti moduli principali:

- `config.py`: definizione dei parametri di configurazione (DB, porte, Ollama).

- `db.py`: gestione della connessione al database MariaDB.

- `models.py`: modelli Pydantic per validazione degli input e degli output.

- `seed.py`: popolamento dinamico iniziale del database da file TSV.

- `logic/`: implementazione della logica per le diverse funzionalità (`add`, `schema`, `search`, `text_to_sql`).

- `api/`: definizione degli endpoint FastAPI, ciascuno dei quali richiama la corrispondente funzione in `logic/`.

- `main.py`: entrypoint dell'applicazione, registra i router e avvia il server FastAPI.

Questa suddivisione garantisce una chiara separazione delle responsabilità e rende il sistema facilmente manutenibile ed estendibile.

3.2 Frontend

Il frontend, realizzato con FastAPI e Jinja2, comprende:

- `routes.py`: routing e interazione con il backend.

- `templates/`: template HTML (es. `base.html`, `index.html`, `schema.html`) per la visualizzazione dei dati.
- `static/style.css`: foglio di stile per l'interfaccia.

L'interfaccia è progettata come una chat interattiva: l'utente può inviare domande in linguaggio naturale o query SQL, ricevendo i risultati in formato tabellare.

3.3 Qualità del codice

Oltre alla suddivisione in moduli, il codice adotta soluzine che ne migliorano leggibilità,manutenzione e robustezza:

- **Leggibilità e pulizia**: il codice utilizza *type hints*, docstring e commenti esplicativi, con funzioni brevi e nomi significativi. Ciò favorisce la leggibilità e la manutenzione.
- **Gestione dei casi limite**: sono stati previsti controlli per input non validi, query SQL non sicure o mal formate, ed eventuali errori di connessione al database o al modello LLM.
- **Robustezza**: la validazione tramite Pydantic e la gestione esplicita delle eccezioni permettono di ridurre al minimo gli errori a runtime.

4 Endpoint implementati

Il backend espone diversi endpoint REST, realizzati con FastAPI, che costituiscono i punti di accesso principali al sistema. Ogni endpoint è definito nel pacchetto `api/` e delega la logica di business al corrispondente modulo in `logic/`, secondo un approccio che separa chiaramente routing e logica applicativa. L'**entrypoint** dell'applicazione è `main.py`, che crea l'istanza FastAPI e registra tutti i router provenienti dai moduli di `api/`. In questo modo il codice rimane compatto, con un unico punto di avvio, mentre la definizione delle singole funzionalità è demandata ai file specifici.
Gli endpoint implementati coprono le seguenti operazioni:

- **/db_health**: verifica la connessione al database e restituisce lo stato.
- **/schema_summary**: fornisce un riassunto dello schema del database, utile anche per la generazione dei prompt LLM.
- **/add**: consente di inserire o aggiornare film, registi e piattaforme a partire da una stringa normalizzata.
- **/sql_search**: permette l'esecuzione di query SQL dirette, distinguendo tra query valide, non valide e non sicure.
- **/search**: implementa la funzionalità di Text-to-SQL, traducendo domande in linguaggio naturale in query SQL tramite il modello LLM.

Questa organizzazione rende il sistema estendibile: per aggiungere una nuova funzionalità è sufficiente implementare un modulo in `logic/` e il relativo router in `api/`, senza apportare modifiche alla struttura esistente.

5 Funzionalità del frontend

Il frontend offre un'unica interfaccia in stile chat attraverso la quale l'utente può interrogare lo schema del database, aggiungere o aggiornare dati, inviare query SQL dirette oppure domande in linguaggio naturale. Ogni richiesta genera una "bolla" che mostra la query SQL eseguita, l'esito della validazione (valid, invalid, unsafe) e, se presenti, i risultati tabellari.

6 Gestione degli errori e validazione

Ogni endpoint prevede controlli di validità sugli input e gestione esplicita delle eccezioni.
Le query SQL vengono considerate *valid* se eseguite correttamente, *invalid* se l'esecuzione produce errori e *unsafe* se non iniziano con `SELECT`. I dati inseriti tramite `/add` sono soggetti a parsing e normalizzazione con segnalazione immediata di errori. L'uso di modelli Pydantic garantisce la validazione automatica delle richieste e la produzione di risposte JSON sempre coerenti, permettendo al sistema di gestire in maniera robusta i casi limite senza interrompere il servizio.

7 Testing e consegna

Il sistema è stato verificato utilizzando lo script ufficiale `test_backend_progetto_finale.py`, fornito per la prova finale. I test sono statieseguiti in ambiente Docker su Ubuntu 22.04.5. L'applicazione ha superato con successo tutte le verifiche previste, producendo in output *ALL TESTS PASS*.