Progetto Finale di Laboratorio di Ingegneria Informatica

Alessandro Caruso 1944592 **Gabriele Di Perna 1867288**

1 Introduzione

Il progetto ha come obiettivo principale l'implementazione di un sistema web-based per l'interrogazione di database tramite Linguaggio naturale usando una forma basilare di Natural Language Processing. L'applicazione permette inoltre l'aggiunta di nuovi dati nel DataBase attraverso l'inserimento di una stringa formattata in modo specifico (simil CSV) e di effettuare query direttamente in SQL.

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:

- ***regista***
  Contiene informazioni relative ai registi,
  Attributi: chiave primaria *idR*,nome(chiave),età
- ***piattaforma***
  Raccoglie le piattaforme di distribuzione o streaming
  Attributi: ,chiave primaria *idP*, nome(chiave)
- ***movies (film)***
  Contiene informazioni relative ai film
  *Attributi:*chiave primaria *idF*, titolo/name(chiave),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

La tech stack utilizzata è la seguente:

- backend: FastAPI
- frontend: HTML, CSS e Jinja2
- db: MariaDB
- llm: Ollama (il modello utilizzato per le richieste è gemma3:1b-it-qat)

Il tipo di architettura modulare scelta per la codebase prevede una divisione netta tra frontend e backend. Per la parte del backend i moduli core che effettuano la inizializzazione della applicazione sono in /app. All'interno della stessa poi vi sono anche le cartelle /api e /logic, che contengono rispettivamente le definizioni degli endpoint, e la implementazione delle funzioni per la logica applicativa. Per il frontend il principio è molto simile, con la differenza che qui si divide invece il codice tra /static e /templates.

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 Funzionalità e altre specifiche sul 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.

Il frontend fornisce un'interfaccia web in stile chat, ispirata a sistemi come ChatGPT, realizzata con template e componenti dinamici che gestiscono l'invio delle richieste agli endpoint REST e la visualizzazione dei risultati sotto forma di modali o "bolle" con query, validazione ed eventuali dati tabellari.

4 Endpoint implementati

Ogni endpoint è definito nel pacchetto `api/` e delega la logica di business al corrispondente modulo in `logic/`, secondo un approccio che separa routing e logica applicativa. 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.

5 Gestione degli errori e validazione

Il codice integra controlli di validità sugli input e gestione esplicita delle eccezioni, supportati da type hinting e dalla validazione automatica offerta da Pydantic. Le interrogazione, indipendentemente dalla modalità, vengono considerate *valid* se eseguite correttamente, *invalid* se l'esecuzione produce errori e classificate *unsafe* quando non si tratta di una query di sola lettura. I dati inseriti tramite `/add` sono soggetti a parsing e normalizzazione con eventuale segnalazione di errori.

6 Dockerizzazione

Il progetto è stato containerizzato per garantire portabilità e facilità di esecuzione. Sono stati definiti `Dockerfile` dedicati per backend e frontend, mentre per MariaDB e Ollama vengono utilizzate immagini ufficiali preesistenti, orchestrate tramite `docker-compose`. In questo modo l'intero sistema può essere avviato con un singolo comando

```
docker compose up --build
```

senza configurazioni manuali aggiuntive.

Nota I : *Al primo utilizzo di* `search` *l'esecuzione risulta più lenta, in quanto Ollama scarica il modello da impiegare.*

Nota II : *Quando il sistema restituisce i risultati, il campo* `item_type` *è fissato a* `film` *per garantire il superamento dei test automatici; nel codice è comunque presente una versione dinamica, al momento commentata, per la classificazione degli* `item_type`.

Nota III : *Nella codebase è stata adottata la seguente convenzione: commenti in italiano (al fine di facilitarne la correzione), funzioni e messaggi di log in inglese.*