# Esonero Laboratorio di Ingegneria Informatica

**Valerio Massimo Dessena**
Università La Sapienza / Roma
dessena.2045709@studenti.uniroma1.it

## 1 Obiettivo del progetto

Il progetto consiste in un'applicazione Web che consente agli utenti di interrogare un database con delle query in linguaggio naturale attraverso un'interfaccia grafica.
L'applicazione è strutturata in microservizi.
Gli obiettivi che si pone il progetto sono quelli di implementare una architettura client-server attraverso protocollo HTTP (usando architettura REST), di containerizzare ogni microservizio attraverso Docker e di farli comunicare tra loro.
In questo modo si ottiene un'applicazione modulare e scalabile.

## 2 Organizzazione del database

Viene utilizzato il database `esonero_db`, che contiene informazioni su film e registi. Tale database è costituito da una tabella di nome `movies`.

| Nome Colonna | Tipo di Dato |
|---|---|
| titolo | VARCHAR(255) PRIM. KEY |
| regista | VARCHAR(255) NOT NULL |
| eta_autore | INT NOT NULL |
| anno | INT NOT NULL |
| genere | VARCHAR(255) NOT NULL |
| piattaforma_1 | VARCHAR(255) |
| piattaforma_2 | VARCHAR(255) |

Si osservi che non possono esistere due film con lo stesso titolo, dunque un identificatore per la tabella è la colonna `titolo`, che è stato scelto come chiave primaria.

## 3 Organizzazione del codice

Il progetto è strutturato in tre microservizi/moduli gestiti da tre container Docker contenenti rispettivamente: server di backend, server di frontend, database.

La comunicazione tra i microservizi avviene in questo modo:

1. L'utente interagisce con il server di frontend effettuando una richiesta HTTP su un URL specifico.
2. Il server di frontend riceve tale richiesta e la inoltra al server di backend.
3. Il server di backend, sulla base della richiesta, interagisce con il database e fornisce una risposta HTTP indietro al server frontend.
4. Questo a sua volta fa la stessa cosa verso l'utente renderizzando nel browser il risultato su un template HTML.

I due server sono scritti in Python e implementano il protocollo REST. Per farlo, fanno ricorso alle librerie Python: Fast API, Uvicorn, Pydantic.
Entrambi usano Uvicorn per hostare il proprio servizio su una specifica porta.
Il server backend usa Pydantic per modellizzare i dati inviati nelle risposte HTTP. Il database è costruito su un immagine di mariadb.

### 3.1 Struttura dei file del progetto

Il codice del server backend si trova nel file `backend.py` e nel pacchetto `utils` dove troviamo funzioni utili usate dalle funzioni handler degli endpoint HTTP.
Il codice del server frontend si trova in `frontend.py`.
Sono presenti quattro template HTML per ospitare il risultato delle funzionalità dell'applicazione.
I container dei due server si basano sulle immagini definite nei relativi `Dockerfile`.
Il file `docker-compose.yaml` permette l'orchestrazione dei container dell'intero progetto. Contiene anche le informazioni di configurazione per il container del database (per il quale non è

presente un apposito `Dockerfile`).
Il database viene creato nella cartella `mariadb_data/` e inizializzato dallo script `init.sql` che crea la tabella e inserisce le righe.

#### 3.1.1 Server backend

Il server backend utilizza la libreria Python mariadb per interagire con il database.
Presenta tre endpoint HTTP:

- `/search/{natural_lang_query}` [GET]:
  converte il contenuto della richiesta (in linguaggio naturale) in una query in linguaggio SQL, interroga il database di conseguenza e restituisce i risultati ottenuti.
- `/schema_summary` [GET]:
  interroga il database e restuituisce come output le informazioni che descrivono lo schema logico del database (nomi delle tabelle e delle rispettive colonne).
- `/add` [POST]:
  Riceve come payload della richiesta una riga da inserire nel database sotto forma di stringa ed attraverso un'instruzione SQL ne fa l'inserimento in una tabella.

  *Nota: Se si prova ad inserire una nuova riga, nel caso sia già presente una riga che ha lo stesso valore della nuova nei campi della primary key, viene effettuato un update e non un nuovo inserimento.*

In questa fase del progetto, la conversione è implementata semplicemente come una mappatura predeterminata di stringhe specifiche nelle relative query SQL. Nel codice questo viene gestito attraverso delle regex e un dizionario.

#### 3.1.2 Server frontend

Il server frontend fornisce un'interfaccia grafica all'utente per permettergli di interagire facilmente con l'applicazione. L'interfaccia grafica è implementata attraverso dei template in linguaggio HTML resi dinamici grazie all'uso della libreria Python Jinja2.
Il server presenta tre endpoint HTTP:

- `/` [GET]:
  entry point del sito web.
- `/query` [GET]:
  fa richiesta HTTP sull'URL `/search/{natural_lang_query}` del server backend, inserendo in `natural_lang_query` il contenuto di una specifica casella di testo nella pagina HTML.
- `/schema` [GET]:
  fa richiesta HTTP sull'URL `/schema_summary` del server backend
- `/add_data` [POST]:
  fa richiesta HTTP sull'URL `/add` del server backend, inviando come payload il contenuto di una specifica casella di testo nella pagina HTML.

#### 3.1.3 Docker

L'orchestrazione dei tre container Docker viene gestita con il codice presente nel file `docker-compose.yaml`.
Vengono creati i container `db`, `backend`, `frontend` sui quali vengono esposte rispettivamente le porte interne 3306, 8003, 8001.
Per un corretto funzionamento dell'applicazione i container devono essere eseguiti nell'ordine: `db`, `backend`, `frontend`. Perciò in `docker-compose.yaml` vengono inseriti i parametri di `healthcheck` e `depends_on`.

#### 3.1.4 Gestione degli Errori

Nel caso il formato del contenuto delle richieste HTTP sia invalido, il server di backend ritorna `Error 422`.

## 4 Esecuzione

Per eseguire l'applicazione eseguire, nella cartella root del progetto, il comando `docker compose up --build` e collegarsi tramite browser al server frontend all'URL `127.0.0.1:8001/`.

*Nota: Affinché funzionino gli healthcheck è stata usata l'immagine mariadb:11.7.2-ubi9. Se si modifica la struttura del database e si vuole testare di nuovo il progetto: estrarre nuovamente l'intero progetto dal file .zip. Altrimenti si può eliminare il contenuto della cartella `mariadb_data/` con `sudo rm -rf mariadb_data/*`, ora però per ricreare il database da zero, prima di lanciare `docker compose up --build`, è necessario fornire i permessi alla cartella `mariadb_data/` con `sudo chmod 0777 mariadb_data/`.*