UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Magistrale in Informatica

BASI DI DATI II

"CinePlus"

Docenti:
Genoveffa TORTORA
Luigi DI BIASI

Studenti:
Rita CUCCARO **Mat. 0522501864**
Marta COIRO **Mat. 0522501611**
Katia BUONOCORE **Mat. 0522501744**

ANNO ACCADEMICO 2023/2024

Indice

1 Introduzione

1.1 Descrizione e obiettivo del progetto

CinePlus si propone di diventare una piattaforma cinematografica dedicata alla visualizzazione in streaming di Film e SerieTV. L'obiettivo è quello di soddisfare i vari gusti cinematografici, permettendo a ciascun utente di accedere ad un vasto catalogo di Film e SerieTV da poter guardare in streaming senza vincoli di tempo. L'obiettivo è quello di coinvolgere sia gli utenti che utilizzano un pc ma anche altri dispositivi. Il sito web è strutturato in modo responsive.

2 Architettura del sistema

2.1 Frontend: HTML, CSS e Bootstrap per la struttura e lo stile dell'interfaccia utente

Per il front-end di "CinePlus", abbiamo utilizzato HTML, CSS e Bootstrap per creare un'interfaccia utente solida e responsiva.

HTML ha fornito la struttura fondamentale dell'applicazione. Utilizzando HTML5, abbiamo costruito pagine ben organizzate con elementi semantici che migliorano l'accessibilità e la SEO, facilitando la manutenzione e l'aggiornamento del sito.

CSS è stato impiegato per definire l'aspetto visivo dell'applicazione. Con un file CSS personalizzato, abbiamo curato dettagli come colori, font, margini e padding per ottenere un design pulito e moderno. Questo ha permesso di rendere l'interfaccia visivamente coerente e piacevole da usare.

Bootstrap è stato utilizzato per garantire la responsività del design. Grazie a questo framework, abbiamo implementato layout e componenti che si adattano automaticamente a diverse dimensioni di schermo, assicurando un'esperienza utente ottimale su dispositivi mobili e desktop.

L'uso, quindi, combinato di HTML, CSS e Bootstrap ha permesso di realizzare una web app con una struttura chiara, uno stile elegante e una funzionalità reattiva, ottimizzando l'esperienza utente su qualsiasi dispositivo.

La web app CinePlus è progettata per offrire un'esperienza completa e intuitiva sia per gli utenti che per gli amministratori. All'accesso alla piattaforma, gli utenti vengono accolti dalla homepage, che presenta un elenco di film e serie TV disponibili. Questi contenuti sono visualizzati in modo accattivante attraverso delle card che mostrano l'immagine di copertina, il titolo, l'anno di rilascio e il genere. In cima alla homepage, si trova una sezione dedicata alla Top 10, che evidenzia i dieci film e serie TV con i punteggi più alti, consentendo agli utenti di scoprire facilmente i contenuti più apprezzati. Per gli utenti registrati, ogni card include un pulsante "Aggiungi ai Preferiti", permettendo di salvare i titoli preferiti per un accesso futuro più rapido.

Ogni film e serie TV nella homepage è dotato di un pulsante "View Details" (Visualizza Dettagli), che porta a una pagina con informazioni complete sul titolo selezionato. Questa pagina fornisce dettagli come il titolo, l'anno di

uscita, il punteggio su IMDb, una trama approfondita e tutto il cast. È presente anche un pulsante "View on IMDb" che rimanda a un link esterno su IMDb per ulteriori dettagli e recensioni.

Gli utenti possono esplorare l'intero catalogo di film e serie TV attraverso le pagine dedicate a ciascuno di essi. La pagina Film mostra un elenco completo di tutti i film disponibili, mentre la pagina Serie TV offre una panoramica di tutte le serie TV. Entrambe le pagine presentano le loro informazioni essenziali come nella Homepage.

Per una navigazione più mirata, la pagina Generi offre un elenco di pulsanti che rappresentano i vari generi disponibili nella web app. Cliccando su uno di questi pulsanti, l'utente viene reindirizzato a una pagina che mostra un elenco di film e serie TV appartenenti al genere selezionato, permettendo di esplorare contenuti specifici basati sui gusti personali.

La navbar, presente in tutte le pagine, è progettata per adattarsi alle esigenze degli utenti registrati e non registrati. Per gli utenti non registrati, la navbar include il link per il Login, una Barra di Ricerca per cercare film e serie TV per titolo, data, direttore/scrittore/attore, e i link per le pagine Film, Serie TV e Generi. Gli utenti registrati, invece, vedono la navbar arricchita con un link per il Logout, oltre alla Barra di Ricerca e ai link per le pagine di film e serie TV. Inoltre, hanno accesso a un menu a tendina che include opzioni per navigare tra i Generi, i Film da Rivedere e i Preferiti.

Una funzionalità importante per gli utenti registrati è la lista dei "Film da Rivedere". Quando un utente guarda un film o una serie TV, questo viene automaticamente aggiunto a una lista di film da rivedere. Questa funzionalità permette agli utenti di tenere traccia dei titoli che hanno già visto e che potrebbero voler rivedere in futuro, rendendo l'esperienza più personalizzata e organizzata.

Per quanto riguarda la gestione dei contenuti, gli amministratori hanno accesso a funzionalità speciali che permettono di mantenere la piattaforma aggiornata e ben organizzata. Gli amministratori possono aggiungere nuovi film e serie TV attraverso una pagina di inserimento prodotto, che include moduli per inserire i dettagli. La pagina di modifica prodotto consente di aggiornare le informazioni esistenti su film e serie TV, mentre la pagina di cancellazione prodotto permette di rimuovere contenuti non più rilevanti dal database. Infine, la funzione di ricerca è identica a quella degli utenti, facilitando la gestione dei contenuti.

Questa struttura garantisce un'esperienza utente fluida e personalizzata per tutti gli utenti di "CinePlus", mentre fornisce agli amministratori gli strumenti necessari per gestire efficacemente il catalogo dei contenuti.

2.1.1 Javascript per l'interattività lato client

Nel front-end di "CinePlus", JavaScript gioca un ruolo cruciale nel fornire un'esperienza utente interattiva e dinamica. Attraverso l'uso di JavaScript, siamo stati in grado di migliorare l'interazione dell'utente con la web app, rendendo l'applicazione più coinvolgente e reattiva.

- **Gestione delle Interazioni dell'Utente:** JavaScript è utilizzato per gestire e rispondere alle azioni dell'utente, come clic, hover e input. Ad esempio:
  - **Pulsante "Aggiungi ai Preferiti":** Per gli utenti registrati, ogni film o serie TV nella homepage ha un pulsante "Aggiungi ai Preferiti". JavaScript gestisce il click su questo pulsante, aggiornando in tempo reale l'interfaccia per riflettere lo stato di aggiunta ai preferiti e memorizzando questa preferenza nel backend.
  - **Dropdown e Menu:** Il menu a tendina nella navbar per gli utenti registrati è gestito da JavaScript, che controlla la visualizzazione dei sottomenu (come Generi, Film da Rivedere e Preferiti) quando l'utente interagisce con il menu principale.
- **Manipolazione del DOM:** JavaScript è essenziale per modificare e aggiornare dinamicamente il contenuto della pagina senza richiedere un ricaricamento completo. Alcuni esempi includono:
  - **Messaggi di Aggiunta ai Preferiti:** Quando un utente aggiunge un film o una serie TV ai preferiti, un messaggio temporaneo viene mostrato confermando l'azione. Questo messaggio scompare automaticamente dopo pochi secondi.
  - **Aggiunta alla Lista "Da Rivedere":** Quando un utente registrato guarda un film o una serie TV, questi vengono automaticamente aggiunti alla lista dei "Film da Rivedere". Questo processo viene gestito tramite una chiamata AJAX che aggiorna il database e riflette immediatamente le modifiche nell'interfaccia utente.
- **Validazione dei Moduli:** Per garantire che i dati inseriti dagli utenti siano corretti, JavaScript viene utilizzato per la validazione dei moduli:
  - **Modulo di Login e Registrazione:** JavaScript controlla che tutte le informazioni necessarie siano state fornite e che soddisfino i criteri richiesti (ad es. formato email valido, password sufficientemente sicura) prima di inviare i dati al server.
  - **Inserimento e Modifica di Prodotti:** Gli amministratori possono aggiungere o modificare film e serie TV attraverso moduli dedicati. JavaScript verifica che tutti i campi obbligatori siano completati e che i dati siano nel formato corretto prima di inviare le informazioni al backend.
- **Gestione delle Richieste Asincrone:** JavaScript, in combinazione con AJAX o Fetch API, gestisce le richieste asincrone per caricare dati senza ricaricare la pagina:
  - **Aggiornamento delle Preferenze:** Le modifiche alle preferenze dell'utente, come l'aggiunta di film ai preferiti, sono gestite tramite

richieste asincrone, garantendo che l'interfaccia utente sia aggiornata immediatamente senza necessità di ricaricare la pagina.

- **Animazioni e Transizioni:** JavaScript è utilizzato per implementare animazioni e transizioni che migliorano l'esperienza visiva:
  - **Effetti di Hover e Transizioni:** Quando un utente passa il mouse su una card di film o serie TV, JavaScript applica effetti di hover come ingrandimenti o cambiamenti di colore, rendendo l'interazione più coinvolgente.
  - **Transizioni tra Pagine:** Alcune transizioni tra pagine o sezioni dell'applicazione sono gestite tramite JavaScript, per fornire un'esperienza di navigazione più fluida e meno frammentata.

2.2 Backend: Flask come framework Python per la gestione delle richieste http e la logica dell'applicazione

Per il backend di "CinePlus", abbiamo scelto Flask, un framework web leggero e versatile basato su Python. Flask ci ha permesso di gestire efficacemente le richieste HTTP e implementare la logica dell'applicazione in modo strutturato e scalabile.

2.2.1 Gestione delle Richieste HTTP

Flask è stato utilizzato per gestire tutte le richieste HTTP dell'applicazione, fornendo una base robusta per le interazioni tra il front-end e il server. Alcuni degli aspetti chiave includono:

- **Routing:** Flask ci ha consentito di definire percorsi chiari e organizzati per ogni funzionalità dell'applicazione. Ad esempio, abbiamo creato endpoint specifici per caricare la homepage, visualizzare i dettagli di film e serie TV, gestire le operazioni di login e registrazione, e molto altro.
- **Richieste GET e POST:** Utilizzando Flask, abbiamo gestito richieste GET per recuperare dati dal server e richieste POST per inviare dati al server. Questo è fondamentale per operazioni come la ricerca di contenuti, l'aggiornamento delle preferenze utente, e l'inserimento o modifica di prodotti da parte degli amministratori.
- **Gestione degli Errori:** Flask ci ha permesso di implementare una gestione degli errori personalizzata, fornendo messaggi di errore chiari e informativi agli utenti in caso di problemi come pagine non trovate o errori del server.

2.2.2 Logica dell'Applicazione

Flask è stato essenziale per implementare la logica di business di "CinePlus", gestendo tutto, dalle operazioni di autenticazione degli utenti alla gestione del database:

- **Autenticazione e Autorizzazione**: Abbiamo utilizzato Flask per gestire il processo di login e registrazione, assicurandoci che solo gli utenti autorizzati possano accedere a determinate funzionalità.
- **Gestione dei Preferiti:** Per gli utenti registrati, Flask gestisce la logica per aggiungere e rimuovere film e serie TV dai preferiti, aggiornando in tempo reale il database e assicurando che le modifiche siano riflesse nell'interfaccia utente.
- **Gestione dei Film da Rivedere:** Flask è stato utilizzato anche per gestire la lista dei film da rivedere. Quando un utente registrato guarda un film o una serie TV, Flask aggiorna automaticamente il database aggiungendo il film alla lista dei "Film da Rivedere". Inoltre, Flask consente agli utenti di rimuovere film da questa lista, assicurando che le modifiche siano immediatamente visibili nell'interfaccia utente.
- **Operazioni degli Amministratori:** Flask è stato utilizzato per implementare le funzionalità di amministrazione, permettendo agli amministratori di aggiungere, modificare e cancellare prodotti. Attraverso form sicuri e validazioni lato server, Flask garantisce che solo dati corretti e completi vengano inseriti nel database.

2.2.3 Integrazione con il Front-End

Flask ha svolto un ruolo cruciale nell'integrazione tra front-end e back-end, fornendo API RESTful che il front-end può utilizzare per recuperare e inviare dati. Questo ha permesso di creare un'applicazione fluida e interattiva, dove le azioni dell'utente sul front-end si riflettono immediatamente nel back-end.

2.2.4 MongoDB come dataset NoSQL per la memorizzazione dei dati relativi ai prodotti e agli utenti

Per la memorizzazione dei dati di "CinePlus", abbiamo scelto MongoDB, un database NoSQL noto per la sua flessibilità e scalabilità. MongoDB ci ha permesso di gestire efficacemente i dati relativi ai film, alle serie TV e agli utenti, supportando le esigenze dinamiche della nostra applicazione.

2.2.4.1 Analisi del Dataset

In questa fase, abbiamo analizzato un dataset complesso che conteneva 29 colonne e 15.481 righe. Per gestire in modo efficiente i dati all'interno della nostra applicazione, abbiamo deciso di concentrarci sui primi 2000 prodotti e selezionare alcuni degli attributi più interessanti e rilevanti per "CinePlus". Gli attributi selezionati sono:

- **Title:** Il titolo del film o della serie TV.
- **Genre:** Il/I genere/i del prodotto (ad esempio, dramma, commedia, azione).

- **Series or Movie:** Indica se il prodotto è una serie TV o un film.
- **IMDb Score:** Il punteggio del prodotto su IMDb.
- **Release Date:** La data di rilascio del film o della serie TV.
- **IMDb Link:** Il link alla pagina IMDb del prodotto.
- **Summary:** Una breve trama o descrizione del prodotto.
- **Image:** Un URL all'immagine del prodotto.
- **Name:** Nome degli attori o dei membri dello staff associati al prodotto.
- **Role**: Il ruolo degli attori o dei membri dello staff (ad esempio, attore, regista).

2.2.4.2 Suddivisione del Dataset in Collezioni

Per una gestione più organizzata e intuitiva dei dati, il dataset è stato suddiviso in tre collection principali:

- **Movies_Series:** Questa collection contiene tutti i dati relativi ai film e alle serie TV. Ogni documento in questa collection include attributi come il titolo, il genere, il tipo (film o serie), il punteggio IMDb, la data di rilascio, il link a IMDb, la trama e l'immagine.

Figure 1: Collection Movies_Series

- **People:** Questa collection contiene dati sugli attori e altri membri dello staff coinvolti nei film e nelle serie TV. Ogni documento include il nome della persona e il suo ruolo (ad esempio, attore, regista).

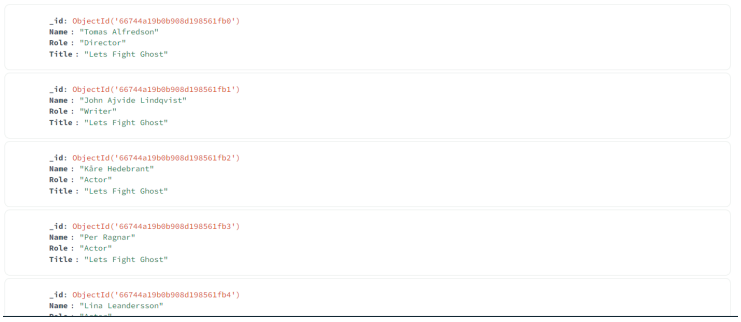


Figure 2: Collection People

- **Users:** Questa collection contiene tutte le informazioni relative agli utenti registrati. Ogni documento include dettagli come il nome utente, la password (criptata), l'email, le preferenze dell'utente e i film/serie TV da rivedere.

Figure 3: Collection Users

2.2.4.3 Integrazione con Flask

L'integrazione tra Flask e MongoDB è stata realizzata utilizzando pymongo, un driver MongoDB per Python. Questo ci ha permesso di interagire facilmente con il database e di eseguire operazioni CRUD (Create, Read, Update, Delete). Alcuni aspetti chiave dell'integrazione includono:

- **Connessione al Database:** Al momento dell'avvio dell'applicazione, Flask si connette a MongoDB utilizzando pymongo. Questa connessione viene mantenuta per eseguire tutte le operazioni necessarie sul database.
- **Operazioni CRUD:** Flask gestisce le operazioni CRUD per le collection Movies_Series e People. Ad esempio, quando un amministratore aggiunge un nuovo film, Flask inserisce un nuovo documento nella collection Movies_Series. Allo stesso modo, gli aggiornamenti e le cancellazioni vengono gestiti attraverso operazioni appropriate su MongoDB.
- **Query Efficaci:** Grazie a MongoDB, siamo in grado di eseguire query efficienti per recuperare dati specifici in base a vari criteri, come il genere, il punteggio IMDb o la data di rilascio. Questo è fondamentale per fornire funzionalità di ricerca e filtraggio avanzate agli utenti.

3 Struttura del codice

La struttura del codice di "CinePlus" è stata attentamente progettata per garantire organizzazione, modularità e facilità di manutenzione. La chiara organizzazione dei file e delle cartelle facilita il lavoro degli sviluppatori e rende il progetto più comprensibile e scalabile.

3.1 Organizzazione dei file e delle cartelle nel progetto

Il progetto "CinePlus" è organizzato all'interno di una cartella principale denominata "Progetto DB2". All'interno di questa cartella, ci sono altre sottocartelle e file chiave che svolgono funzioni specifiche per la gestione del front-end, del back-end e dei dati dell'applicazione. Ecco una panoramica dettagliata della struttura del progetto:

- **static:** Questa cartella contiene tutti i file relativi agli stili CSS che definiscono l'aspetto visivo dell'applicazione, garantendo che l'interfaccia utente sia coerente e accattivante.
  - **styles.css:** File principale degli stili personalizzati per l'applicazione.
  - **styles_button_generi.css:** File degli stili relativi ai bottoni della pagina Generi.
  - **style_movie_details.css:** File degli stili relativi alla pagina del dettaglio di un film o una serie tv.
  - **styles_admin.css:** File degli stili relativi alle pagine dell'amministratore.
- **template:** Questa cartella contiene tutte le pagine HTML. Questi file costituiscono le diverse viste dell'applicazione, che vengono renderizzate e visualizzate dagli utenti.
  - **index.html:** Homepage dell'applicazione con l'elenco di tutti i film e le serie TV e la top 10 basata sul punteggio.
  - **movie_details.html:** Pagina dei dettagli di un film o di una serie TV, con informazioni dettagliate come tipo, anno, genere, punteggio IMDb, trama e cast.
  - **favorites.html:** Pagina con l'elenco di tutti i film e serie TV preferite aggiunte dall'utente. È possibile visualizzarne i dettagli e rimuoverlo dai preferiti.
  - **to_review.html:** Pagina con l'elenco di tutti i film e serie TV viste dall'utente nel momento in cui clicca su "View on IMDb".
  - **film.html:** Pagina con l'elenco di tutti i film.
  - **series.html:** Pagina con l'elenco di tutte le serie TV.
  - **generi.html:** Pagina con l'elenco dei generi.
  - **listaGenere.html:** Pagina che visualizza un elenco di film e serie TV in base al genere selezionato dall'utente.

  - **login.html:** Pagina di login per gli utenti.
  - **register.html:** Pagina per registrare un utente.
  - **add_product.html:** Pagina relativa all'amministratore per aggiungere un prodotto.
  - **admin_dashboard.html:** Pagina relativa alla visualizzazione della dashboard dell'amministratore.
  - **edit_product.html:** Pagina per la modifica di un prodotto da parte dell'amministratore.

- **app.py:** Questo è il file principale dell'applicazione Flask. Contiene tutte le rotte dell'applicazione, definendo come le richieste HTTP sono gestite e quali template HTML vengono renderizzati. È il cuore dell'applicazione, collegando il front-end con il back-end.

- **connectionMongoDB.py:** File dedicato alla gestione della connessione a MongoDB. Contiene la logica necessaria per connettersi al database, eseguire query e gestire i dati in modo sicuro ed efficiente.

- **movies.csv:** File contenente il dataset originale dei film e delle serie TV.

- **moviesDB.csv:** Una versione ridotta del dataset, utilizzata per ridurre il carico durante lo sviluppo.

- **dropRigheDataset.py:** File che gestisce la cancellazione delle righe del dataset originale ("movies.csv").

- **configurazioneFlask.py:** File che gestisce la configurazione di Flask.

Questa struttura ben definita permette una chiara separazione delle responsabilità, facilitando sia lo sviluppo che la manutenzione del progetto. La divisione tra i file di stile (CSS), i template HTML e la logica dell'applicazione (Python) consente agli sviluppatori di lavorare in modo efficiente, concentrandosi su aspetti specifici senza creare confusione nel codice.

Di seguito viene proposta la struttura della web app in maniera semplificata:

```
ProgettoDB2/
    static/
        styles.css
        styles_button_generi.css
        styles_movie_details.css
        styles_admin.css
    templates/
        favorites.html
        films.html
        generi.html
        index.html
        listaGenere.html
```

```
    login.html
    movie_details.html
    register.html
    series.html
    to_review.html
    add_product.html
    admin_dashboard.html
    edit_product.html
app.py
configurazioneFlask.py
connectionMongoDB.py
dropRigheDataset.py
movies.csv
moviesDB.csv
```

4 Funzionalità principali

La web app "CinePlus" è progettata per offrire un'esperienza ricca e personalizzata sia per gli utenti ospiti che per gli utenti registrati, oltre a fornire strumenti amministrativi per la gestione dei contenuti. Ogni categoria di utenti ha accesso a un insieme di funzionalità specifiche, progettate per soddisfare le loro esigenze e migliorare l'interazione con la piattaforma. Di seguito, descriviamo in dettaglio le principali funzionalità disponibili per le diverse tipologie di utenti: ospite, registrato e amministratore.

4.1 Utente Ospite

1. **Visualizzare il catalogo dei prodotti (Film e SerieTV), compresa la Top 10.**

```python
@app.route('/')
def index():
    # Recupera e ordina i primi 10 film/serie in base al punteggio IMDb direttamente con MongoDB
    top_10 = list(movies_collection.find().sort('IMDb Score', -1).limit(10))

    # Recupera tutti i film/serie (senza ordinamento)
    movies_and_series = list(movies_collection.find())

    return render_template('index.html', movies_and_series=movies_and_series, top_10=top_10)
```

Figure 4: Catalogo prodotti e Top 10

La funzione 'index()' recupera i film o serie TV con il punteggio più alto nel database. Utilizza la query find() per ottenere tutti i documenti dalla collezione di film (movies_collection), ordina questi documenti in base al punteggio IMDb in ordine decrescente (query sort()), e li limita ai primi 10 risultati (query limit()). Inoltre, raccoglie tutti i film e le serie TV presenti nel database senza alcun ordinamento o limitazione, utilizzando la query find().

2. **Visualizzare i dettagli di ciascun prodotto.**

```python
@app.route('/movie/<movie_id>')
def movie_details(movie_id):

    # Converti movie_id in ObjectId
    try:
        movie_id = ObjectId(movie_id)
    except:
        return jsonify({'status': 'error', 'message': 'Invalid movie ID format'})

    movie = movies_collection.find_one({'_id': ObjectId(movie_id)})

    if movie is None:
        return jsonify({'status': 'error'})

    pipeline = [
        {
            "$match": {
                "_id": movie_id
            }
        },
        {
            "$lookup": {
                "from": "People",
                "let": { "title": "$Title" },
                "pipeline": [
                    {
                        "$match": {
                            "$expr": { "$eq": ["$Title", "$$title"] }
                        }
                    },
                    {
                        "$group": {
                            "_id": "$Role",
                            "people": { "$push": "$Name" }
```

Figure 5: Dettagli prodotto 1

```python
def movie_details(movie_id):
                            "people": { "$push": "$Name" }
                        }
                    }
                ],
                "as": "people_involved"
            }
        },
        {
            "$project": {
                "_id": 1,
                "Title": 1,
                "Genre": 1,
                "Series or Movie":1,
                "IMDb Score": 1,
                "Release Date": 1,
                "IMDb Link":1,
                "Summary": 1,
                "Image": 1,
                "People Involved": "$people_involved"
            }
        }
    ]

    # Esegui l'aggregazione
    media_details = list(movies_collection.aggregate(pipeline))

    if not media_details:
        return jsonify({'status': 'error'})

    return render_template('movie_details.html', movie=media_details[0])
```

Figure 6: Dettagli prodotto 2

La funzione 'movie_details()' riceve un ID di un film dalla URL. Utilizza poi una query "find_one()" per trovare il documento del film specifico in

base al suo ID e una pipeline di aggregazione per ottenere informazioni dettagliate sul film, incluse le persone coinvolte. La pipeline di aggregazione contiene diverse fasi:

- **$match:** Questa operazione filtra i documenti per trovare solo quello con l'ID specificato (movie_id).
- **$lookup:** Esegue una join con la collezione People per trovare tutte le persone coinvolte nel film, raggruppandole per ruolo.
- **$project:** Seleziona i campi specifici del documento da includere nel risultato finale, come titolo, genere, ecc.

Esegue la pipeline e converte i risultati in una lista.

3. **Ricercare per Titolo, Data e Direttore/Scrittore/Attore.**

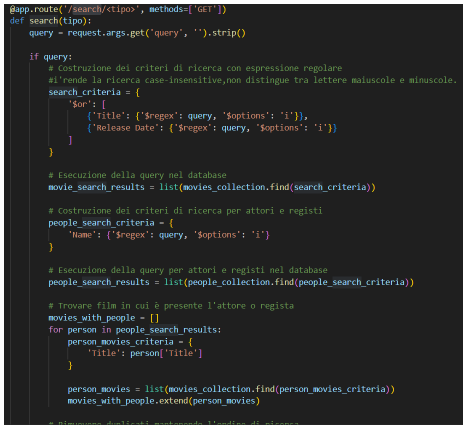


Figure 7: Ricerca 1

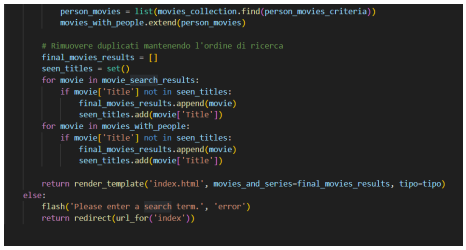


Figure 8: Ricerca 2

La funzione 'search()' gestisce le ricerche nel database per film, serie TV, attori e registi, utilizzando il termine di ricerca fornito dall'utente. La funzione inizia estraendo il termine di ricerca dalla richiesta HTTP. Questo

termine viene ripulito da eventuali spazi bianchi superflui utilizzando il metodo *strip()*. Se il termine di ricerca non è vuoto, la funzione costruisce una query definendo dei criteri di ricera utilizzando espressioni regolari come “regex” per cercare il termine di ricerca nei campi ”Title” e ”Release Date” dei documenti nel database, ignorando le differenze tra maiuscole e minuscole attraverso “($options: ’i’)”. La funzione costruisce anche una query per cercare attori e registi il cui nome corrisponde al termine di ricerca (nello stesso modo precedente). I risultati della ricerca per film vengono combinati con i film associati agli attori e registi trovati. Per evitare duplicati, i film vengono aggiunti a una lista finale solo se non sono già presenti.

4. **Visualizzare l’elenco dei film/serie TV per genere.**

Figure 9: Elenco film/serieTV per genere

La funzione ’show_genre()’ permette di cercare film e serie TV per genere. In particolare, crea un’espressione regolare (RegEx) per cercare il genere specifico all’interno dei dati. L’espressione regolare è configurata per trovare esattamente il genere richiesto, ignorando eventuali differenze tra maiuscole e minuscole. Se si sta cercando un film, la funzione cerca tutti i documenti nella collezione dei media che sono classificati come film e che contengono il genere specificato. Farà la stessa cosa se si sta cercando una serie. Dopo aver trovato tutti i film o le serie TV che corrispondono ricerca, prepara una lista di risultati che vengono poi passati al template HTML.

5. **Registrarsi.**

```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        firstname = request.form['firstname']
        lastname = request.form['lastname']
        birthdate = request.form['birthdate']
        gender = request.form['gender']
        username = request.form['username']
        password = request.form['password']

        if users_collection.find_one({'username': username}):
            flash('Username already exists. Please choose a different one.', 'error')
        else:
            today = datetime.now()
            dob = datetime.strptime(birthdate, '%Y-%m-%d')
            age = today.year - dob.year - ((today.month, today.day) < (dob.month, dob.day))
            if age < 18:
                flash('You must be at least 18 years old to register.', 'error')
            else:
                hashed_password = generate_password_hash(password)
                users_collection.insert_one({
                    'firstname': firstname,
                    'lastname': lastname,
                    'birthdate': birthdate,
                    'gender': gender,
                    'username': username,
                    'password': hashed_password,
                    'to_review': []  # Initialize the to_review list
                })

                flash('Registration successful. Please log in.', 'success')
                return redirect(url_for('login'))

    return render_template('register.html')
```

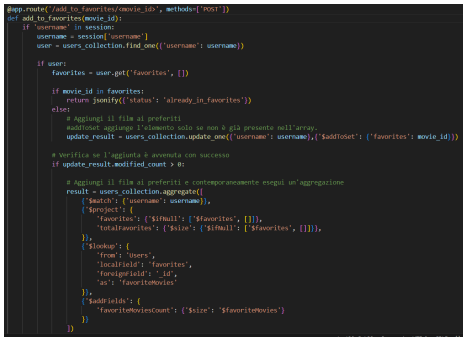
Figure 10: Registrazione

La funzione ‘register()‘ gestisce il processo di registrazione di un nuovo utente. Se la richiesta è di tipo POST, la funzione recupera i dati forniti, come nome, cognome, data di nascita, genere, nome utente e password. Controlla se esiste già un utente con lo stesso username e se già in uso, viene visualizzato un messaggio di errore. Nel caso dell’età si verifica se ha almeno 18 anni. Se l’età è inferiore, mostra un messaggio di errore. Se le credenziali sono corrette la password viene criptata e l’utente viene aggiunto alla collezione ‘users_collection‘. Visualizza un messaggio di successo e reindirizza l’utente alla pagina di login. Se la richiesta è di tipo GET, la funzione renderizza il modulo di registrazione.

4.2 Utente Registrato

Ha gli stessi permessi di un Utente Ospite con la possibilità di:

1. **Aggiungere Film e SerieTV alla lista dei preferiti.**



```python
@app.route('/add_to_favorites/<movie_id>', methods=['POST'])
def add_to_favorites(movie_id):
    if 'username' in session:
        username = session['username']
        user = users_collection.find_one({'username': username})

        if user:
            favorites = user.get('favorites', [])

            if movie_id in favorites:
                return jsonify({'status': 'already_in_favorites'})
            else:
                # Aggiungi il film ai preferiti
                #addToset aggiunge l'elemento solo se non è già presente nell'array.
                update_result = users_collection.update_one({'username': username},{'$addToSet': {'favorites': movie_id}})

            # Verifica se l'aggiunta è avvenuta con successo
            if update_result.modified_count > 0:

                # Aggiungi il film ai preferiti e contemporaneamente esegui un'aggregazione
                result = users_collection.aggregate([
                    {'$match': {'username': username}},
                    {'$project': {
                        'favorites': {'$ifNull': ['$favorites', []]},
                        'totalFavorites': {'$size': {'$ifNull': ['$favorites', []]}},
                    }},
                    {'$lookup': {
                        'from': 'Users',
                        'localField': 'favorites',
                        'foreignField': '_id',
                        'as': 'favoriteMovies'
                    }},
                    {'$addFields': {
                        'favoriteMoviesCount': {'$size': '$favoriteMovies'}
                    }}
                ])
```

Figure 11: Aggiungere prodotto ai preferiti 1

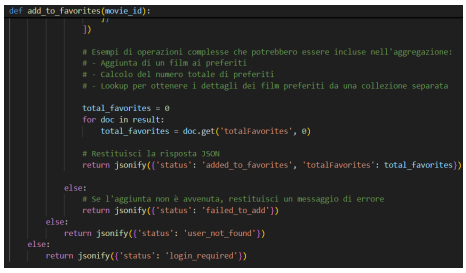


Figure 12: Aggiungere prodotto ai preferiti 2

La funzione ‘add_to_favorites()‘ gestisce l’aggiunta di un film alla lista dei preferiti di un utente autenticato. In particolare, verifica se l’utente è autenticato, se non lo è restituisce un messaggio di errore indicando che è necessario effettuare il login. Utilizzando una query, recupera i dettagli dell’utente tra cui la lista dei film già presenti nei suoi preferiti. Aggiunge il film alla lista eseguendo la query “update_one” che aggiorna un singolo documento nella collezione che soddisfa un determinato criterio (documento di uno specifico utente) per poi eseguire un operazione di aggiornamento “$addToSet” che è un operatore di MongoDB che aggiunge un valore a un array solo se quel valore non è già presente nell’array. Questo previene i duplicati. Se il valore è già presente, restituisce un messaggio che indica che il film è già nei preferiti. Dopo aver aggiornato la lista dei preferiti, esegue una pipeline di aggregazione per ottenere informazioni aggiornate sui preferiti, dove:

- **$match:** Filtra i documenti per trovare quello dell’utente specificato.
- **$project:** Calcola il numero totale di film nei preferiti.
- **$lookup:** Esegue una join con la collezione ‘Users‘ per ottenere dettagli sui film preferiti.
- **$addFields:** Aggiunge il conteggio dei film preferiti come nuovo campo.

Se l’aggiornamento è avvenuto con successo, restituisce il numero totale di preferiti mentre se l’aggiunta non è riuscita, restituisce un messaggio di errore.

2. **Quando guarda un film o una serieTV questo automaticamente viene aggiunto ad una lista dei film da rivedere.**

Figure 13: Aggiunta prodotto alla sezione da rivedere

La funzione 'add_to_review()' gestisce l'aggiunta di un film alla lista di film che un utente desidera rivedere. La funzione, prima controlla l'utente se è autenticato, in caso contrario restituisce un messaggio richiedendo l'autenticazione. Aggiunge il film alla lista eseguendo la query "update_one" che aggiorna un singolo documento nella collezione che soddisfa un determinato criterio (documento di uno specifico utente) per poi eseguire un operazione di aggiornamento '$set', un operatore di MongoDB che imposta il valore di un campo specifico. Se il campo esiste già, il suo valore viene aggiornato. Se il campo non esiste, viene aggiunto al documento. Se il film è stato aggiunto con successo alla lista di film da rivedere, restituisce una risposta JSON con lo stato 'added to review'.

3. **Eliminare il suo profilo.**

Figure 14: Eliminare profilo

La funzione 'delete_profile()' gestisce la cancellazione del profilo dell'utente attualmente autenticato. In particolare, controlla se il nome utente è autenticato, in caso contrario la funzione non esegue alcuna operazione e semplicemente reindirizza l'utente alla pagina principale. Se l'utente è autenticato, la funzione esegue una query "delete_one" di cancellazione di un documento di uno specifico utente dalla collezione 'users_collection'.Dopo aver eliminato il profilo dell'utente dal database, la funzione rimuove il nome utente dalla sessione, disconnettendo effettivamente l'utente e aggiunge un messaggio di conferma alla sessione che indica che il profilo è stato eliminato con successo. Infine, reindirizza l'utente alla pagina principale.

4.3 Utente Amministratore

1. **Visualizzare la lista di tutti i Film e le SerieTV presenti nel dataset.**

Figure 15: Lista prodotti amministratore

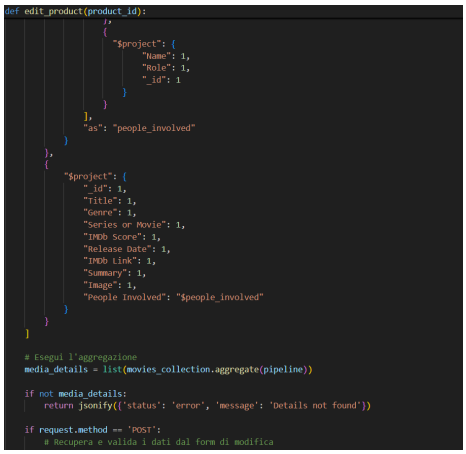
La funzione 'admin_dashboard()' è responsabile del caricamento della pagina iniziale per gli amministratori. Mostra una lista di film e serie TV presenti nel database. In particolare, verifica se l'utente ha i privilegi di amministratore controllando se ''is_admin'' è presente nella sessione dell'utente. Se non lo è, mostra un messaggio di errore e reindirizza l'utente alla pagina di login. Se l'utente è un amministratore, esegue una query sulla collezione 'movies_collection' per ottenere tutti i documenti (film e serie TV). La funzione 'find()' recupera tutti i documenti nella collezione, e 'list()' converte il cursore restituito da 'find()' in una lista di documenti. Passa la lista di prodotti (film e serie TV) al template 'admin_dashboard.html' per visualizzarli nella pagina. Se l'utente non è un amministratore, visualizza un messaggio di errore e reindirizza l'utente alla pagina di login.

2. **Aggiungere, Modificare, Eliminare prodotti.**

Figure 16: Aggiungere prodotto 1

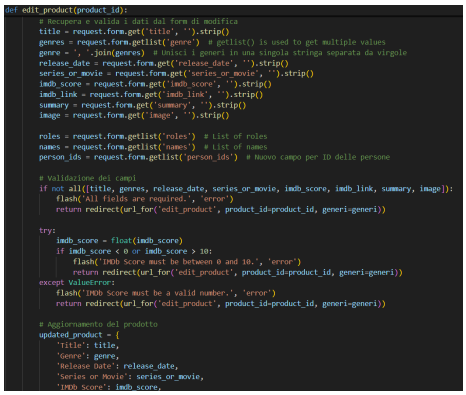
Figure 17: Aggiungere prodotto 2

Figure 18: Modificare prodotto 1



```
def edit_product(product_id):
                    },
                    {
                      "$project": {
                          "Name": 1,
                          "Role": 1,
                          "_id": 1
                      }
                    }
                ],
                "as": "people_involved"
            }
        },
        {
            "$project": {
                "_id": 1,
                "Title": 1,
                "Genre": 1,
                "Series or Movie": 1,
                "IMDb Score": 1,
                "Release Date": 1,
                "IMDb Link": 1,
                "Summary": 1,
                "Image": 1,
                "People Involved": "$people_involved"
            }
        }
    ]

    # Esegui l'aggregazione
    media_details = list(movies_collection.aggregate(pipeline))

    if not media_details:
        return jsonify({'status': 'error', 'message': 'Details not found'})

    if request.method == 'POST':
        # Recupera e valida i dati dal form di modifica
```

Figure 19: Modificare prodotto 2

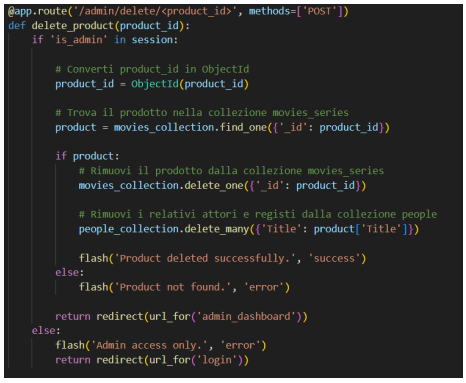


```
def edit_product(product_id):
    # Recupera e valida i dati dal form di modifica
    title = request.form.get('title', '').strip()
    genres = request.form.getlist('genre')  # getlist() is used to get multiple values
    genre = ', '.join(genres)  # Unisci i generi in una singola stringa separata da virgole
    release_date = request.form.get('release_date', '').strip()
    series_or_movie = request.form.get('series_or_movie', '').strip()
    imdb_score = request.form.get('imdb_score', '').strip()
    imdb_link = request.form.get('imdb_link', '').strip()
    summary = request.form.get('summary', '').strip()
    image = request.form.get('image', '').strip()

    roles = request.form.getlist('roles')  # List of roles
    names = request.form.getlist('names')  # List of names
    person_ids = request.form.getlist('person_ids')  # Nuovo campo per ID delle persone

    # Validazione dei campi
    if not all([title, genres, release_date, series_or_movie, imdb_score, imdb_link, summary, image]):
        flash('All fields are required.', 'error')
        return redirect(url_for('edit_product', product_id=product_id, generi=generi))

    try:
        imdb_score = float(imdb_score)
        if imdb_score < 0 or imdb_score > 10:
            flash('IMDb Score must be between 0 and 10.', 'error')
            return redirect(url_for('edit_product', product_id=product_id, generi=generi))
    except ValueError:
        flash('IMDb Score must be a valid number.', 'error')
        return redirect(url_for('edit_product', product_id=product_id, generi=generi))

    # Aggiornamento del prodotto
    updated_product = {
        'Title': title,
        'Genre': genre,
        'Release Date': release_date,
        'Series or Movie': series_or_movie,
        'IMDb Score': imdb_score,
```

Figure 20: Modificare prodotto 3

```
def edit_product(product_id):
        'Series or Movie': series_or_movie,
        'IMDb Score': imdb_score,
        'IMDb Link': imdb_link,
        'Summary': summary,
        'Image': image,
    }

    movies_collection.update_one({'_id': product_id}, {'$set': updated_product})

    # Aggiorna il titolo delle persone
    old_title = movie['Title']
    if old_title != title:
        people_collection.update_many(
            {'Title': old_title},
            {'$set': {'Title': title}}
        )

    # Aggiorna le persone
    for person_id, role, name in zip(person_ids, roles, names):
        try:
            person_id = ObjectId(person_id) # Assicurati che person_id sia un ObjectId valido
            print(f'Updating person with Role: {role}, Name: {name}')
            result = people_collection.update_one(
                {'_id': person_id },
                {'$set': {'Role': role,'Name': name}}
            )
        except Exception as e:
            flash(f'Error updating person with ID: {person_id}. Error: {str(e)}', 'error')
            return redirect(url_for('edit_product', product_id=product_id, generi=generi))

    flash('Movie/Series updated successfully.', 'success')

    return redirect(url_for('edit_product', product_id=product_id, generi=generi))

return render_template('edit_product.html', movie=media_details[0], generi=generi)
```

Figure 21: Modificare prodotto 4



```
@app.route('/admin/delete/<product_id>', methods=['POST'])
def delete_product(product_id):
    if 'is_admin' in session:

        # Converti product_id in ObjectId
        product_id = ObjectId(product_id)

        # Trova il prodotto nella collezione movies_series
        product = movies_collection.find_one({'_id': product_id})

        if product:
            # Rimuovi il prodotto dalla collezione movies_series
            movies_collection.delete_one({'_id': product_id})

            # Rimuovi i relativi attori e registi dalla collezione people
            people_collection.delete_many({'Title': product['Title']})

            flash('Product deleted successfully.', 'success')
        else:
            flash('Product not found.', 'error')

        return redirect(url_for('admin_dashboard'))
    else:
        flash('Admin access only.', 'error')
        return redirect(url_for('login'))
```

Figure 22: Eliminare prodotto

AGGIUNTA PRODOTTO

La funzione ‘add_product()‘ gestisce l’aggiunta di un nuovo film o serie TV al database. E’ accessibile solo agli amministratori e utilizza un modulo HTML per raccogliere i dettagli del prodotto e delle persone coinvolte. Viene invocata la funzione ‘*get_generi()*‘ che recupera la lista dei generi disponibili per popolare un campo di selezione nel modulo HTML. Verifica se l’utente ha i privilegi di amministratore. Se non è un amministratore, l’utente viene reindirizzato alla pagina di login con un messaggio di errore. La richiesta POST estrae i dati inviati dal modulo HTML e li pulisce rimuovendo spazi extra. Verifica che tutti i campi obbligatori siano compilati e validi. Se manca qualche dato, viene mostrato un messaggio di errore e l’utente viene reindirizzato al modulo di inserimento. Crea un documento con i dettagli del film o serie TV e lo inserisce nella collezione ‘movies_collection‘ usando una query “insert_one“. Estrae i ruoli e i nomi

delle persone coinvolte nel film o serie TV dal modulo e crea un documento per ogni persona e lo inserisce nella collezione 'people_collection' utilizzando la query "insert_one". Dopo aver inserito i dati, mostra un messaggio di successo e reindirizza l'utente alla dashboard dell'amministratore.

MODIFICA PRODOTTO

La funzione 'edit_product()' gestisce la modifica dei dettagli di un film o una serie TV. La funzione '*get_generi()*' ottiene la lista dei generi disponibili, che verrà usata per popolare il modulo di modifica. Con l'ID del prodotto, passato nella URL, vengono recuperati i dettagli di uno specifico film o serie TV e se il prodotto non esiste, restituisce un errore. La pipeline di aggregazione esegue le seguenti operazioni:

- **$match:** Filtra il documento con l'ID specificato.
- **$lookup:** Esegue una join con la collezione 'People' per ottenere informazioni sulle persone coinvolte nel film o serie TV.
- **$project:** Seleziona i campi da restituire, inclusi i dettagli del film e le persone coinvolte.

Esegue la pipeline di aggregazione e converte il risultato in una lista. La POST recupera i dati inviati dal modulo HTML, esegue la validazione per assicurarsi che tutti i campi siano compilati. Crea un documento con i dati aggiornati del film o serie TV e aggiorna il documento esistente utilizzando una query "update_one". Se il titolo del film è cambiato, aggiorna il campo 'Title' di tutti i documenti nella collezione 'people_collection' che contengono il vecchio titolo, utilizzando la query "update_many". Per ogni persona coinvolta aggiorna il ruolo e il nome nella collezione 'people_collection' con una query "update_one". Dopo aver completato l'aggiornamento, mostra un messaggio di successo e reindirizza l'utente al modulo di modifica con i nuovi dettagli.

ELIMINAZIONE PRODOTTO

La funzione 'delete_product()' gestisce la cancellazione di un prodotto (film o serie TV) dal database. Verifica se l'utente ha i privilegi di amministratore, controllando se la chiave ''is_admin'' è presente nella sessione. Se non è presente, mostra un messaggio di errore e reindirizza l'utente alla pagina di login. Converte l'ID del prodotto, passato nella URL, in un oggetto 'ObjectId' di MongoDB. Questo è necessario perché gli ID nei documenti di MongoDB sono di tipo 'ObjectId'. Cerca il prodotto nella collezione 'movies_collection' utilizzando l'ID convertito e un apposita query. Se il prodotto non esiste, viene mostrato un messaggio di errore. Rimuove il prodotto dalla collezione 'movies_collection' utilizzando l'ID del prodotto (query) e tutte le persone (attori, registi, ecc.) dalla collezione 'people_collection' che sono associate al titolo del prodotto eliminato (query). Mostra un messaggio di successo se il prodotto è stato eliminato correttamente e reindirizza l'utente alla dashboard dell'amministratore.

3. **Ricercare Film e SerieTV per Titolo, Data e Artista.**

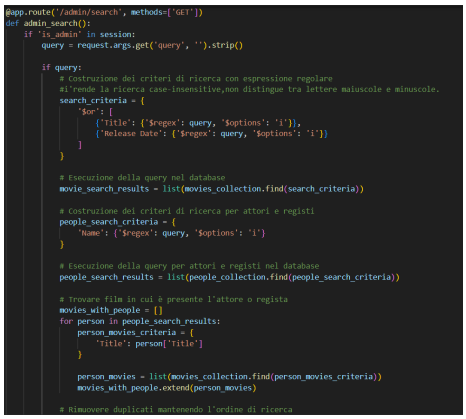


Figure 23: Ricerca amministratore 1

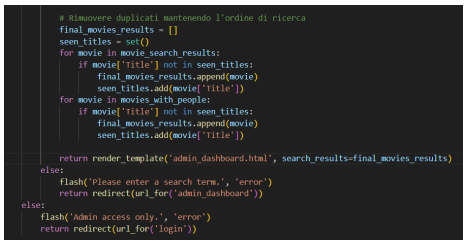


Figure 24: Ricerca amministratore 2

La funzione 'admin_search()' permette agli amministratori di cercare film e serie TV, così come attori e registi, nel database. Se un amministratore effettua una ricerca, il sistema restituirà i risultati pertinenti. In particolare, verifica se l'utente ha i privilegi di amministratore, controllando se ''is_admin'' è presente nella sessione. Se non lo è, mostra un messaggio di errore e reindirizza l'utente alla pagina di login. Ottiene la query di ricerca dalla richiesta HTTP GET. Utilizza '*strip()*' per rimuovere eventuali spazi bianchi all'inizio e alla fine della query e un'espressione regolare per cercare film o serie TV nel database. La ricerca è case-insensitive. Esegue la query nella collezione 'movies_collection' e recupera tutti i documenti che soddisfano i criteri di ricerca. I risultati sono convertiti in una lista. Costruisce una query per cercare attori e registi nel database. Anche questa ricerca è case-insensitive. Esegue la query nella collezione 'people_collection' e recupera tutti i documenti che soddisfano i criteri di ricerca. Per ogni persona trovata, cerca i film associati a quella persona nella collezione 'movies_collection' e aggiunge i risultati alla lista

‘movies_with_people‘. Combina i risultati della ricerca di film e di quelli associati agli attori e registi, assicurandosi di non includere duplicati. Utilizza un set ‘seen_titles‘ per tenere traccia dei titoli già aggiunti ai risultati finali.