

“Requestarr”

Progetto Finale di Tecnologie Web

Taccini Lorenzo (mat. 165715)

AA. 2023/2024

Sommario

Introduzione	2
Funzionalità richieste e differenziazione di ruoli.....	2
Features principali	2
Tipologie di utenti	3
Focus sullo scopo del servizio.....	4
Riassunto delle Autorizzazioni.....	5
Modelli e Struttura del Database.....	5
Rappresentazione logica	5
Diagramma delle classi	6
Modelli in Django.....	7
Realizzazione di Requestarr	8
Tecnologie utilizzate	8
Frontend	8
Backend.....	8
Suddivisione in Django Applications.....	9
Creazione User-Profile	11
Funzionalità di ricerca	11
Recommendation System	12
Unit Testing	12
Problemi riscontrati.....	12
Risultato finale	13
Home Page: differenza tra utente loggato e anonimo	13
Pagina con dettagli del titolo	14
Ricerca con auto completamento e suggestions.....	14
Moderator Dashboard	15
Watchlist.....	16

Introduzione

L'obiettivo è quello di realizzare un **servizio di Media Discovery and Request** per un utilizzo in-house.

L'idea nasce dalla necessità personale e concreta di rendere più efficace e fruibile l'utilizzo di un media server self-hosted per utilizzo privato che offra servizi come Plex e/o Jellyfin per lo streaming di contenuti multimediali in rete locale: i fruitori dei suddetti servizi (amici, parenti) vorrebbero poter visualizzare un catalogo di film completo e aggiornato, da cui selezionare titoli di loro interesse per una futura visione.

Se non già presente nel **catalogo del media server**, gli utenti potranno richiedere il titolo di loro interesse in modo che l'utente gestore del servizio possa adoperarsi per acquistarlo e/o noleggiarlo e renderlo disponibile per tutti, (soddisfando, quindi, la richiesta).

La libreria self-hosted si va così componendo di titoli espressamente voluti dai fruitori del servizio, a differenza di più blasonati servizi streaming dove l'utente è un semplice "osservatore" e non ha potere decisionale sul catalogo proposto.

L'idea generale prende ispirazione da progetti preesistenti e diffusi nel mondo dei media server self-hosted, come Overseerr, cercando una reinterpretazione più personale e concreta, e realizzata utilizzando il framework Django.

Non si tratta, quindi, di proporre un servizio di media streaming, quanto piuttosto un servizio che permetta agli utenti di scoprire e scegliere titoli in base ai loro interessi e di richiederne la disponibilità su una piattaforma esterna e arbitraria.

I dati utilizzati nel catalogo sono basati su parti del **Database ufficiale di TMDb**, punto di riferimento nella categoria, opportunamente adattati alle necessità del servizio web che si va a creare.

La scelta del nome dell'applicazione, "**Requestarr**", prende ancora una volta spunto da una famiglia di variegati servizi open-source rivolti al mondo del LAN media streaming e sviluppati nel corso degli anni, dove il suffisso "**-rr**" vuole indicare l'appartenenza ad **un unico ecosistema di applicazioni** che, seppur nate dalla dedizione e passione di sviluppatori indipendenti, offrono funzionalità condivise e simbiotiche.

Funzionalità richieste e differenziazione di ruoli

Features principali

- **Catalogo di titoli aggiornato e aggiornabile**, con dettagli su ogni elemento, ottenuti direttamente da [TMDb](#) tramite una utility di **scraping** che permette di estrarre dati (randomici, per finalità di testing) tramite richieste alle API di TMDb.

- **Scraping di immagini:** le immagini di copertina vengono ottenute direttamente da TMDB tramite richieste real-time utilizzando le API ufficiali. Si evita il salvataggio in locale di una grande quantità di immagini, con risparmio in termini di spazio di archiviazione.
- **Richiesta di titoli** ai moderatori da parte di utenti registrati.
- **Dashboard personale** per la customizzazione del profilo utente.
- **Watchlist** privata per-user, che permette di creare un catalogo privato di titoli da guardare in futuro.
- Dashboard utente per visualizzare lo **stato** delle proprie richieste.
- **Recommendation System** basato su affinità tra generi televisivi.
- **Strumento di ricerca** avanzato sui dati presenti a catalogo, con correzione degli errori sulla query di ricerca nel caso di *mispelling* da parte dell'utilizzatore finale.

Le funzionalità di richiesta di titoli, del loro inserimento o rimozione in watchlist, della gestione delle richieste da parte di utenti autorizzati, e dell'aggiunta di nuovi titoli a catalogo da parte di questi ultimi, richiede necessariamente l'implementazione di **operazioni CRUD** sul database, per permettere modifica, aggiunta, ed eliminazione di dati.

Tipologie di utenti

Risulta inoltre fondamentale aggiungere un metodo di **differenziazione degli utenti** e dei loro permessi: un servizio del genere potrà infatti essere sfruttato da utenti con diverse necessità, scopi e ruoli, ed è quindi naturale doverli distinguere.

In particolare:

- **Utenti ANONIMI**, che non hanno effettuato alcun tipo di processo di registrazione (sign-up) o di accesso (login), avranno comunque accesso alla visione dell'intero catalogo di titoli proposto, ma in modalità "*read-only*": non avranno la possibilità di visualizzare schermate dedicate alle richieste di titoli, alla watchlist, né avranno modo di effettivamente compiere azioni, rispettivamente, di aggiunta o richiesta di essi. Un tipico esempio di utente anonimo potrebbe essere rappresentato da una persona che vuole toccare con mano il servizio, interagire con l'interfaccia utente, e valutare una futura registrazione.
- **Utenti Registered**, o registrati, avranno effettuato la creazione di un account "con permessi minimi" che consente la fruizione delle principali funzionalità offerte da Requestarr, come la richiesta di titoli o l'inserimento di essi in watchlist. Viene proposta, ovviamente, una nuova dashboard in cui personalizzare le proprie informazioni e poter aggiungere una foto profilo personalizzata.

Si prevede che gli utenti Registered rappresentino la maggioranza degli agenti fruitori della piattaforma, svolgendo sostanzialmente il ruolo di “clienti” del servizio.

- **Utenti Moderator**, o moderatori, rappresentano gli “addetti ai servizi”: il loro account può essere registrato solo dall’amministratore della piattaforma, in modo che esso possa selezionare accuratamente a chi fornire questa classe di permessi aggiuntivi. Oltre a poter usufruire delle funzionalità di un utente Registered, i Moderator hanno la possibilità (tramite una dashboard dedicata) di gestire le richieste di tutti gli utenti, approvandole, rifiutandole, o lasciandole eventualmente in uno stato di attesa (*pending*).
Possono inoltre, per ogni titolo con almeno una richiesta, **marcarlo come disponibile** sulla piattaforma di streaming.
Infine, gli utenti Moderator hanno la possibilità di **aggiungere nuovi film a catalogo**, in modo rapido. Vedremo di seguito questa e altre funzionalità illustrate nel dettaglio.
- **Utente Admin**, è l’utente amministratore di default definito dal framework Django. I suoi permessi sono completi: estende ulteriormente le capacità dell’utente Moderator aggiungendo, appunto, la possibilità di registrare questi ultimi, oltre a garantire l’accesso (anche da GUI) al pannello di amministrazione di Django.

Focus sullo scopo del servizio

È necessario spendere qualche parola aggiuntiva sul ruolo di alcune funzioni brevemente accennate in precedenza, che permettono di meglio comprendere lo scopo e l’utilità di un servizio come Requestarr:

- **Mark as Available**, ossia la possibilità da parte di utenti Moderator (o amministratori) di segnare un titolo, richiesto da uno o più utenti come disponibile, è una funzionalità fondamentale e rappresentativa dello scopo dell’intera piattaforma: un titolo marcato come disponibile segnala, anche visivamente all’utente fruitore, che quest’ultimo è stato reperito (dal moderatore, o dall’amministratore) previa richiesta da parte di uno o più utenti e reso accessibile su una piattaforma di streaming self-hosted terza (che esula dallo scopo di questo progetto).
Di conseguenza, il titolo in questione non potrà più essere richiesto da parte di nessun utente, che potrà trovarlo già disponibile per la visione dove previsto.

L’obiettivo è quindi stato raggiunto:



- La possibilità da parte dei soli amministratori di poter creare utenti Moderator rappresenta la volontà di fornire permessi di alto livello solo a persone “fidate”: un amministratore concederà il ruolo di moderatore a un utente con buon senso nel giudizio di approvazione delle richieste, dimestichezza nell'utilizzo della piattaforma ma soprattutto dotato di una visione d'insieme sullo stato del server; un moderatore non può, per esempio, non avere i mezzi o le competenze necessarie per conoscere lo spazio di archiviazione ancora disponibile sull'hardware, oppure non essere in grado di reperire, e quindi di conseguenza inserire, titoli all'interno della piattaforma di streaming locale scelta.

Riassunto delle Autorizzazioni

Di seguito una breve **tabella riassuntiva** delle autorizzazioni relative alle varie tipologie di utente:

	Catalogo titoli	Piattaforma richieste	Watchlist	Gestione richieste	Creazione Moderator	Pannello Admin Django
Anonimo	•					
Registered	•	•	•			
Moderator	•	•	•	•		
Admin	•	•	•	•	•	•

Modelli e Struttura del Database

Rappresentazione logica

Si ha ora la necessità di progettare la struttura del database che conterrà i dati necessari al nostro servizio e ne governerà le relazioni.

La scelta delle classi è quella più logica e semplice per un *use-case* come questo:

- Una classe **Movie**, a rappresentare un titolo del catalogo, con tutti i campi necessari a caratterizzare un film o una serie tv:
 - ID di TMDB, utile nelle richieste API. Siccome anche su TMDB l'ID di un film è sempre univoco, si è scelto di delegare a questo campo il ruolo di primary key della tabella.
 - Generi
 - Anno di rilascio
 - Descrizione
- La classe **Profile**, volta a estendere la funzionalità della classe “User” che di default rappresenta un utente in Django, e che aggiunge elementi come la foto profilo. È in

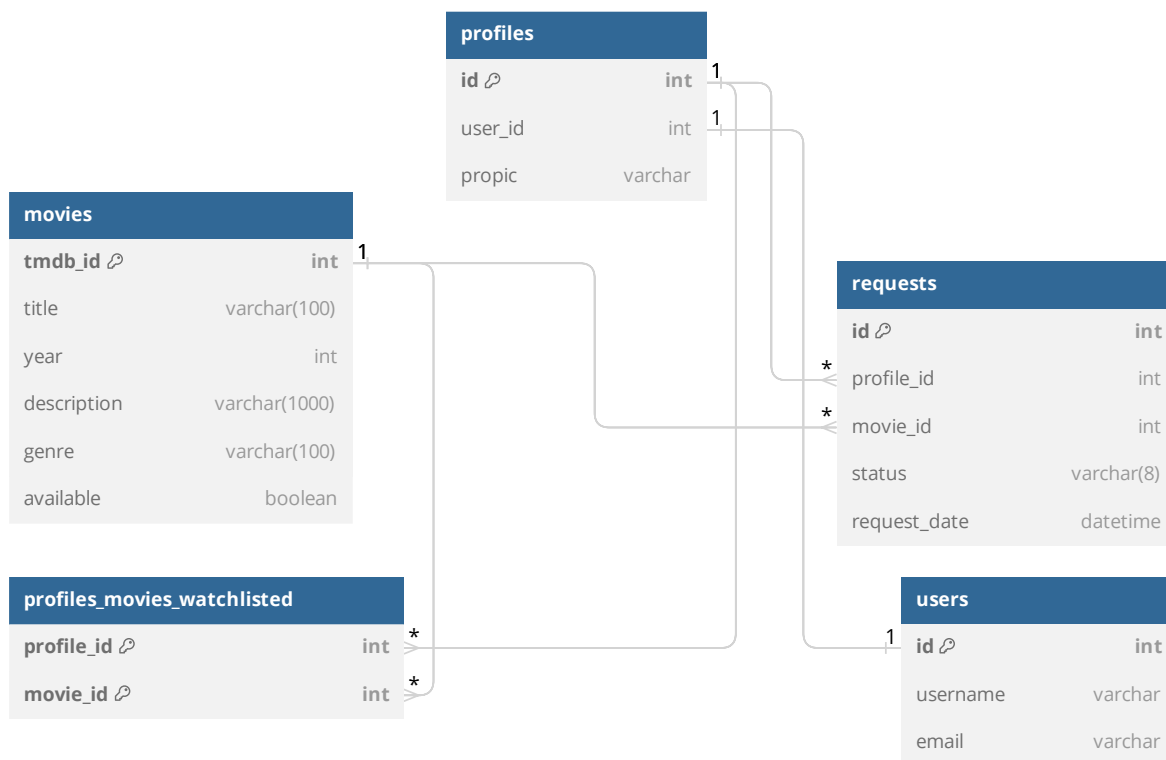
relazione one-to-one con User. Inoltre, la classe contiene due relazioni many-to-many, entrambe con la tabella Movie:

- “requests”, tramite tabella Request, vuole rappresentare tutti i titoli richiesti da un certo profilo.
 - “watchlisted”, indica tutti i titoli inseriti nella watchlist da un determinato utente.
- La classe **Request**, volta a rappresentare una singola richiesta effettuata da un utente su un titolo, lega le classi Movie e Profile, aggiungendo informazioni come la data della richiesta e il suo stato (disponibile tra PENDING, APPROVED, REJECTED). Per un utente (Profile) e un Movie può esistere una sola entry in Request, per questo viene aggiunto un **unique constraint** che lega questi due campi ed evita una qualsivoglia duplicazione.

Diagramma delle classi

La struttura del Database necessaria per l’implementazione di tale servizio è quindi piuttosto semplice e immediata, per questo non si è sentita la necessità di variare tipologia di DB rispetto a quello di default scelto da Django, ossia **SQLite**.

Di seguito è riportato un esplicativo diagramma delle classi sopra brevemente descritte, che evidenzia anche la relazione che unisce le varie tabelle del DB (one-to-one, many-to-one, ecc.):



In questo caso, le relazioni many-to-many della classe Profile vengono rappresentate in modo diverso, e molto simile a quello che avviene poi nella logica applicativa di Django: una relazione many-to-many viene infatti rappresentata tramite la creazione di una nuova tabella

fittizia che contiene le foreign key delle due classi coinvolte nella relazione. Nel caso della relazione many-to-many “requests” però, specificando il parametro “through” (attraverso), si indica di usare come tabella fittizia la tabella Request.

Modelli in Django

Di seguito è brevemente riportata l’implementazione dei modelli Django (ereditati dalla classe di default model.Model), che permettono tramite operazioni di migrazione di generare automaticamente la struttura del Database:

```
class Movie(models.Model):
    tmdb_id = models.IntegerField(primary_key=True)
    title = models.CharField(max_length=100)
    year = models.IntegerField(null=True)
    description = models.TextField(max_length=1000)
    genre = models.CharField(max_length=100)
    available = models.BooleanField(default=False)
    class Meta:
        ordering = ["title"]
        verbose_name_plural = 'Movies'
```

```
class Profile(models.Model):
    user = models.OneToOneField(User, related_name='profile', on_delete=models.CASCADE)
    propic = models.ImageField(
        upload_to='profilepictures/',
        default=join('static', 'unknown_user.png'),
        blank=True
    )
    requests = models.ManyToManyField(Movie, through='Request', blank=True,
                                     related_name='requested_by', default=None)
    watchlisted = models.ManyToManyField(Movie, blank=True, related_name='watchlisted', default=None)
```

```
class Request(models.Model):
    APPROVED = 'approved'    REJECTED = 'rejected'    PENDING = 'pending'    STATUS = [
        (APPROVED, 'Request has been approved'),
        (REJECTED, 'Request has been rejected'),
        (PENDING, 'Request not yet evaluated')
    ]
    profile = models.ForeignKey(Profile, related_name='profile', on_delete=models.CASCADE)
    movie = models.ForeignKey(Movie, related_name='movie_requests', on_delete=models.CASCADE)
    status = models.CharField(choices=STATUS, default=PENDING, max_length=8)
    request_date = models.DateTimeField(auto_now_add=True)
    class Meta:
        ordering = ["-request_date"]
        verbose_name_plural = 'Requests'
        constraints = [models.UniqueConstraint(fields=['profile', 'movie'],
                                              name='unique_profile_movie_request')]
```

Si noti come i modelli sono stati qui riportati rimuovendo le funzioni aggiuntive presenti nella loro implementazione, in quanto non rilevanti per comprendere la modellazione del DB.

Realizzazione di Requestarr

Tecnologie utilizzate

Entrando nel vivo della realizzazione del progetto, approfondiamo e motiviamo brevemente le tecnologie utilizzate per l'implementazione:

Frontend

Data l'assoluta inesperienza in web design, per il frontend si è scelto un approccio "classico", utilizzando una combinazione di **HTML**, stilizzato grazie all'utilizzo di Bootstrap4 e CSS, e **JavaScript**, oltre alla ovvia presenza di linguaggio DTL per l'interazione con il Framework Django.

JavaScript ricopre un ruolo fondamentale per abilitare un funzionamento simil-dinamico delle pagine web, consentendo ad esempio la comparsa di banner e messaggi *modal*, oppure modificando oggetti HTML e loro proprietà senza il bisogno di fare il refresh della pagina nel Browser; in particolare, molto utile è stato il suo impiego in combinazione con **Ajax**, permettendo l'invio di richieste GET e POST asincrone a views di Django senza la necessità di reindirizzamento ad altre pagine (o alla pagina stessa).

Si è poi scelto di aggiungere una dipendenza dalla libreria FontAwesome, che fornisce un'ampia scelta di icone personalizzabili da inserire all'interno delle pagine.

Backend

Per un corretto funzionamento del backend, si è reso necessario l'utilizzo di applicazioni come:

- **Pillow**, per la gestione delle immagini all'interno dei modelli e delle tabelle del database, nel nostro caso utile per il campo `propic` nella tabella `Profile`, che rappresenta le foto profilo di ogni utente.
- **Django-braces**, per aggiungere ulteriori funzionalità di altrimenti difficile implementazione nelle CBV, come ad esempio controlli sui permessi di accesso agli utenti (classi `SuperUserRequiredMixin` o `LoginRequiredMixin`).
- **API TMDb**, necessarie per interfacciarsi con "The Movie Database" in tutti gli use cases, dal popolamento iniziale della tabella `Movie` del DB, all'aggiunta di nuovi titoli da parte di moderatori e amministratori, al *retrieve* di ogni immagine di copertina in real-time quando necessario. Le richieste vengono effettuate tramite modulo "requests", di seguito è riportata in un esempio la costruzione di una richiesta GET per ottenere un certo numero di titoli da TMDb:

```
base_url = 'https://api.themoviedb.org/3'
endpoint = '/discover/movie'
params = {
    'api_key': YOUR_API_KEY,
    'sort_by': 'popularity.desc',
    'include_adult': 'false',
    'include_video': 'false',
    'page': page_number_where_to_get_movies_from,
}

# Effettua la richiesta GET all'API di TMDB
response = requests.get(base_url + endpoint, params=params)
if response.status_code == 200:
    movies = response.json()['results']
    # Estrai casualmente 'num_movies' film
    random_movies = random.sample(movies, num_movies)
    return random_movies
```

- **Difflib**, libreria Python utile per calcolare la similarità tra due sequenze di caratteri, fondamentale per l'implementazione della funzionalità di ricerca “insensibile” ad una certa quantità di errori grammaticali nelle query di ricerca. Di seguito approfondiremo nel dettaglio.

Suddivisione in Django Applications

Per la realizzazione di Requestarr, la logica implementativa Django è stata divisa in tre applicazioni, inclusa l'applicazione iniziale del progetto. Come già accennato in precedenza, le *views* sono opportunamente progettate per garantire una corretta differenziazione di permessi e autorizzazioni delle categorie di utenti. Non si tratta solo, difatti, di una strategia implementativa basata sulla selettività di funzioni mostrate/nascoste a livello frontend, ma anche caratterizzata da protezione a livello backend.

Di seguito è riportato un breve excursus del ruolo ricoperto da ogni applicazione

- **TechWebProject**: applicazione di default di Django, qui delegata a gestire i processi di sign-in, sign-up, login e logout degli utenti, e della modifica del profilo tramite form.
- **Movieapp**: app che gestisce tutta la logica implementativa che orchestra titoli e relative richieste, oltre alle funzionalità di ricerca globale sul catalogo.

Nel dettaglio:

- **MovieListView** per la visualizzazione del catalogo completo di titoli, con paginazione.
- **SearchMovieListView** (eredita dalla view precedente), mostra i risultati di ricerca.
- **MovieAutocomplete**, fornisce funzionalità di suggerimento durante la digitazione per una ricerca, risponde a richieste GET effettuate dalla componente Ajax del template principale “base.html”

- MovieDetailView mostra i dettagli di un titolo e la sezione di titoli consigliati proposti dal Recommendation System.
- create/remove_request_ajax sono views complementari per la creazione/rimozione di richieste di un titolo, da parte di un utente, nella tabella Request, sempre da richieste ajax.
- add_movie_to_watchlist, permette di inserire elementi di Movie nella watchlist personale. Come le due views precedenti, è richiamata lato frontend con Ajax.
- **userdashboard:** applicazione che si occupa di implementare tutte le views rivolte a un utilizzo per-user, cui un utente ha accesso una volta effettuato il login. In particolare:
 - user_dashboard fornisce un semplice pannello stile “pagina utente” dove un utente può vedere la sua immagine profilo, modificare i dati di quest’ultimo, o accedere ad altre pagine personali.
 - my_watchlist implementa la visualizzazione della watchlist personale.
 - MyRequests propone una pagina in cui visualizzare le proprie richieste.
 - ModeratorDashboard è una view esclusiva per gli utenti moderatori o amministratori, da cui è possibile vedere tutte le richieste effettuate dagli utenti raggruppate per titolo.

Da qui un utente addetto può non solo approvare o rifiutare le richieste, ma anche marcare un titolo come definitivamente disponibile per lo streaming.

Il raggruppamento per titolo, inoltre, svolge implicitamente una funzione piuttosto rilevante, ossia quella di mostrare l’indice di interesse della *user base* nei confronti di un certo film/serie tv: un alto numero di richieste indica abbastanza realisticamente un alto interesse, e quindi potrebbe suggerire la necessità di assegnare una priorità maggiore al reperimento del titolo in questione rispetto ad altri.

L’altra funzione offerta dalla view è l’aggiunta di titoli a catalogo tramite l’inserimento del solo ID di TMDB, evitando la compilazione manuale di tutti i campi della tabella Movie. L’operazione è svolta dal modulo movie_search_API.
 - manage_requested_title gestire richieste Ajax per l’approvazione/rifiuto di titoli dalla view ModeratorDashboard.
 - add_title permette di aggiungere, come già accennato, un titolo al catalogo, attingendo dal DB di TMDB tramite API request.

Creazione User-Profile: Django Signals

Come si può notare dal diagramma delle classi, è presente una relazione one-to-one tra i modelli User e Profile. Avvenuta la registrazione di un nuovo utente, che sia esso registered o Moderator (o amministratore), vorremmo quindi che alla nuova entry della tabella User appena creata corrispondesse la creazione di una entry anche nella tabella Profile: questo comportamento è possibile grazie all'utilizzo di **Django Signals**.

I Django Signals sono un meccanismo che permette di effettuare operazioni arbitrarie ad avvenuta creazione/modifica di una specifica entry del DB: nel nostro caso, chiameremo la creazione e il successivo salvataggio di una nuova entry per la tabella Profile ad avvenuta creazione di un nuovo elemento di User, con cui persisterà una relazione one-to-one.

Funzionalità di ricerca

Come da obiettivo, il programma offre una funzionalità di **ricerca globale** che opera query sulla tabella Movie, nei campi titolo e descrizione, e restituisce titoli anche se la query fornita dall'utente non fa match perfetto con uno dei campi, proponendo dei **suggerimenti simili**.

Il risultato è ottenuto abbinando una query resa case insensitive all'utilizzo di una funzione apposita che sfrutta il modulo **difflib** per calcolare la distanza sintattica tra la stringa inserita dall'utente e tutti i titoli presenti a catalogo, misurata dalla funzione “.ratio()” che restituisce un valore numerico compreso tra zero e uno.

Di seguito è riportata la funzione che implementa il calcolo della similarità e restituisce una lista di titoli simili, il cui funzionamento risulta immediato:

```
def find_similar_movies(movie_titles: list, query):
    tolerance = 0.6    similar_movies = []
    for s in movie_titles:
        s_ratio = difflib.SequenceMatcher(None, query, s).ratio()
        if s_ratio >= tolerance:
            similar_movies.append(s)
    return similar_movies
```

Come si può vedere la similarità restituita viene confrontata con un valore di tolleranza, impostato a 0.6 dopo numerosi test, per affinare la selezione dei risultati.

La ricerca è abbinata a una utility di **auto-completamento**, che scongiura possibili errori da parte dell'utente, e sfrutta anch'essa la funzione sopracitata per proporre suggerimenti anche in caso di mancato match.

Recommendation System

Si propone un sistema di raccomandazione semplice ma efficace e ampiamente utilizzato anche in servizi di streaming/media request molto più blasonati: ci si basa sui generi televisivi del titolo di cui si stanno visualizzando i dettagli per proporre altri cinque titoli il più affini possibili a quello preso in considerazione.

In particolare, essendo il campo “genre” del modello Movie una lista di uno o più generi, si cercano i titoli con più generi possibili in comune con il titolo selezionato, e si ordinano i risultati in ordine decrescente.

Se un titolo non ha affinità riscontrabili con almeno 5 elementi, ne verranno proposti in numero minore, di fatto escludendo “falsi positivi” in cui il numero di generi affini potrebbe essere pari a 0.

Unit Testing

Si sono create, nella applicazione “movieapp”, alcune classi dedicate allo unit testing mirato e completo di alcune funzionalità del programma.

In particolare è stato scelto di fare testing per la funzionalità di richiesta di un titolo da parte di un utente, e per la funzione di retrieve dei poster dei film tramite API TMDB.

Problemi riscontrati

A progetto ultimato, si riscontrano due problemi concreti su funzionalità la cui implementazione non è stata possibile, ma che potrebbe essere utile risolvere in futuro:

- **Meccanismo di caching di poster del catalogo:** sebbene la scelta di evitare il salvataggio di un elevato numero di poster ad alta risoluzione su supporto fisico presenti numerosi vantaggi, il caricamento delle pagine potrebbe risultare rallentato, in quanto si deve attendere il *retrieve* delle immagini direttamente dai server di TMDB.
Questo problema si potrebbe risolvere con un semplice meccanismo di caching: si effettua il download in locale di un dato numero di poster, appena un poster non presente in memoria viene richiesto, esso verrà scaricato e salvato, mentre l'immagine con la data di utilizzo meno recente verrà eliminata. Purtroppo non è stato possibile implementare questa funzionalità per mancanza di tempo pratico.
- **Ordinamento dei film in base alla data delle richieste:** per il momento non si è riusciti a ordinare gli elementi della pagina “Moderator Dashboard” in modo che i titoli con richieste più recenti vengano mostrati per primi, a causa di un'elevata complessità data dalla struttura adottata per modellare il DB.

Risultato finale

Home Page: differenza tra utente loggato e anonimo

Requestarr

Watchlist

My Requests

Mod Dashboard

Search for a movie...

Search Q

mod - Moderator

All Movies

Godzilla x Kong: The ...

Following their explosive showdown, Godzilla and Kong must reunite against ...

+

Add to Watchlist

Grizzly Man - 2005

Werner Herzog's documentary film about the "Grizzly Man" Timothy...

+

Remove from Watchlist

IF - 2024

A young girl who goes through a difficult experience begins to see everyone's...

+

Inside Out - 2015

When 11-year-old Riley moves to a new city, her Emotions team up to help h...

+

Requestarr

Search for a movie...

Search Q

Anonymous User

All Movies

Godzilla x Kong: The ...

Following their explosive showdown, Godzilla and Kong must reunite against ...

+

Grizzly Man - 2005

Werner Herzog's documentary film about the "Grizzly Man" Timothy...

+

IF - 2024

A young girl who goes through a difficult experience begins to see everyone's...

+

Inside Out - 2015

When 11-year-old Riley moves to a new city, her Emotions team up to help h...

+

Inside Out 2 - 2024

Teenager Riley's mind headquarters is undergoing a sudden demolition to make...

+

Add to Watchlist

Jaws of the Jungle - ...

Ceylonese natives are forced to flee into the dangerous jungle after an attack by...

+

Add to Watchlist

Kung Fu Panda 4 - 2...

Po is gearing up to become the spiritual leader of his Valley of Peace, but also...

+

Add to Watchlist

M*A*S*H - 1970

The staff of a Korean War field hospital use humor and hijinks to keep their sanity l...

+

Add to Watchlist

Inside Out 2 - 2024

Teenager Riley's mind headquarters is undergoing a sudden demolition to make...

+

Jaws of the Jungle - ...

Ceylonese natives are forced to flee into the dangerous jungle after an attack by...

+

Kingdom of the Plane...

Several generations in the future following Caesar's reign, apes are now the...

+

Kleeks Academy - 2024

A seemingly ordinary girl finds her way into the eponymous Academy to explore the wo...

+

«

«

1

2

3

»

Made by Lorenzo Taccini, built with Django © 2024

All Movies

Kung Fu Panda 4 - 2...

Po is gearing up to become the spiritual leader of his Valley of Peace, but also...

+

M*A*S*H - 1970

The staff of a Korean War field hospital use humor and hijinks to keep their sanity l...

+

«

«

1


2

3

»

Made by Lorenzo Taccini, built with Django © 2024

Pagina con dettagli del titolo

 Requestarr [Watchlist](#) [My Requests](#)

[chiara - Registered](#)

[< Back](#)

i Your request status is: Request not yet evaluated



Godzilla x...

Following their explosive showdown,...



Kingdom ...

Several generations in the future...



Alienoid: ...

Ean has a critical mission to return to t...



Atlas - 2024

A brilliant counterterroris m analyst wit...

Made by Lorenzo Taccini, built with Django © 2024

Ricerca con auto completamento e suggestions

Inside Out (2015)

Inside Out 2 (2024)

Moderator Dashboard

Requestarr

WatchlistMy RequestsMod Dashboard

Search for a movie...Search

mod - Moderator

Add New Movie

Insert TMDB ID here

Add

Movies with Requests

A Quiet Place: Day One

Now Available

Reject

Production Year: 2024

Requested by:

chrium

Request has been approved

6 July 2024

manumod

Request has been approved

6 July 2024

Alienoid: Return to the Future

Mark as Available

Approve

Production Year: 2024

Requested by:

chiara

Request has been rejected

6 July 2024

lollo

Request has been rejected

6 July 2024

Atlas

Mark as Available

Reject

Production Year: 2024

Requested by:

lollo

Request has been approved

6 July 2024

Civil War

Mark as Available

Approve

Production Year: 2024

Requested by:

annalisa

Request has been rejected

6 July 2024

lollo

Request has been rejected

4 July 2024

chiara

Request has been rejected

2 July 2024

Das Boot

Mark as Available

Approve

Reject

Production Year: 1981

Requested by:

lollo

Request not yet evaluated

4 July 2024

123»»

Made by Lorenzo Taccini, built with Django © 2024

Watchlist


Requestarr

WatchlistMy Requests

Search for a movie...Search

chiara - Registered


Your Watchlist



A Quiet Place: Day One - 2024

As New York City is invaded by alien creatures who hunt by sound, a woman named Sammy fights to survive.


Remove from Watchlist



Atlas - 2024

A brilliant counterterrorism analyst with a deep distrust of AI discovers it might be her only hope when a mission to capture a renegade robot goes awry.


Remove from Watchlist



Inside Out 2 - 2024

Teenager Riley's mind headquarters is undergoing a sudden demolition to make room for something entirely unexpected: new Emotions! Joy, Sadness, Anger, Fea...


Remove from Watchlist



MR-9: Do or Die - 2023

Masud Rana is a Secret Agent with code name MR-9 of the Bangladesh Counter Intelligence Agency

Remove from Watchlist



The Last Kumite - 2024

When Karate champion Michael Rivers wins the last tournament of his career, shady businessman Ron Hall offers him the opportunity to fight in an illegal Kumite i...

Remove from Watchlist

12»»

Made by Lorenzo Taccini, built with Django © 2024

16