Progetto Tecnologie Web

Traccia del progetto:

1. I gestori della piattaforma (registrati come utente admin) possono inserire e rimuovere film (locandina, titolo, trama, data di uscita, durata), modificarne i prezzi ed eliminare recensioni con linguaggio inopportuno

2. Utenti anonimi possono navigare la piattaforma e visualizzare i film con le recensioni, le votazioni degli altri utenti e il prezzo del film. Non possono lasciare recensioni, commenti e ovviamente non possono acquistare i film.

3. Un utente, previa registrazione al sito, può decidere se stipulare un abbonamento (mensile/annuale) oppure acquistare il singolo film che visualizzerà nella

4. Un utente registrato senza abbonamento non può lasciare recensioni o valutare il film, però può lasciare un piccolo commento sotto al film.

6. Un utente abbonato può lasciare delle recensioni per il film e ha accesso libero al catalogo.

7. Meccanismo di voto da parte degli utenti abbonati basato su valori numerici (es. voto da 1 a 5). Il voto assegnato al film verrà visualizzato nella pagina relativa, inoltre sarà data la possibilità di ordinare i film disponibili in ordine di giudizio degli utenti (in alternativa al prezzo). È stato creato anche un sistema di filtraggio basato sui generi dei film.

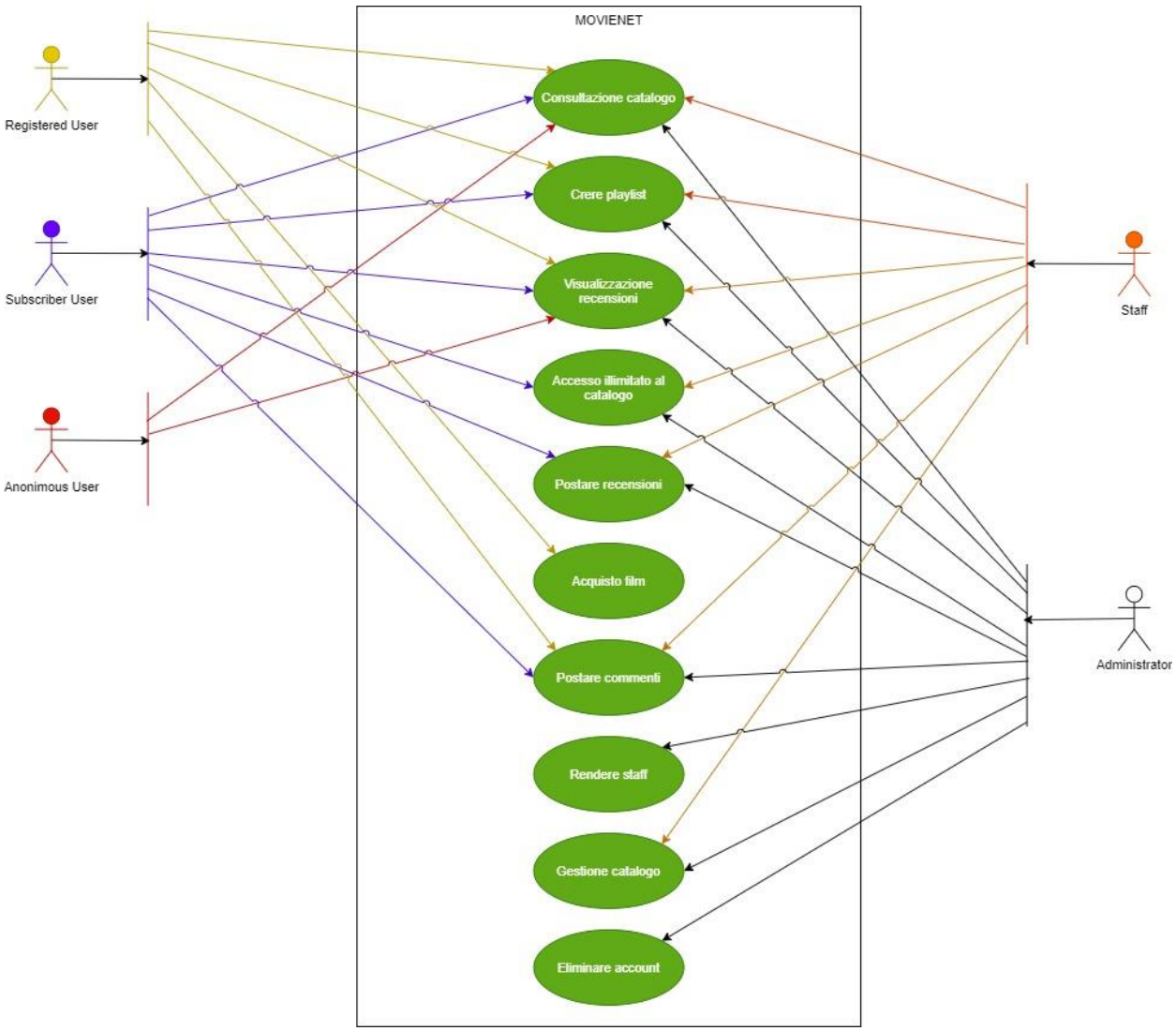
8. Il recommendation system tiene traccia dei film visti dall'utente e di conseguenza restituisce una lista di film che ancora non ha acquistato del suo genere preferito.

9. Funzionalità playlist: l'utente può creare una o più playlist assegnando loro un nome e può successivamente aggiungere dei film all'interno delle stesse

10. Funzionalità forum del film: tale funzionalità permette agli utenti di pubblicare commenti relativamente al film selezionato e rispondere ai commenti di altri utenti. Necessario tag "spoiler", ovvero un apposito tag da utilizzare per marcare un commento come potenziale spoiler. L'interfaccia consente l'utilizzo del filtro "mostra/nascondi spoiler" in modo tale da permettere all'utente di scegliere (e memorizzare la scelta) se visionare i contenuti marcati come spoiler o meno.

USE-CASE

Movienet è una piattaforma di streaming online pensata sia per gli appassionati di cinema e sia per gli spettatori occasionali. Di seguito verranno mostrati vari diagrammi che esplicheranno ciò che è possibile fare con l'applicazione.



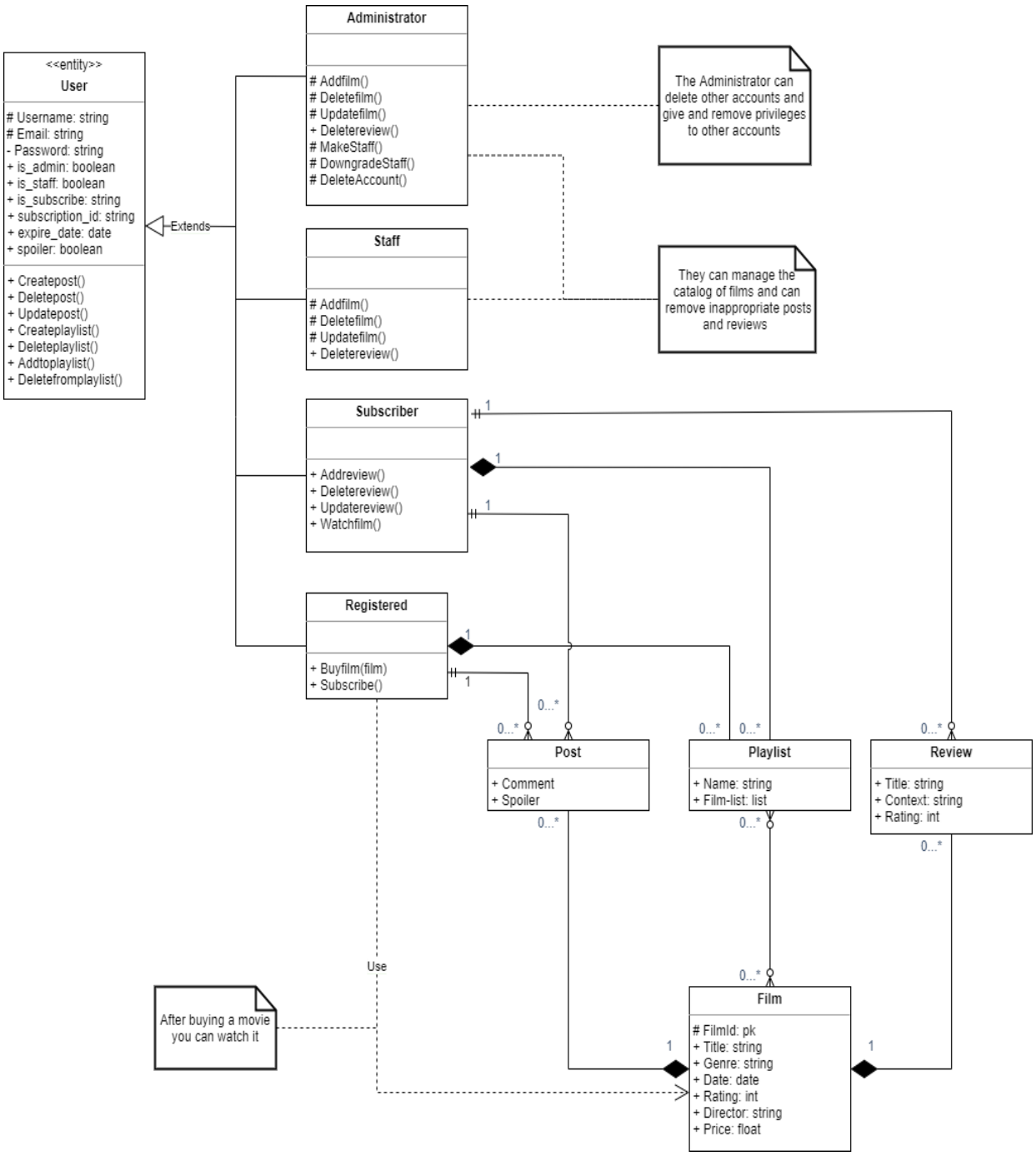
Lo schema mostra quali azioni si possono svolgere sulla piattaforma evidenziando le differenze dei diversi utenti. Possiamo notare come ci sono vari tipi di utenti con diversi privilegi. L'utente anonimo può soltanto navigare la piattaforma e visualizzare i film con le relative recensioni. All'utente registrato con il piano gratuito viene data la possibilità di acquistare i singoli film, partecipare in modo attivo commentando e rispondendo alle varie discussioni create nella sezione commenti. Un'altra funzionalità garantita all'utente dopo la semplice registrazione è la possibilità di creare delle playlist personalizzate dando loro un nome ed inserendo i propri film preferiti. L'utente abbonato, oltre ai privilegi riservati agli utenti registrati con il piano gratuito, può godere di un'esperienza completa

grazie all’accesso illimitato al catalogo e alla possibilità di creare contenuti inediti sulla piattaforma tramite le proprie recensioni.

Un utente appartenente allo staff gode di tutti i privilegi sopra citati e in più può gestire il catalogo aggiungendo, aggiornando e rimuovendo film e rimuovere post o recensioni inopportune. Infine, l'utente amministratore è l'utente a capo dello staff e detiene i privilegi per rendere un utente parte dello staff, “licenziare” un utente dallo staff ed infine eliminare utenti dalla piattaforma

DIAGRAMMA DELLE CLASSI

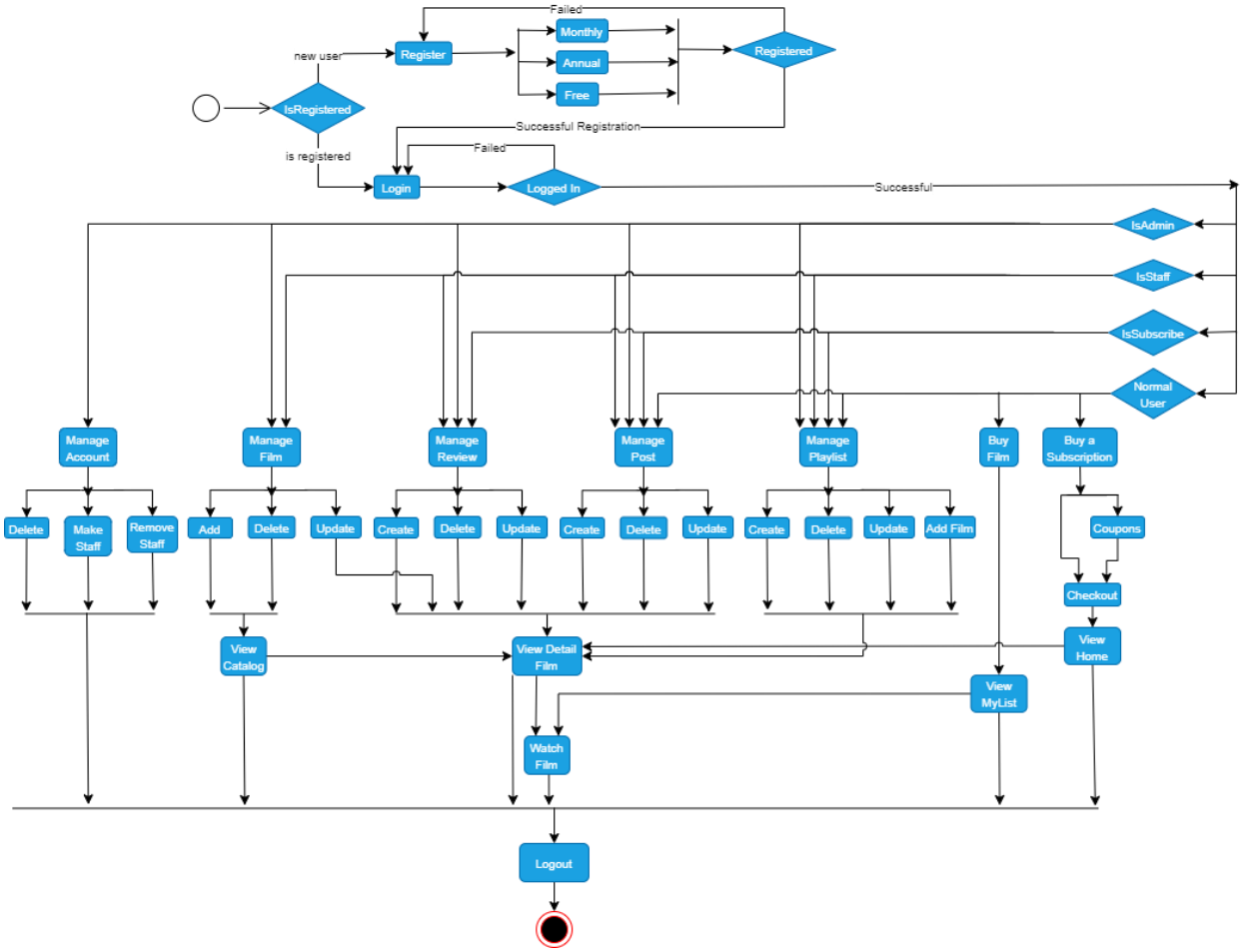
Tramite il diagramma delle classi vengono illustrate le diverse entità con le relative caratteristiche e le loro relazioni. Di seguito vengono mostrate nel dettaglio la struttura del sito mostrando le varie relazioni e le varie azioni che si possono svolgere.



ACTIVITY DIAGRAM

Dopo aver mostrato i collegamenti e la struttura del progetto continuiamo con l'activity diagram, analizzando come viene gestita una qualsiasi azione fatta sulla piattaforma e mostrando le possibili conseguenze che una data azione può avere.

Seguendo il flusso, che indica la sequenza temporale con cui devono essere effettuate le diverse attività, è possibile capire cosa accadrà e cosa ci si dovrebbe aspettare come risposta.



Tecnologie utilizzate

Per la realizzazione del progetto abbiamo fatto affidamento su Django, il quale permette di gestire la piattaforma lato back-end.

Con Django abbiamo modellato tutta la parte logica del progetto andando a definire i vari modelli e le varie applicazioni.

Esso permette di creare applicazioni web in modo molto veloce seguendo il principio DRY (Don't Repeat Yourself) inoltre sfrutta il paradigma MTV (Model-Template-View). Il Model, basato su Object-relational database, serve a rappresentare la collezione di classi Python direttamente sulle tabelle del nostro database. Le View in Django vanno a gestire tutta la logica della nostra applicazione e vanno ad interfacciarsi ai dati dei modelli grazie all'API Python. Infine, i Template ci garantiscono la separazione logica (View) dalla rappresentazione (Template).

Un altro aspetto da non sottovalutare quando si va a scegliere il Framework da utilizzare per lo sviluppo della propria applicazione è la messa in sicurezza della stessa. Infatti, Django permette di proteggersi da molti attacchi (SQL Injection, XSS, CSRF, Clickjacking, ecc.) in modo davvero semplice.

Per la realizzazione delle pagine web abbiamo fatto affidamento soprattutto su Bootstrap, il quale mette a disposizione vari fogli di stile e diversi Template da cui partire per creare delle pagine web responsive.

Il database utilizzato per la nostra applicazione è SQLite3, che viene impostato di default direttamente da Django.

Organizzazione Logica

Seguendo uno dei princìpi di Django ("pluggability") siamo andati a suddividere i vari componenti del nostro progetto in diverse app. La suddivisione creata e data da quattro app che sono: Account, Movie, Playlist, Post.

Questo tipo di suddivisione è stata scelta in quanto, l'app Account va a modellare tutto quello che concerne l'utente; l'app Movie modella il prodotto del nostro sito web; la Playlist è una struttura dati facilmente riutilizzabile in altri contesti; l'app Post ci consente di modellare la sezione commenti del nostro applicativo in modo quasi del tutto indipendente dal progetto.

Ognuna di queste quattro app con qualche lieve modifica è facilmente riutilizzabile in moltissimi altri ambiti. Ad esempio, se si volesse modellare una biblioteca online nella quale comprare degli e-book si potrebbe far uso dell'app Account nella sua interezza e dell'app Movie applicandole qualche lieve modifica, in questo modo si avrebbe una modellazione perfettamente coerente con l'ambito studiato (lo stesso vale per l'app Post e Playlist).

Per quanto concerne le recensioni si è deciso di non creare un'altra app in quanto loro sono strettamente legate ai Film o più in astratto ai prodotti.

Uno dei vantaggi di questa scelta implementativa è che installando l'app Movie su un altro progetto, si usufruirebbe della possibilità di definire delle recensioni per il prodotto che si andrebbe a modellare.

Recommendation System

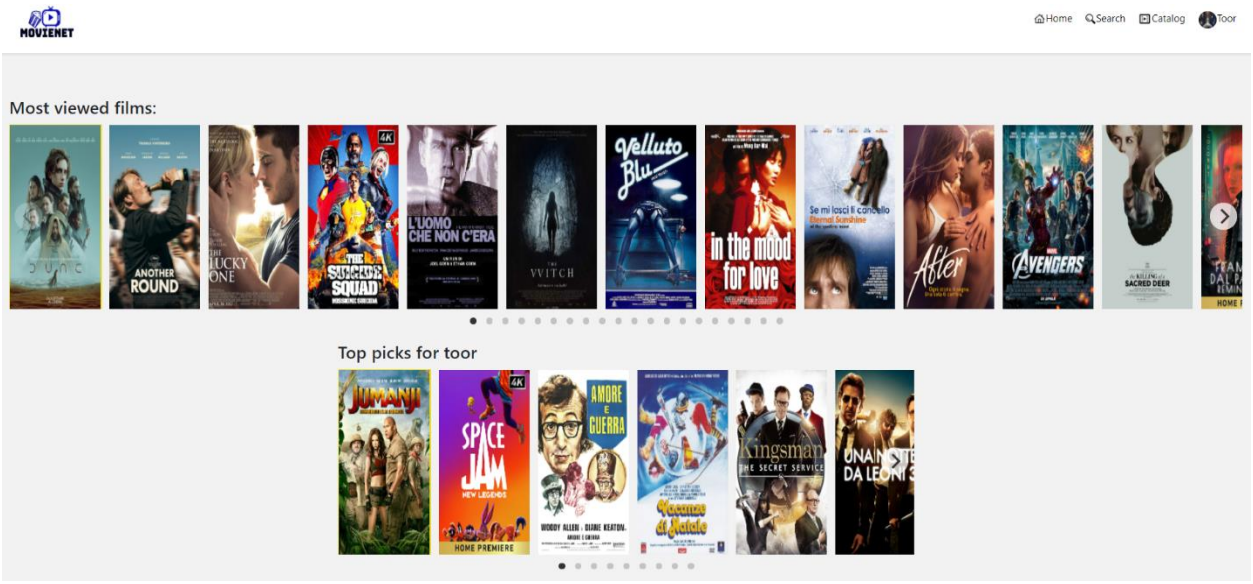
L'algoritmo di recommendation system implementato nel progetto si basa sull'approccio *content-based- filtering.* Questo approccio si basa sui passati acquisti dell'utente e ritorna gli articoli più simili a quelli acquistati. Nel nostro caso specifico il recommendation system è basato sul genere preferito dell'utente (per genere preferito si intende il genere di film più acquistato) in modo da consigliarli i film simili non ancora acquistati.

Ci sono vari vantaggi nell'utilizzare l'approccio content-based, il principale vantaggio risiede nel non risentire del cosiddetto *cold start problem*. Infatti, nel nostro caso, il recommendation system riesce a funzionare anche su un utente nuovo che acquista un singolo film. Invece, lo svantaggio principale del nostro algoritmo è il non tener conto degli acquisti e delle azioni degli altri utenti.

L'unico campo che viene preso in considerazione nell'algoritmo è il numero di visualizzazioni che riceve il film. In questo modo l'algoritmo è in grado di ritornare non solo i film del genere preferito ma bensì, i film del genere preferito che hanno raggiunto il maggior numero di visualizzazioni sulla piattaforma.

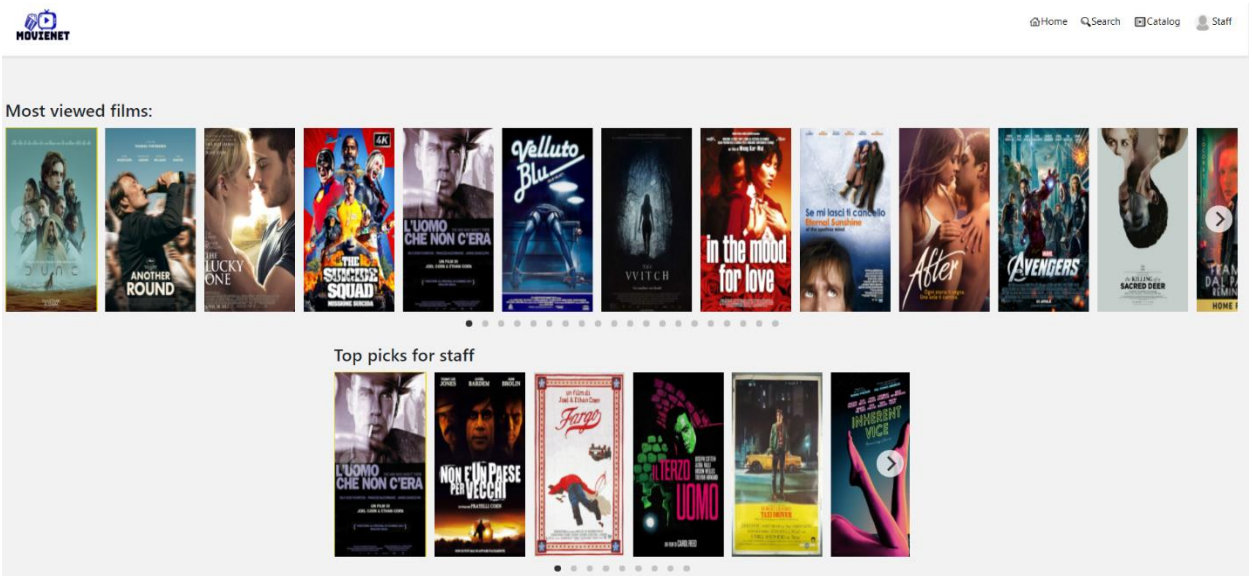
Con questo semplice algoritmo si riesce a personalizzare l'esperienza d'uso in base ai gusti personali di ogni singolo utente.

Risultati finali



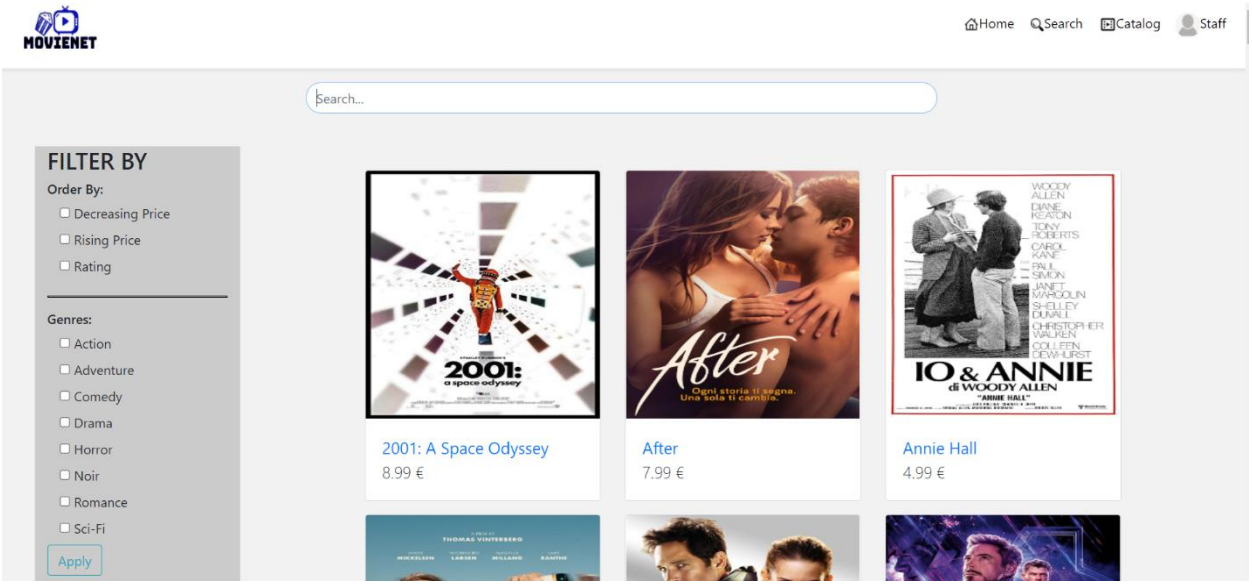
L’immagine in alto mostra la pagina principale della nostra applicazione, e in questo caso specifico mostra come la vista viene personalizzata per l’utente Toor.

Da notare come nel primo carrello vengono mostrati i film più visti sulla piattaforma e nel secondo quelli calcolati dall’algoritmo di recommendation system.



Con questa seconda immagine si vuol soffermare l’attenzione sui diversi film suggeriti all’utente Staff rispetto all’utente Toor, inoltre si può notare come anche l’ordine dei film più visti sia cambiato (causato ovviamente dalla visione del film da parte di un utente),

infatti, nella seconda immagine Velluto Blu di David Lynch ha superato The Witch di Robert Eggers.



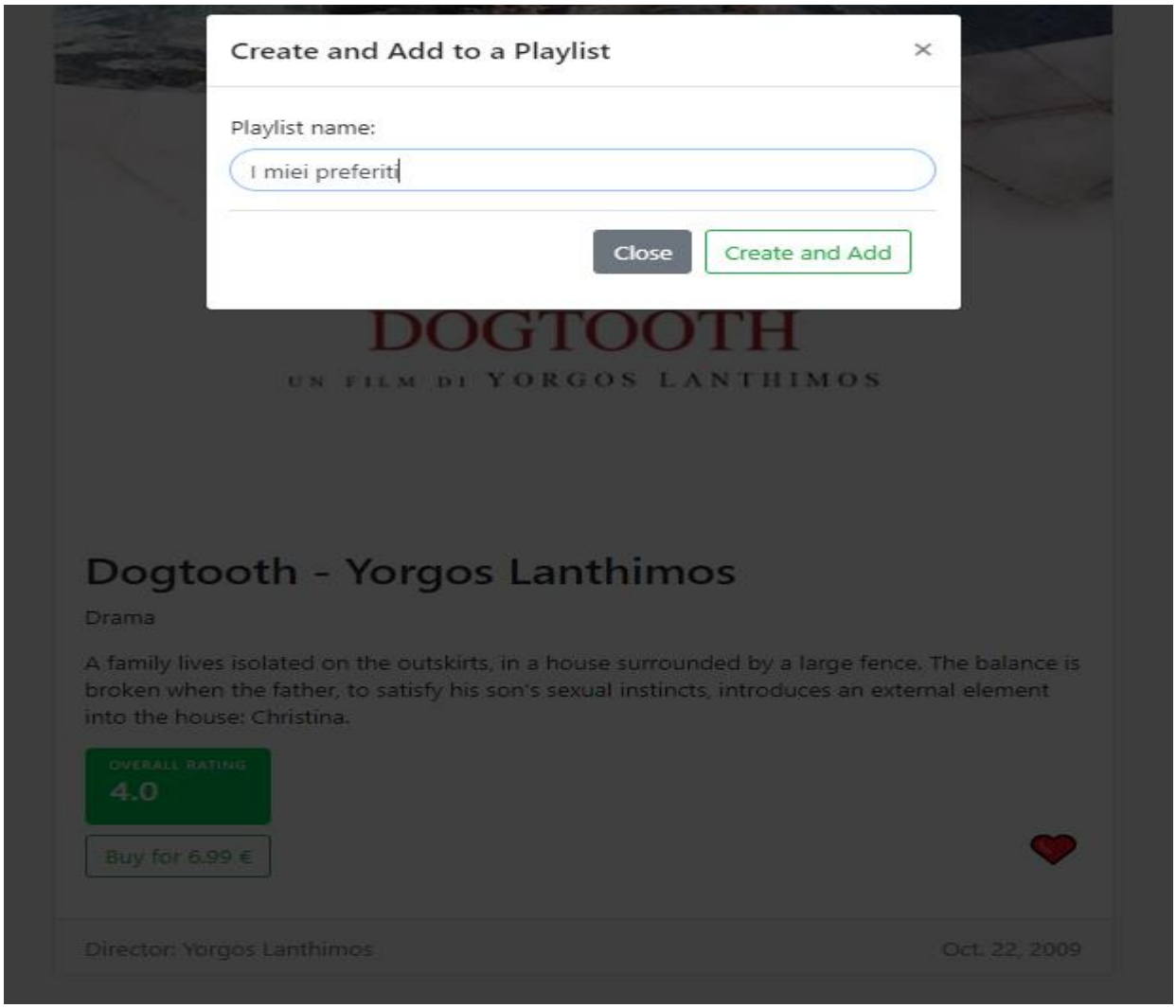
Nella pagina relativa al catalogo di default i film vengono mostrati in ordine alfabetico. Per ottimizzare la visualizzazione e migliorare la ricerca abbiamo aggiunto una sezione che permette di applicare vari filtri ed anche una barra di ricerca con cui cercare film specifici. Oltre alla scelta dei generi da mostrare è possibile anche decidere l'ordinamento generale dei risultati. Per esempio, è possibile ordinare i film per prezzo (crescente/decrescente) oppure è possibile ordinarli per gradimento.

In sovrimpressione è mostrata la sezione commenti che è possibile trovare in fondo alla pagina di ogni singolo film.

La check box che nell'immagine risulta spuntata consente di mostrare i commenti segnalati come spoiler. La scelta sarà salvata in modo tale da non dover scegliere ogni volta se mostrare gli spoiler o meno. Per contenere la grandezza della pagina si è scelto di tenere la sezione commenti nascosta, la quale diverrà visibile cliccando sulla scritta *'Show Comments'*.

Per i cinefili avere uno spazio dove esporre le proprie idee è fondamentale.

Per venire incontro a questa esigenza è stato creato un modello che implementa le review. Questa funzionalità spetta solo agli abbonati, che possono scrivere la recensione e valutare il film, valutazione che verrà usata per calcolare il voto medio del film calcolando la media dei voti assegnati dagli abbonati nelle loro recensioni.



Visitando la pagina dedicata al singolo film e cliccando sul cuore è possibile inserire il film tra i preferiti creando così delle playlist personalizzate. Ovviamente è possibile creare più playlist, le quali saranno visibili in una pagina apposita. Nella suddetta pagina è data la possibilità di creare o eliminare le playlist e di rimuovere i film (dalle proprie playlist).

Test Effettuati

Sono stati effettuati test sugli URL dell’app Account.

Coverage for **account\urls.py**: 100%

6 statements | 6 run | 0 missing | 0 excluded

```python
from django.conf import settings
from django.urls import path

from account.decorator import admin_required
from account.views import PricingView, RegisterUser, LogoutUser, LoginUser, UpdateUser, MustAuthenticate, CheckoutView, \
    PasswordChangeViewP, DeleteAccountView, ManageAccount, make_staff, downgrade, ImgView

app_name = 'account'

urlpatterns = [

    path('pricing/', PricingView.as_view(), name='pricing'),
    path('register/', RegisterUser.as_view(), name='register'),
    path('logout/', LogoutUser.as_view(), {'next_page': settings.LOGOUT_REDIRECT_URL}, name='logout'),
    path('login/', LoginUser.as_view(), name='login'),
    path('', UpdateUser.as_view(), name='account'),
    path('must_authenticate/', MustAuthenticate.as_view(), name='must_authenticate'),
    path('manage_account/', admin_required(ManageAccount.as_view()), name='manage_account'),
    path('manage_img/<int:pk>', ImgView.as_view(), name='manage_img'),
    path('makestaff/<int:pk>', make_staff, name='make_staff'),
    path('downgrade/<int:pk>', downgrade, name='downgrade'),
    path('checkout/', CheckoutView.as_view(), name='checkout'),
    path('<int:pk>/delete', DeleteAccountView.as_view(), name='delete'),
    path('password_change/', PasswordChangeViewP.as_view(template_name='registration/password_change.html'),
         name='password_change'),
]
```

Inoltre, sono stati effettuati dei test su delle View dell'app Account.

account\views.py: 33% | 81 | 165 | 0

```python
class LogoutUser(LoginRequiredMixin, LogoutView):

    def get_next_page(self):

        next_page = super(LogoutUser, self).get_next_page()

        messages.add_message(
            self.request, messages.SUCCESS,
            'You successfully log out!'
        )
        return next_page


class LoginUser(LoginView):

    form_class = AccountAuthenticationForm
    template_name = 'account/login.html'
    success_url = reverse_lazy('home')

    def get(self, request, *args, **kwargs):
        try:
            self.success_url = request.GET['next']
        except:
            pass

        return super(LoginUser, self).get(request, *args, **kwargs)
```

In particolare, ci siamo soffermati su due View: la LoginView e la LogoutView. Abbiamo testato il funzionamento delle viste provando ad invocarle in diverse condizioni, essendo delle view a cui è possibile accedere solo se si è effettuato il login, abbiamo voluto verificare il funzionamento sia nel caso che la richiesta venga fatta da un utente registrato sia che venga effettuata una richiesta anonima (utente non registrato). Inoltre, abbiamo controllato che lo status_code fosse quello atteso e che il content fosse quello atteso.

Problemi riscontrati

Le difficoltà riscontrate sono dovute alla progettazione dei vari modelli e alle loro relazioni. Questa difficoltà crediamo sia derivata dalla poca esperienza nella progettazione e nella modellizzazione.

Inizialmente ridefinivamo sempre il metodo Dispatch() anche nei casi in cui non c'era nessun bisogno. Successivamente durante la progettazione ci siamo resi conto che spesso era sufficiente ridefinire i metodi Get() e Post().

Il modo di operare, soprattutto nelle fasi iniziali, andava a creare dei problemi con i valori inseriti nei form e con il salvataggio dei dati nei database.

Un altro problema riscontrato è stato capire come recuperare e passare i dati da visualizzare nei template. Dopo alcune ricerche ed aver compreso il modo corretto di operare è stato molto più semplice proseguire con la progettazione e di conseguenza velocizzare lo sviluppo.

Inoltre, nonostante l'utilizzo di Bootstrap abbiamo riscontrato dei piccoli problemi con l'HTML, soprattutto quando volevamo personalizzare i template offerti da Bootstrap. Una risorsa importante nella risoluzione di questi piccoli problemi è stata sicuramente W3schools.

Altri problemi sono stati riscontrati nel gestire ed effettuare il pagamento dei singoli film e dei vari abbonamenti.

Dopo alcune valutazioni abbiamo optato per affidarci a Stripe che mette a disposizione delle API con le quali interfacciarsi. In questo modo siamo stati in grado di gestire sia singoli acquisti, sia gli abbonamenti.

CONCLUSIONI

In conclusione, l’aver dovuto affrontare queste problematiche di diversa difficoltà ha avuto sicuramente un esito positivo sul nostro modo di approcciarci ai problemi. Ora sappiamo che alle volte è meglio spendere più tempo sulla progettazione e soffermarsi nel capire meglio determinati processi anziché iniziare subito con la scrittura del codice. A posteriori crediamo che l’aver sviluppato il progetto in due abbia avuto dei riscontri positivi su entrambi ma non neghiamo che ci sono stati dei momenti in cui abbiamo avuto delle opinioni divergenti sul come procedere. Nonostante questo, siamo sicuri che questo modo di lavorare ci sarà d’aiuto nel momento in cui andremo a far parte di un gruppo di lavoro.

Enzo Rotonda

Davis Innangi