Relazione sul progetto

Progetto svolto da: Alice Torsani

Matricola: 164915

1. Traccia e funzionalità

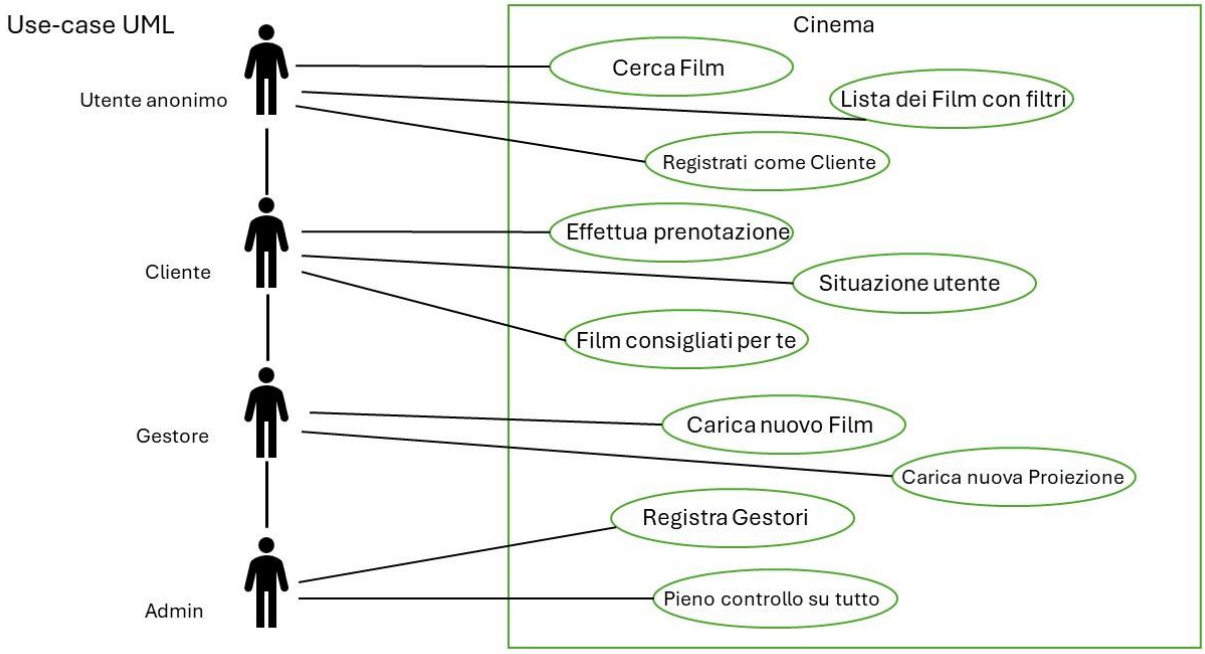
Gestione sistema di prenotazioni di un cinema

Sito web per la gestione di prenotazioni di un cinema. Il sistema deve offrire i seguenti servizi:

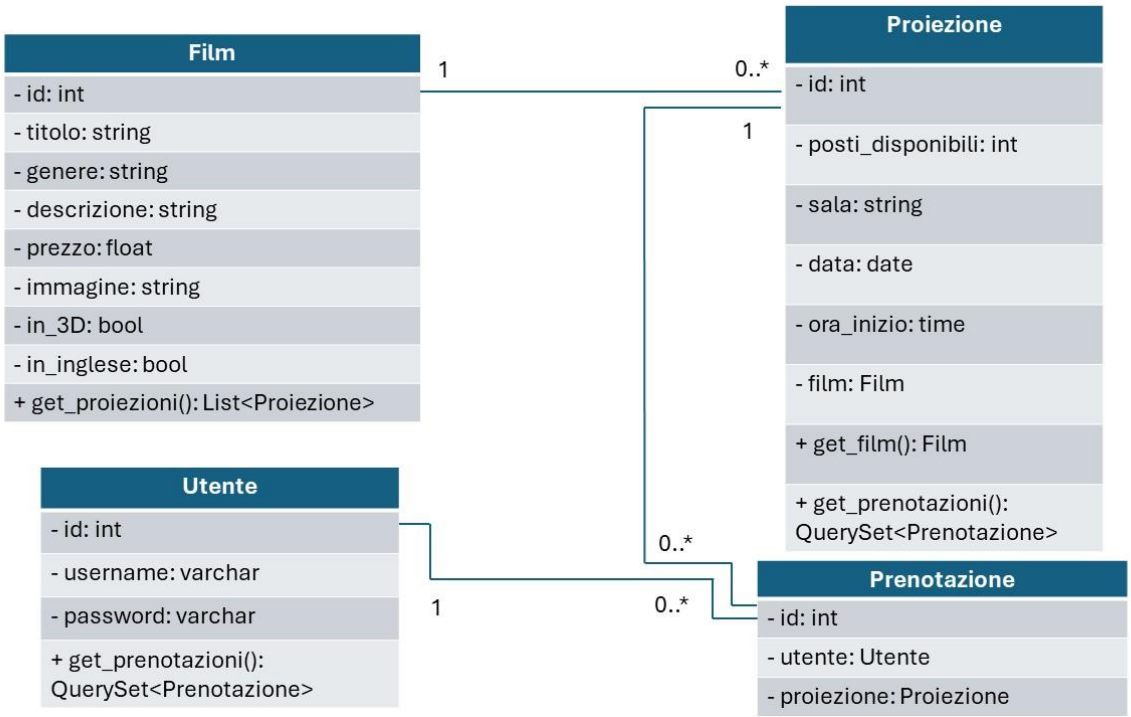
- Gli utenti anonimi possono navigare il sito e visualizzare i film, con le rispettive proiezioni disponibili, con relativa descrizione, prezzo e caratteristiche.
- Ogni utente può registrarsi come gestore o cliente.
- Ogni utente registrato come gestore deve poter caricare un nuovo film, inserire il titolo, un genere (intesa come categoria di film: ad esempio 'Fantasy', 'Azione', 'Fantascienza', ...), la descrizione del film (intesa come trama), il prezzo ed una immagine. Se il film inserito ha caratteristiche particolari (ad esempio se è in 3D o in inglese) queste caratteristiche devono essere inserite. Inoltre, il gestore deve poter inserire le date nelle quali il film sarà disponibile, e per ogni data, dovrà inserire gli orari di inizio di ogni proiezione prevista per quella data.
- Ogni utente registrato da cliente può effettuare una prenotazione per una proiezione di un film. Egli può scegliere il film nella lista dei film disponibili, dopodiché potrà selezionare la proiezione che desidera acquistare tra quelle disponibili selezionando l'orario d'inizio della proiezione scelta. Egli potrà visualizzare lo storico delle sue prenotazioni effettuate.
- I film disponibili possono essere visualizzati in base a un sistema di ricerca attraverso filtri, che permette di visualizzare i prodotti in base a diverse caratteristiche. Tra queste è possibile visualizzare i film filtrandoli in base al genere (esempio: elencare tutti i Fantasy), in base a caratteristiche particolari che possono caratterizzare alcuni film presenti (in particolare alcuni film possono essere in lingua inglese oppure in 3D; è quindi possibile richiedere al sistema di visualizzare quali film sono disponibili in 3D oppure quali siano disponibili in lingua inglese), in base ad una data selezionata dall'utente (l'utente inserisce la data che preferisce e viene visualizzata la lista dei film con le rispettive proiezioni disponibili previste per quella data).
- Gli utenti possono effettuare una ricerca per titolo: possono inserire il titolo del film desiderato e il sistema visualizzerà i giorni in cui il film è programmato, con le rispettive proiezioni per ogni giorno.
- Il sistema deve essere dotato di un sistema di recommendation basato su caratteristiche simili. Verranno consigliati film con caratteristiche simili a quelli acquistati precedentemente dall'utente.

2. Descrizione del progetto dell'applicazione

- Use case UML



- Diagramma delle classi



Note sul diagramma delle classi.

Nella classe **Proiezione** l'attributo **'film'** è un riferimento a un oggetto **Film** e rappresenta la chiave esterna che collega la proiezione al film a cui appartiene.

Nella classe **Prenotazione** gli attributi **'utente'** e **'proiezione'** fanno riferimento a un oggetto **Utente** e a un oggetto **Proiezione** e rappresentano le chiavi esterne che collegano la prenotazione all'utente che la ha effettuata e alla proiezione che è stata selezionata al momento della prenotazione.

Nella classe **Film** il metodo **get_proiezioni()** ritorna tutte le proiezioni associate a un film.

Nella classe **Proiezione** il metodo **get_prenotazioni()** ritorna tutte le prenotazioni associate ad una proiezione.

Nella classe **Utente** il metodo **get_prenotazioni()** ritorna tutte le prenotazioni associate ad un utente.

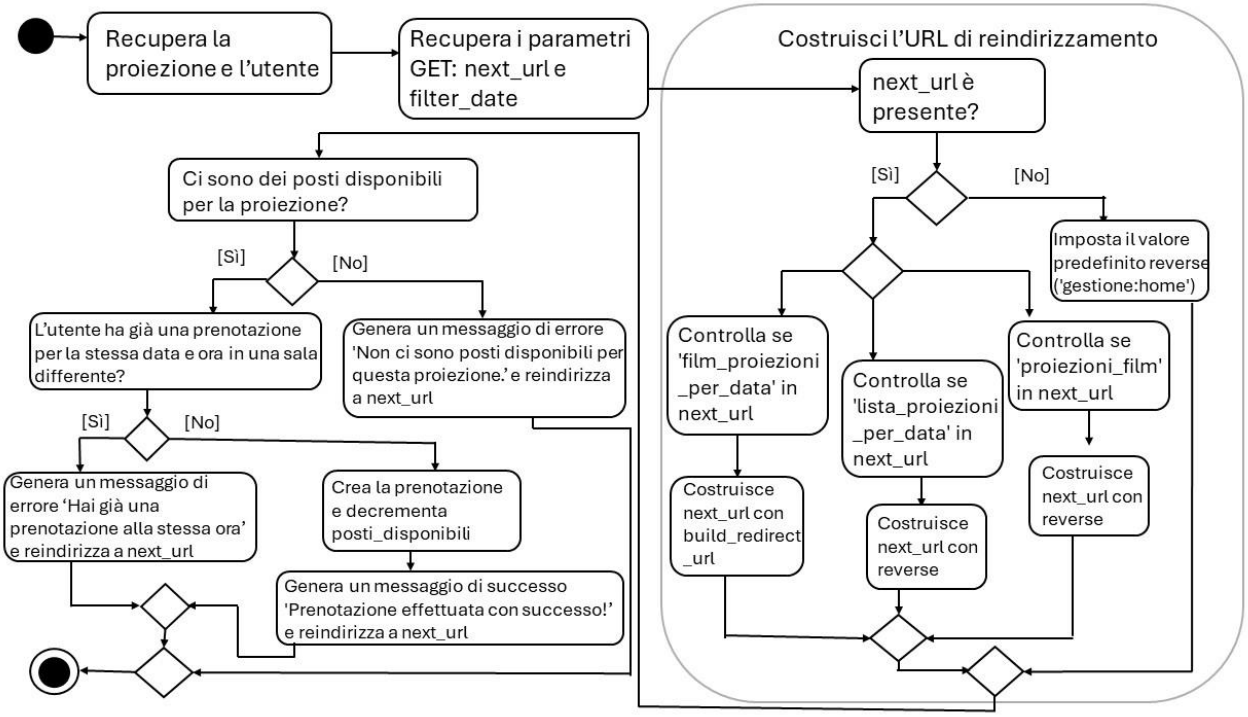
Descrizione cardinalità del diagramma delle classi:

Relazioni:

- Utente-Prenotazione:
  - Un utente può avere molte prenotazioni (0..*).
  - Ogni prenotazione è associata a un solo utente (1).
- Prenotazione-Proiezione:
  - Una proiezione può avere molte prenotazioni (0..*).
  - Ogni prenotazione è associata a una sola proiezione (1).
- Proiezione-Film:
  - Un film può avere molte proiezioni (0..*).
  - Ogni proiezione è associata a un solo film (1).

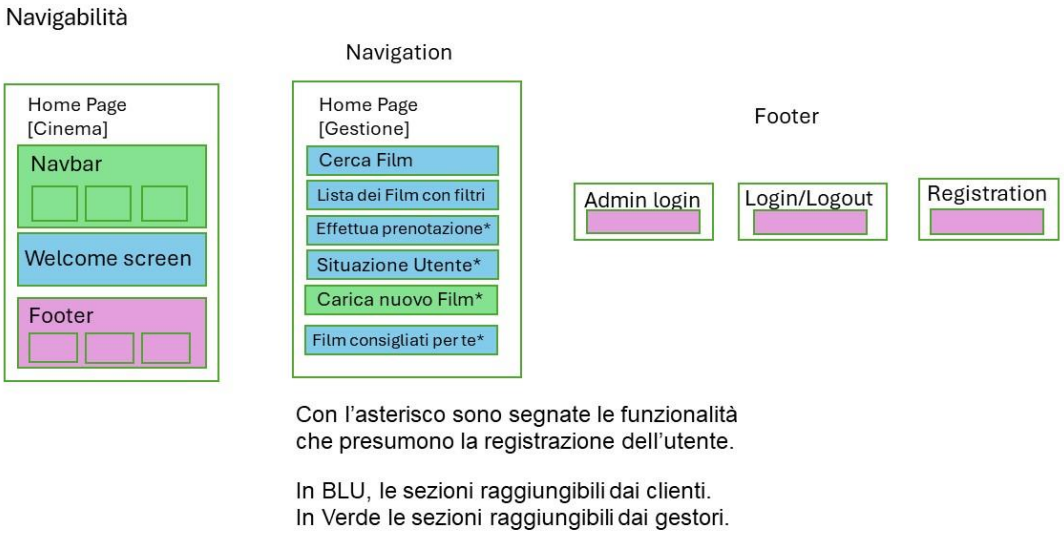
- Activity diagram

  Di seguito è riportato l'activity diagram che descrive la funzionalità "effettua prenotazione" dell'utente cliente (funzione *prenota_proiezione* in gestione/views.py).



- Schema di navigabilità

  Di seguito è riportato lo schema di navigabilità delle pagine principali del progetto



3. Tecnologie usate e motivazione

Librerie installate per il progetto:

- Pillow-10.3.0

  Pillow è una libreria di Python su cui Django fa affidamento per gestire i file immagine ed è necessario per utilizzare ImageField in Django. MEDIA_URL e MEDIA_ROOT sono stati configurati in settings.py per gestire i file media caricati dagli utenti e sono stati utilizzati in urls.py del progetto per gestire i file media durante lo sviluppo. ImageField nel modello Film utilizza Pillow per gestire i file immagine; quindi, per gestire correttamente le immagini caricate dagli utenti.
  Installata con: *pip install Pillow*.

- django-crispy-forms-2.2

  django-crispy-forms è una libreria di terze parti per Django che fornisce un modo semplice e potente per gestire la visualizzazione e la stilizzazione dei form. Ho scelto di utilizzarla per dare un aspetto più elegante e ben strutturato alla parte di presentazione del mio progetto dal punto di vista grafico. Ho configurato alcune impostazioni nel file settings.py: django-crispy-forms è stato aggiunto alle app installate (aggiunto ad INSTALLED_APPS) e ho Impostato il template pack di default (CRISPY_TEMPLATE_PACK = 'bootstrap4').
  Installata con: *pip install django-crispy-forms*.

- crispy-bootstrap4-2024.1

  crispy-bootstrap4 è un pacchetto per Django che integra il framework di form django-crispy-forms con Bootstrap 4. Questo pacchetto rende facile la

creazione di form ben strutturati e stilizzati utilizzando le classi di Bootstrap 4, senza dover scrivere manualmente il codice HTML per ciascun form. Ho scelto di utilizzarlo per dare un aspetto più elegante e ben strutturato alla parte di presentazione del mio progetto dal punto di vista grafico. Ho configurato alcune impostazioni nel file settings.py: crispy_bootstrap4 è stato aggiunto alle app installate (aggiunto ad INSTALLED_APPS) e ho Impostato il template pack di default (CRISPY_TEMPLATE_PACK = 'bootstrap4', CRISPY_ALLOWED_TEMPLATE_PACKS = "bootstrap4").
Installato con: *pip install crispy-bootstrap4*.

- django-braces-1.15.0
  django-braces è una libreria di terze parti per Django che fornisce una serie di mixins riutilizzabili, progettati per semplificare le operazioni comuni nelle view class-based di Django (CBV). La ho utilizzata per le view per i soli utenti gestori per verificare che gli utenti che tentavano di eseguire le operazioni associate a quelle view appartenessero al gruppo dei gestori e per negare tali funzionalità agli utenti che non fanno parte di tale gruppo. Le funzionalità principali che ho utilizzato sono state due mixins appartenenti agli Access Mixins: LoginRequiredMixin, che garantisce che l'utente sia autenticato prima di poter accedere alla view, e UserPassesTestMixin, che permette di definire una funzione di test personalizzata che deve restituire True affinché l'utente possa accedere alla view.
  Installato con: *pip install django-braces*.

4. Organizzazione logica a livello di codice

Parte dell'applicazione proposta è stata suddivisa nell'app '***gestione***'. Lo scopo di questa app è quello di contenere e presentare le funzionalità principali dedicate agli utenti. Per gli utenti anonimi essa offre le funzionalità di ricerca dei film per titolo e di visualizzare la lista dei film filtrata in base ai filtri disponibili. Per l'utente cliente essa offre anche le funzionalità di effettuare prenotazioni per le proiezioni, vedere la propria situazione (storico), e di visualizzare i film consigliati in base alle prenotazioni effettuate. Per l'utente gestore essa offre anche le funzioni di aggiunta di un nuovo film e di aggiunta di una nuova proiezione. Per l'utente admin essa offre in aggiunta la possibilità di iscrivere nuovi utenti gestori.

5. Scelte fatte sulle funzionalità implementate

Tipo di Recommendation System adottato

Il sistema di Recommendation System scelto è basato sulle prenotazioni effettuate in passato dall'utente. Esso è basato su caratteristiche simili: vengono consigliati film con caratteristiche simili a quelli acquistati precedentemente dall'utente.
Le caratteristiche considerate dal sistema per aggiornare i film consigliati in base ai film acquistati in precedenza sono il genere, il 3D se previsto e la lingua inglese se prevista.
Di seguito riporto l'algoritmo utilizzato per la funzionalità dei recommendation system con una spiegazione passo dopo passo dei passaggi che lo compongono.

Algoritmo

```python

    def get_film_consigliati(self):
        prenotazioni = list(self.prenotazioni.all())
        prenotazioni.reverse()  # Inverti l'ordine delle prenotazioni
        film_consigliati = []

        film_acquistati_ids = [prenotazione.proiezione.film.id for prenotazione in prenotazioni]

        caratteristiche_films = defaultdict(list)
        for prenotazione in prenotazioni:
            film = prenotazione.proiezione.film
            caratteristiche_films[film.genere].append(film)
            if film.in_3D:
                caratteristiche_films['3D'].append(film)
            if film.in_inglese:
                caratteristiche_films['inglese'].append(film)

        def get_similar_films(query, exclude_ids, limit):
            return Film.objects.filter(query).exclude(id__in=exclude_ids).distinct()[:limit]

        num_films_to_recommend = 10  # Numero totale di film da consigliare
        max_weight = 5  # Numero massimo di film da consigliare per il film più recente

        # Calcola la riduzione di film consigliati per ogni prenotazione più vecchia
        total_prenotazioni = len(prenotazioni)
        if total_prenotazioni > 1:
            step = (max_weight - 1) / (total_prenotazioni - 1)
        else:
            step = 0

```

```python

        for idx, prenotazione in enumerate(prenotazioni):
            film = prenotazione.proiezione.film

            # Calcola i limiti per genere e extra
            # Decrementa gradualmente il numero di film consigliati per prenotazioni meno recenti
            limit = max(max_weight - int(idx * step), 1)
            half_limit = limit // 2
            genre_limit = half_limit + 1 if limit % 2 != 0 else half_limit
            extra_limit = half_limit

            # Prima query per film dello stesso genere
            genre_query = Q(genere=film.genere)
            similar_genre_films = get_similar_films(genre_query, film_acquistati_ids, genre_limit)

            # Aggiungi i film dello stesso genere alla lista dei consigliati
            film_consigliati.extend(similar_genre_films) #Aggiungi in coda alla lista
            film_acquistati_ids.extend([f.id for f in similar_genre_films])

            # Seconda query per film con gli stessi extra (3D o inglese), esclusi quelli già inclusi
            extra_query = Q()
            if film.in_3D:
                extra_query |= Q(in_3D=True)
            if film.in_inglese:
                extra_query |= Q(in_inglese=True)

            if extra_query:
                similar_extra_films = get_similar_films(extra_query, film_acquistati_ids, extra_limit)
                film_consigliati.extend(similar_extra_films) #Aggiungi in coda alla lista
                film_acquistati_ids.extend([f.id for f in similar_extra_films])

        # Limita il numero totale di film consigliati
        return film_consigliati[:num_films_to_recommend]

```

Spiegazione dei passaggi

- Recupera le Prenotazioni dell'Utente:

```
15        prenotazioni = list(self.prenotazioni.all())
```

Qui recuperiamo tutte le prenotazioni dell'utente e le trasformiamo in una lista.

- Inverti l'Ordine delle Prenotazioni:

```
16        prenotazioni.reverse()  # Inverti l'ordine delle prenotazioni
```

Le prenotazioni vengono invertite, così che l'ultima prenotazione (la più recente) diventa la prima nella lista.

- Inizializza le Liste per Film Consigliati e IDs di Film Acquistati:

```
17        film_consigliati = []
18
19        film_acquistati_ids = [prenotazione.proiezione.film.id for prenotazione in prenotazioni]
```

film_consigliati è la lista che conterrà i film consigliati. *film_acquistati_ids* è una lista degli ID dei film che l'utente ha già acquistato, per evitare di raccomandare gli stessi film.

- Raccogliere Caratteristiche dei Film:

```
20
21        caratteristiche_films = defaultdict(list)
22        for prenotazione in prenotazioni:
23            film = prenotazione.proiezione.film
24            caratteristiche_films[film.genere].append(film)
25            if film.in_3D:
26                caratteristiche_films['3D'].append(film)
27            if film.in_inglese:
28                caratteristiche_films['inglese'].append(film)
29
```

Utilizzo un dizionario di liste (*caratteristiche_films*) per raccogliere i film per genere, e se sono in 3D o in inglese.

- Definizione della Funzione per Ottenere Film Simili:

```
30        def get_similar_films(query, exclude_ids, limit):
31            return Film.objects.filter(query).exclude(id__in=exclude_ids).distinct()[:limit]
```

Questa funzione esegue una query per ottenere film simili, escludendo quelli già acquistati dall'utente, fino a un certo limite.

- Impostare il Numero di Film da Raccomandare:

```
33        num_films_to_recommend = 10  # Numero totale di film da consigliare
34        max_weight = 5  # Numero massimo di film da consigliare per il film più recente
```

num_films_to_recommend è il numero totale di film che vogliamo consigliare. *max_weight* è il numero massimo di film che possiamo consigliare per la prenotazione più recente.

- Calcolare la Riduzione dei Film Consigliati per Prenotazioni Meno Recenti:

```
36          # Calcola la riduzione di film consigliati per ogni prenotazione più vecchia
37          total_prenotazioni = len(prenotazioni)
38          if total_prenotazioni > 1:
39              step = (max_weight - 1) / (total_prenotazioni - 1)
40          else:
41              step = 0
```

step determina di quanto diminuire il numero di film consigliati per ogni prenotazione meno recente.

- Generare i Consigli:

  Da questa sezione in poi, l'algoritmo lavora su ogni prenotazione dell'utente per generare i consigli basati sui film acquistati, partendo dalle prenotazioni più recenti fino a quelle più vecchie.

- Iterazione sulle Prenotazioni:

```
43          for idx, prenotazione in enumerate(prenotazioni):
44              film = prenotazione.proiezione.film
```

Per ogni prenotazione nell'elenco (che è stato invertito per avere le prenotazioni più recenti per prime), estraiamo il film associato alla proiezione.

- Calcolo del Limite dei Film da Consigliare:

```
48              limit = max(max_weight - int(idx * step), 1)
```

limit è il numero massimo di film da consigliare per quella prenotazione. Viene calcolato come *max_weight* meno un valore che dipende dall'indice della prenotazione (*idx* * *step*). Questo fa sì che il numero di film da consigliare diminuisca man mano che ci si sposta verso prenotazioni meno recenti. *max()* garantisce che *limit* sia almeno 1.

- Divisione del Limite tra Genere ed Extra:

```
49              half_limit = limit // 2
50              genre_limit = half_limit + 1 if limit % 2 != 0 else half_limit
51              extra_limit = half_limit
```

half_limit è la metà del *limit*. Se *limit* è dispari, *genre_limit* sarà metà di *limit* più uno, altrimenti sarà semplicemente *half_limit*. *extra_limit* è sempre *half_limit*. Questo assicura che ci siano più film consigliati per il genere rispetto agli extra quando *limit* è dispari.

- Query per Film dello Stesso Genere:

```
53          # Prima query per film dello stesso genere
54          genre_query = Q(genere=film.genere)
55          similar_genre_films = get_similar_films(genre_query, film_acquistati_ids, genre_limit)
```

Viene creata una query per trovare film dello stesso genere del film acquistato. *get_similar_films* esegue questa query escludendo i film già acquistati e limitando il numero di risultati a *genre_limit*. I film trovati vengono memorizzati in *similar_genre_films*.

- Aggiungere Film dello Stesso Genere ai Consigliati:

```
57          # Aggiungi i film dello stesso genere alla lista dei consigliati
58          film_consigliati.extend(similar_genre_films) #Aggiungi in coda alla lista
59          film_acquistati_ids.extend([f.id for f in similar_genre_films])
```

I film dello stesso genere trovati vengono aggiunti alla lista dei film consigliati (*film_consigliati*). Gli ID di questi film vengono anche aggiunti a *film_acquistati_ids* per evitare duplicati nelle query successive.

- Query per Film con gli Stessi Extra:

```
61          # Seconda query per film con gli stessi extra (3D o inglese), esclusi quelli già inclusi
62          extra_query = Q()
63          if film.in_3D:
64              extra_query |= Q(in_3D=True)
65          if film.in_inglese:
66              extra_query |= Q(in_inglese=True)
```

Viene creata una query per trovare film con gli stessi extra del film acquistato (in 3D o in inglese). Questa query utilizza l'operatore | (OR) per combinare le condizioni.

- Esecuzione della Query per gli Extra e Aggiunta ai Consigliati:

```
68          if extra_query:
69              similar_extra_films = get_similar_films(extra_query, film_acquistati_ids, extra_limit)
70              film_consigliati.extend(similar_extra_films) #Aggiungi in coda alla lista
71              film_acquistati_ids.extend([f.id for f in similar_extra_films])
```

Se *extra_query* contiene delle condizioni, viene eseguita una query per trovare film con gli stessi extra, escludendo quelli già acquistati e limitando il numero di risultati a *extra_limit*. I film trovati vengono aggiunti a *film_consigliati* e i loro ID a *film_acquistati_ids*.

- Limite Finale sui Film Consigliati:

```
73          # Limita il numero totale di film consigliati
74          return film_consigliati[:num_films_to_recommend]
```

Alla fine, la lista *film_consigliati* viene troncata per contenere al massimo *num_films_to_recommend* film. Questo garantisce che l'utente non riceva più film di quanti specificati.

In conclusione, l'algoritmo genera raccomandazioni basate sulle caratteristiche dei film recentemente acquistati dall'utente. I film vengono selezionati in base al genere e agli extra (3D o in inglese), dando la priorità ai film del genere dell'ultimo acquisto e successivamente agli extra. Man mano che ci si sposta verso acquisti più vecchi, il numero di film raccomandati per ogni acquisto diminuisce, ma i film simili al genere vengono raccomandati in quantità maggiore rispetto agli extra.

6. Test fatti

Test su funzione di codice applicativo

Ho testato la funzione *clean_data()* presente nel file *forms.py*. Questa funzione appartiene alla classe *CreateProiezioneForm(forms.ModelForm)*. Essa viene invocata nel momento in cui un utente gestore inserisce una nuova Proiezione: il compito della funzione è quello di controllare che la data inserita dall'utente sia valida. Se la data inserita dall'utente è una data passata, allora deve essere sollevato un *ValidationError* con il messaggio *"La data della proiezione deve essere futura."* Se la data inserita è la data odierna, oppure una data futura, allora la funzione ritorna la data inserita.

Test eseguiti

I test eseguiti per questa funzione sono nel file *tests.py* nella classe *CreateProiezioneFormTest(TestCase)*.

- Funzione *setUp()*
  Funzione di configurazione. La funzione *setUp* viene eseguita prima di ogni test. Qui, creo un dizionario *self.data_obbligatori* che contiene i dati obbligatori per creare una proiezione (senza la data). Questo dizionario verrà copiato e completato con diverse date nei vari test.
- Test: *test_clean_data_past_date()*
  Lo scopo di questa funzione è di verificare che la funzione *clean_data* sollevi un *ValidationError* quando la data è nel passato. Essa esegue i seguenti passaggi: copia i dati obbligatori, imposta la data a ieri, crea una form con questi dati, imposta *cleaned_data* sulla form per simulare la validazione, controlla che venga sollevato un *ValidationError* con il messaggio corretto.
- Test: test_clean_data_today_date()
  Lo scopo di questa funzione è di verificare che la funzione *clean_data* non sollevi un *ValidationError* e ritorni la data corretta quando la data è oggi. Essa esegue i seguenti passaggi: copia i dati obbligatori, imposta la data a oggi, crea una form con questi dati, imposta *cleaned_data* sulla form per simulare la validazione,

tenta di pulire i dati e controlla che la funzione ritorni la data odierna, se viene sollevato un *ValidationError*, il test fallisce.

- Test: test_clean_data_future_date()
  Lo scopo di questa funzione è di verificare che la funzione *clean_data* non sollevi un *ValidationError* e ritorni la data corretta quando la data è nel futuro. Essa esegue i seguenti passaggi: copia i dati obbligatori, imposta la data a domani, crea una form con questi dati, imposta *cleaned_data* sulla form per simulare la validazione, tenta di pulire i dati e controlla che la funzione ritorni la data di domani, se viene sollevato un *ValidationError*, il test fallisce.

Ho testato anche la funzione *clean()* presente nel file *forms.py*. Questa funzione appartiene alla classe *CreateProiezioneForm(forms.ModelForm)*. Essa viene invocata nel momento in cui un utente gestore inserisce una nuova Proiezione: il compito della funzione è quello di controllare che non venga inserita una nuova proiezione con la stessa data, ora di inizio e sala di un'altra proiezione già presente nel database. Se vengono trovate delle proiezioni che si sovrappongono a quella che l'utente sta provando ad inserire allora viene sollevato un *ValidationError* con il messaggio "*Esiste già una proiezione nella sala selezionata alla data e ora specificate*." Se invece non esistono proiezioni nel database che si sovrappongono a quella nuova che si sta cercando di inserire, la funzione ritorna i dati inseriti per la nuova proiezione contenuti nella variabile *cleaned_data*.

Test eseguiti

I test eseguiti per questa funzione sono nel file *tests.py* nella classe *CreateProiezioneFormTest(TestCase)*.

- Funzione *setUp()*
  Funzione di configurazione. La funzione *setUp* viene eseguita prima di ogni test. Qui, creo un dizionario *self.data_obbligatori* che contiene i dati obbligatori per creare una proiezione (senza la data). Questo dizionario verrà copiato e completato con diverse date nei vari test.
- Test: *test_clean_overlapping_proiezione()*
  Lo scopo di questo test è di verificare che il metodo *clean()* del form *CreateProiezioneForm* sollevi un'eccezione *ValidationError* se viene tentato di creare una proiezione che si sovrappone a una già esistente nella stessa sala, alla stessa data e ora. Esso esegue i seguenti passaggi: crea un film di esempio, crea una proiezione con la stessa data, ora e sala che verranno utilizzate nel form, copia i dati di esempio e aggiunge la data, crea un form *CreateProiezioneForm* con questi dati e imposta *cleaned_data* sulla form per simulare la validazione, verifica che il metodo *clean()* sollevi un'eccezione *ValidationError* con il messaggio atteso.
- Test: *test_clean_non_overlapping_proiezione()*
  Lo scopo di questo test è di verificare che il metodo *clean()* del form *CreateProiezioneForm* non sollevi un'eccezione *ValidationError* se viene tentato di creare una proiezione che non si sovrappone a nessuna esistente. Esso esegue i seguenti passaggi: crea un film di esempio, crea una proiezione con una data diversa da quella che verrà utilizzata nel form, copia i dati di esempio e aggiunge

la data futura, crea un form *CreateProiezioneForm* con questi dati e imposta *cleaned_data* sulla form per simulare la validazione, verifica che il metodo *clean()* non sollevi un'eccezione *ValidationError* e restituisca i dati puliti attesi.

Test su “vista” (pagina) utente

Ho testato la vista *FilmProjectionsView(ListView)* presente nel file *views.py*. Essa ha il compito di filtrare le proiezioni di un film specifico dato, mantenendo solo le proiezioni che hanno data odierna o futura.

Test eseguiti

I test eseguiti per questa vista sono nel file *tests.py* nella classe *FilmProjectionsViewTest(TestCase)*.

- Funzione *setUp()*
  Funzione di configurazione. La funzione *setUp* viene eseguita prima di ogni test per creare l'ambiente di test. Qui, vengono creati un film di esempio e tre proiezioni con date diverse (passata, presente, futura): *self.film* è un film di esempio*, self.today* contiene la data di oggi, *self.ora_inizio* è l'ora di inizio delle proiezioni (14:00), *self.proiezione_passata* è una proiezione con data di ieri, *self.proiezione_futura* è una proiezione con data di domani, *self.proiezione_presente* è una proiezione con data di oggi.
- Test: *test_no_proiezioni()*
  Lo scopo di questa funzione è di verificare che venga mostrato il messaggio "Nessuna proiezione trovata." quando non ci sono proiezioni. Essa esegue i seguenti passaggi: rimuove tutte le proiezioni, effettua una richiesta alla vista delle proiezioni del film, verifica che lo status code sia 200, controlla che il messaggio "Nessuna proiezione trovata." sia presente nella risposta, verifica che la lista di oggetti (*object_list*) sia vuota.
- Test: *test_only_past_proiezioni()*
  Lo scopo di questa funzione è di verificare che venga mostrato il messaggio "Nessuna proiezione trovata." quando ci sono solo proiezioni passate. Essa esegue i seguenti passaggi: rimuove tutte le proiezioni future e presenti, effettua una richiesta alla vista delle proiezioni del film, verifica che lo status code sia 200, controlla che il messaggio "Nessuna proiezione trovata." sia presente nella risposta, verifica che la lista di oggetti (*object_list*) sia vuota.
- Test: *test_only_future_proiezioni()*
  Lo scopo di questa funzione è di verificare che vengano mostrate solo le proiezioni future. Essa esegue i seguenti passaggi: rimuove tutte le proiezioni passate e presenti, effettua una richiesta alla vista delle proiezioni del film, verifica che lo status code sia 200, controlla che il titolo del film della proiezione futura sia presente nella risposta, verifica che il messaggio "Nessuna proiezione trovata." non sia presente nella risposta, controlla che la proiezione futura sia nella lista di oggetti (*object_list*).
- Test: *test_only_present_proiezioni()*
  Lo scopo di questa funzione è di verificare che vengano mostrate solo le proiezioni presenti. Essa esegue i seguenti passaggi: rimuove tutte le proiezioni

passate e future, effettua una richiesta alla vista delle proiezioni del film, verifica che lo status code sia 200, controlla che il titolo del film della proiezione presente sia presente nella risposta, verifica che il messaggio "Nessuna proiezione trovata." non sia presente nella risposta, controlla che la proiezione presente sia nella lista di oggetti (*object_list*).

- Test: *test_mixed_proiezioni()*
  Lo scopo di questa funzione è di verificare che vengano mostrate solo le proiezioni future e presenti, escludendo quelle passate. Essa esegue i seguenti passaggi: effettua una richiesta alla vista delle proiezioni del film, verifica che lo status code sia 200, controlla che il titolo del film delle proiezioni future e presenti sia presente nella risposta, verifica che la proiezione presente sia nella lista di oggetti (*object_list*), verifica che la proiezione futura sia nella lista di oggetti (*object_list*), controlla che la proiezione passata non sia nella lista di oggetti (*object_list*).

Ho testato anche la vista *prenota_proiezione(request, proiezione_id)* presente nel file *views.py*. Essa ha il compito di consentire all'utente registrato di effettuare una prenotazione per una proiezione e di negare eventualmente la prenotazione nel caso in cui alcune condizioni necessarie per effettuarla non siano rispettate.
Test eseguiti
I test eseguiti per questa vista sono nel file *tests.py* nella classe *PrenotaProiezioneViewTest(TestCase)*.

- Funzione *setUp()*
  Funzione di configurazione. La funzione *setUp* viene eseguita prima di ogni test per creare l'ambiente di test. Questo metodo crea un utente di test, un film di esempio e una proiezione di esempio che verranno utilizzati nei test successivi.
- Test: *test_accesso_non_autorizzato()*
  Lo scopo di questa funzione è di verificare che un utente non autenticato venga reindirizzato alla pagina di login se tenta di accedere alla view di prenotazione di una proiezione. Essa esegue i seguenti passaggi: esegue una richiesta GET alla view di prenotazione di una proiezione usando *self.client.get*, Verifica che la risposta sia un reindirizzamento (*self.assertRedirects*) alla pagina di login con il parametro next corretto.
- Test: *test_proiezione_inesistente()*
  Lo scopo di questa funzione è di verificare che venga restituito un codice di stato 404 se si tenta di accedere alla prenotazione di una proiezione con un ID che non esiste nel database. Essa esegue i seguenti passaggi: esegue il login con l'utente di test usando *self.client.login*, esegue una richiesta GET alla view di prenotazione di una proiezione con un ID inesistente usando *self.client.get*, verifica che il codice di stato della risposta sia 404 (*self.assertEqual*).
- Test: *test_nessun_posto_disponibile()*
  Lo scopo di questa funzione è di verificare che venga mostrato un messaggio di errore e che l'utente venga reindirizzato alla homepage se tenta di prenotare una proiezione con zero posti disponibili. Essa esegue i seguenti passaggi: imposta il numero di posti disponibili della proiezione a 0 *(self.proiezione.posti_disponibili*

= *0*), salva la proiezione aggiornata usando *self.proiezione.save()*, esegue il login con l'utente di test usando *self.client.login*, esegue una richiesta GET alla view di prenotazione di una proiezione usando *self.client.get*, Verifica che venga mostrato un messaggio di errore (*messages = list(get_messages(response.wsgi_request))*) e controlla che il messaggio sia quello atteso con *assertEqual*, controlla che l'utente venga reindirizzato alla homepage (*self.assertRedirects*).

- Test: *test_prenotazione_riuscita()*
  Lo scopo di questa funzione è di simulare una prenotazione riuscita verificando che l'utente venga reindirizzato alla homepage, che i posti disponibili siano diminuiti e che sia stata creata correttamente una prenotazione nel database. Essa esegue i seguenti passaggi: esegue il login con l'utente di test usando *self.client.login*, esegue una richiesta GET alla view di prenotazione di una proiezione usando *self.client.get*, verifica che l'utente venga reindirizzato correttamente alla homepage dopo aver completato la prenotazione (*self.assertRedirects*), aggiorna la proiezione dal database usando *self.proiezione.refresh_from_db()* per ottenere i dati aggiornati, verifica che il numero di posti disponibili sia diminuito (*self.assertEqual*), verifica che sia stata creata correttamente una prenotazione nel database (*Prenotazione.objects.filter*), verifica che venga mostrato un messaggio di successo (*messages = list(get_messages(response.wsgi_request))*), controlla che il contenuto del messaggio corrisponda al messaggio di successo atteso.
- Test: *test_prenotazione_con_next_lista_proiezioni()*
  Lo scopo di questa funzione è di verificare che l'utente venga reindirizzato correttamente alla lista delle proiezioni di un film specificato quando il parametro next contiene la stringa corretta. Essa esegue i seguenti passaggi: esegue il login con l'utente di test usando *self.client.login*, costruisce l'URL di reindirizzamento per la lista delle proiezioni di un film usando reverse, esegue una richiesta GET alla view di prenotazione di una proiezione usando *self.client.get*, includendo next nell'URL, verifica che l'utente venga reindirizzato correttamente alla lista delle proiezioni di un film (*self.assertRedirects*), verifica che sia stata creata correttamente una prenotazione nel database (*Prenotazione.objects.filter*).
- Test: *test_prenotazione_con_next_proiezioni_film()*
  Lo scopo di questa funzione è di verificare che l'utente venga reindirizzato correttamente alla lista delle proiezioni di un film quando il parametro next contiene la stringa corretta. Essa esegue i seguenti passaggi: esegue il login con l'utente di test usando *self.client.login*, costruisce l'URL di reindirizzamento per le proiezioni di un film usando reverse, esegue una richiesta GET alla view di prenotazione di una proiezione usando *self.client.get*, includendo next nell'URL, verifica che l'utente venga reindirizzato correttamente alle proiezioni di un film (*self.assertRedirects*), verifica che sia stata creata correttamente una prenotazione nel database (*Prenotazione.objects.filter*).
- Test: *test_prenotazione_multiple_per_utente()*

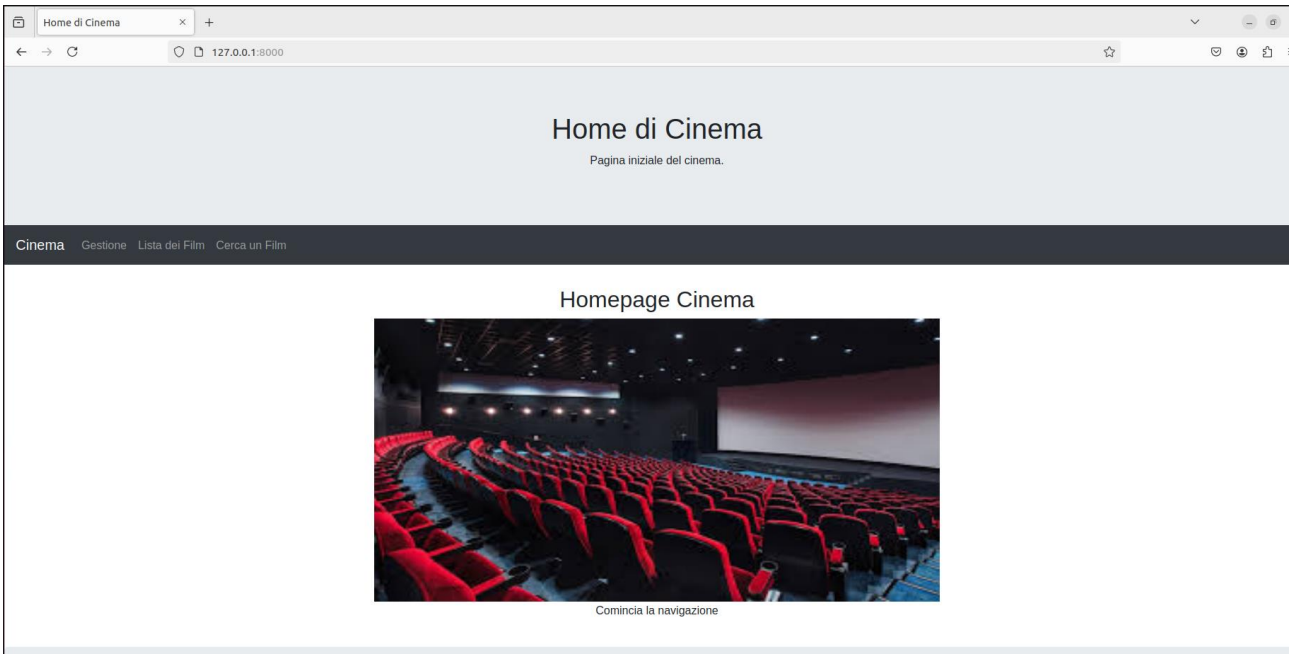
Lo scopo di questa funzione è di simulare la prenotazione della stessa proiezione due volte e verifica che entrambe le prenotazioni siano registrate correttamente nel database. Essa esegue i seguenti passaggi: esegue il login con l'utente di test usando *self.client.login*, esegue una prima richiesta GET alla view di prenotazione di una proiezione usando *self.client.get*, verifica che l'utente venga reindirizzato correttamente alla homepage dopo aver completato la prima prenotazione (*self.assertRedirects*), esegue una seconda richiesta GET alla view di prenotazione di una proiezione usando *self.client.get*, verifica che l'utente venga reindirizzato correttamente alla homepage dopo aver completato la seconda prenotazione (*self.assertRedirects*), verifica che siano state create correttamente due prenotazioni nel database (*Prenotazione.objects.filter*).

- Test: *test_prenotazione_in_sala_differente_stessa_ora()*
  Lo scopo di questa funzione è di verificare che un utente non possa prenotare una proiezione se ha già una prenotazione per la stessa ora ma in una sala diversa, mostrando un messaggio di errore appropriato e non creando una nuova prenotazione. Essa esegue i seguenti passaggi: esegue il login con l'utente di test usando *self.client.login*, crea una seconda proiezione per lo stesso film e orario ma in una sala diversa usando *Proiezione.objects.create*, crea una prenotazione per la seconda proiezione usando *Prenotazione.objects.create*, esegue una richiesta GET alla view di prenotazione di una proiezione usando *self.client.get* per la prima proiezione, verifica che venga mostrato un messaggio di errore appropriato (*messages = list(get_messages(response.wsgi_request))*), controlla con *assertEqual* che il messaggio ottenuto sia quello atteso, verifica che l'utente venga reindirizzato alla homepage (*self.assertRedirects*), controlla che l'utente abbia solo una prenotazione con *assertEqual*, verifica che non sia stata creata la prenotazione per la prima proiezione nel database (*Prenotazione.objects.filter*).
- Test: *test_prenotazione_con_film_proiezioni_per_data_con_filter_date()*
  Lo scopo di questa funzione è di verificare che l'utente venga reindirizzato correttamente alla pagina delle proiezioni di un film in una data specifica quando il parametro next contiene la stringa corretta e *filter_date* è specificato. Essa esegue i seguenti passaggi: esegue il login con l'utente di test usando *self.client.login*, costruisce l'URL di reindirizzamento per le proiezioni di un film in una data specifica usando reverse, esegue una richiesta GET alla view di prenotazione di una proiezione usando *self.client.get*, includendo next e *filter_date* nell'URL, verifica che l'utente venga reindirizzato correttamente alla pagina delle proiezioni di un film in una data specifica (*self.assertRedirects*), verifica che sia stata creata correttamente una prenotazione nel database (*Prenotazione.objects.filter*).
- Test: *test_prenotazione_senza_next_url()*
  Lo scopo di questa funzione è di verificare che l'utente venga reindirizzato correttamente alla homepage dopo aver completato con successo una prenotazione senza specificare un URL di reindirizzamento successivo. Essa esegue i seguenti passaggi: esegue il login con l'utente di test usando *self.client.login*, esegue una richiesta GET alla view di prenotazione di una proiezione usando *self.client.get*, verifica che l'utente venga reindirizzato

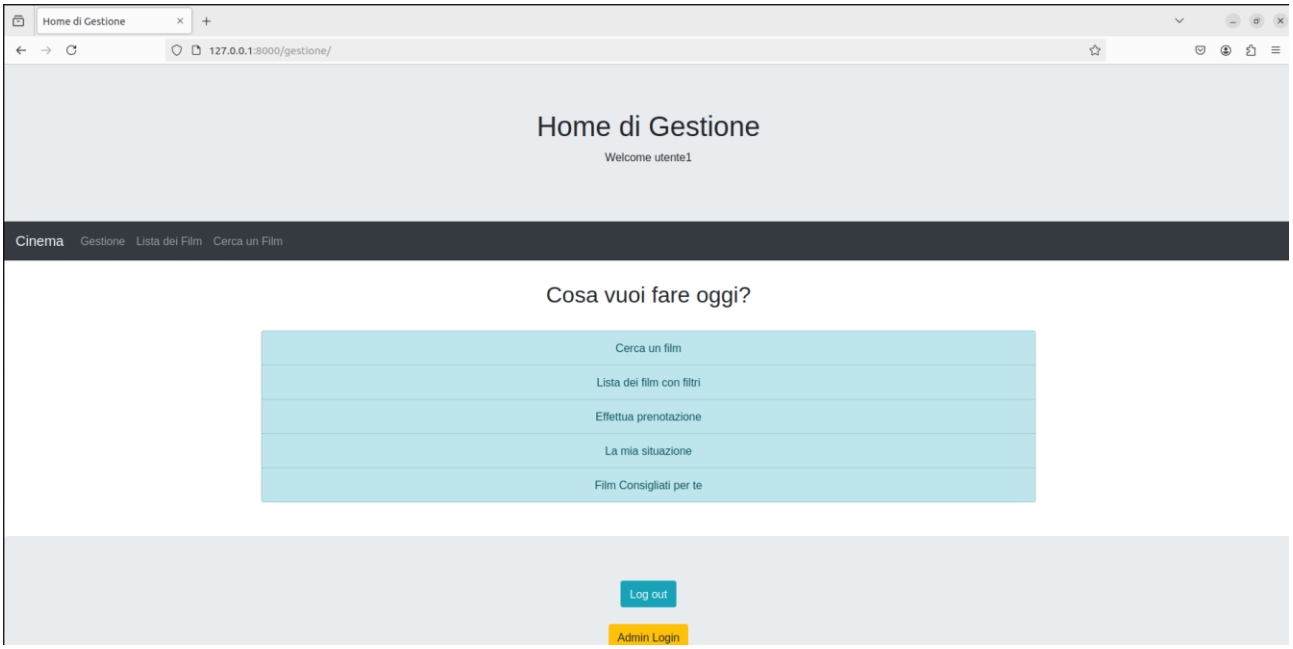
correttamente alla homepage dopo aver completato con successo la prenotazione (*self.assertRedirects*), verifica che sia stata creata correttamente una prenotazione nel database (*Prenotazione.objects.filter*).

7. Risultati

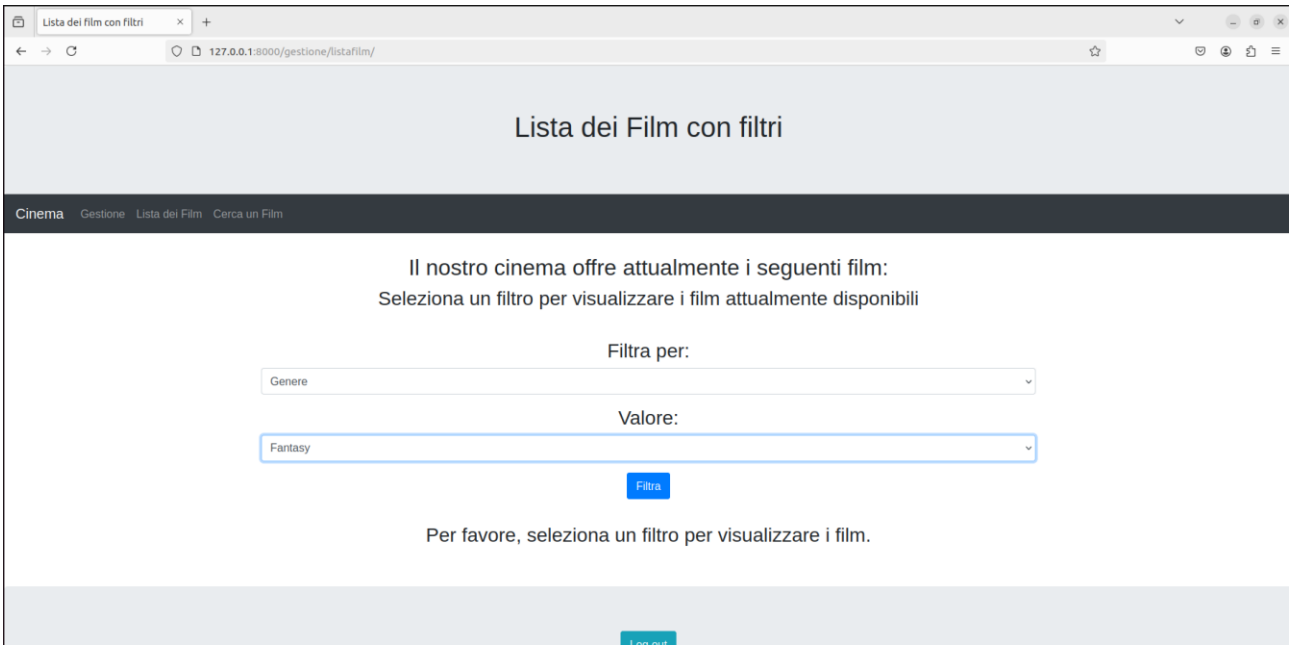
Homepage del sito



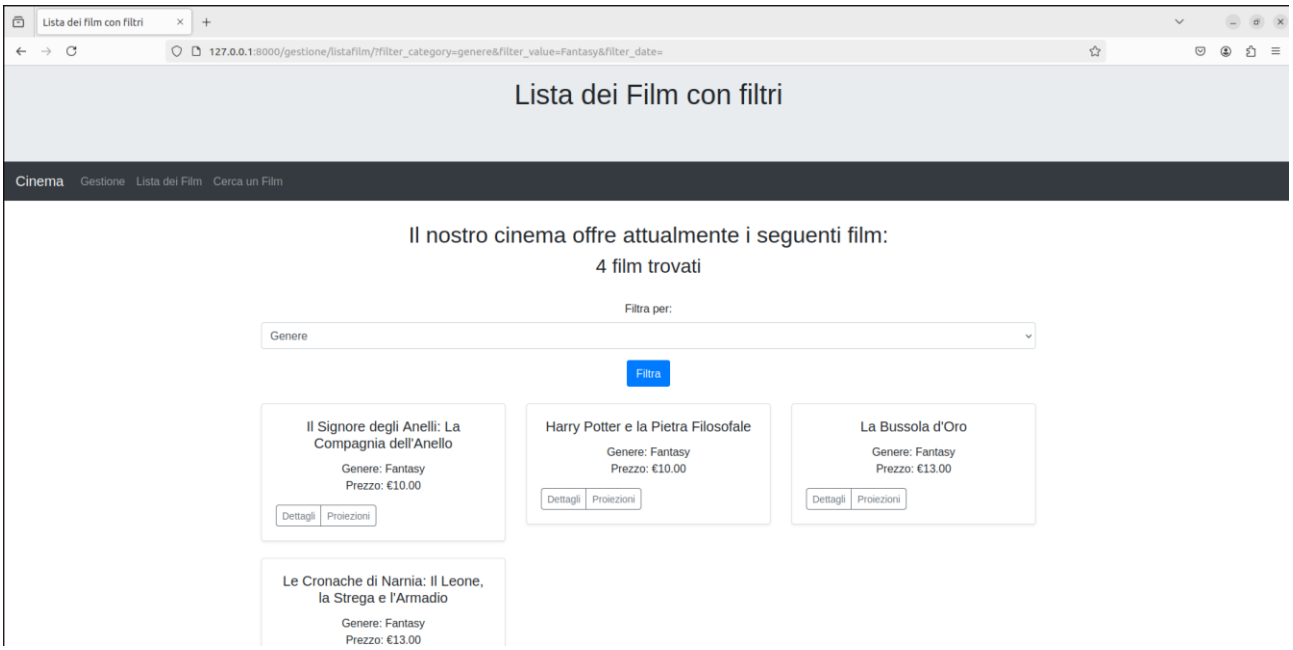
Homepage dell'app Gestione per un utente cliente



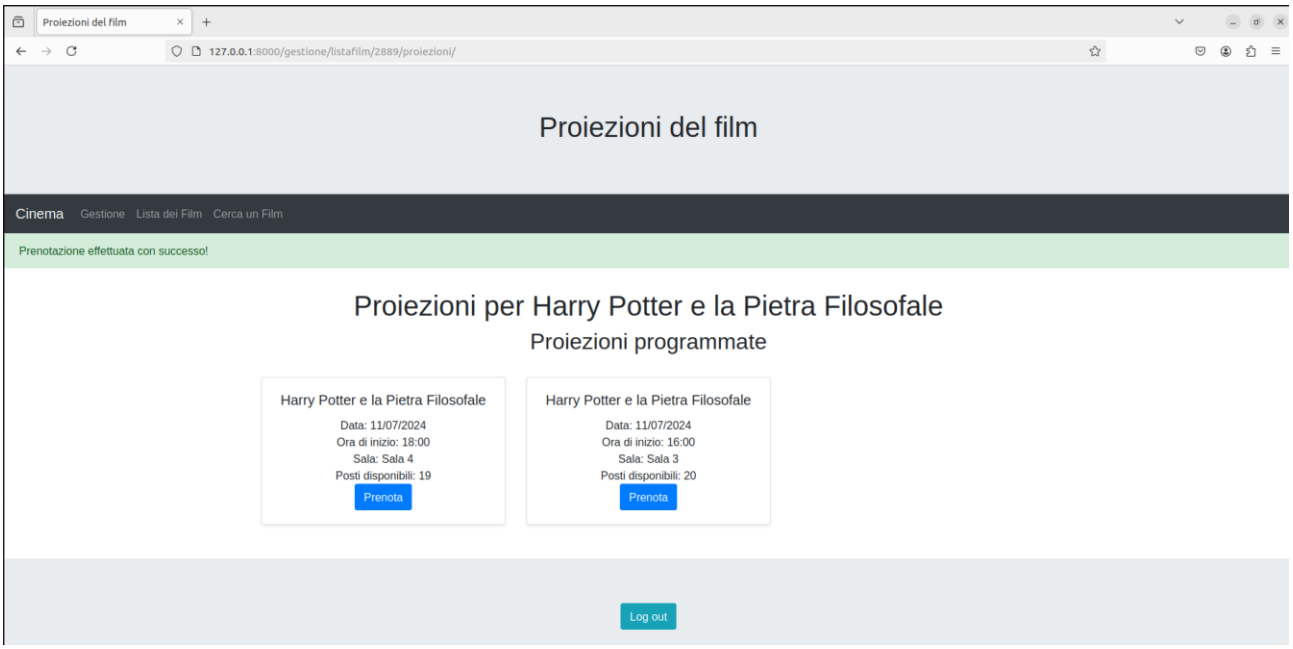
Pagina per filtrare i film prima di aver applicato il filtro scelto



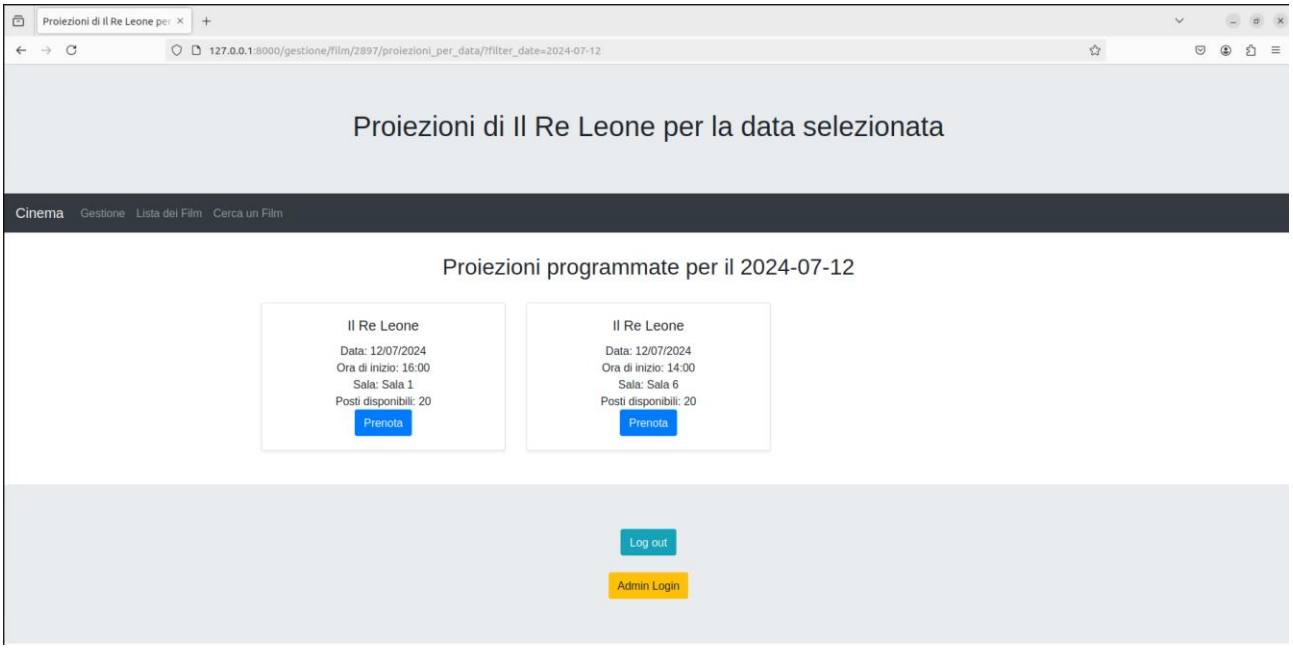
Pagina per filtrare i film dopo aver applicato un filtro



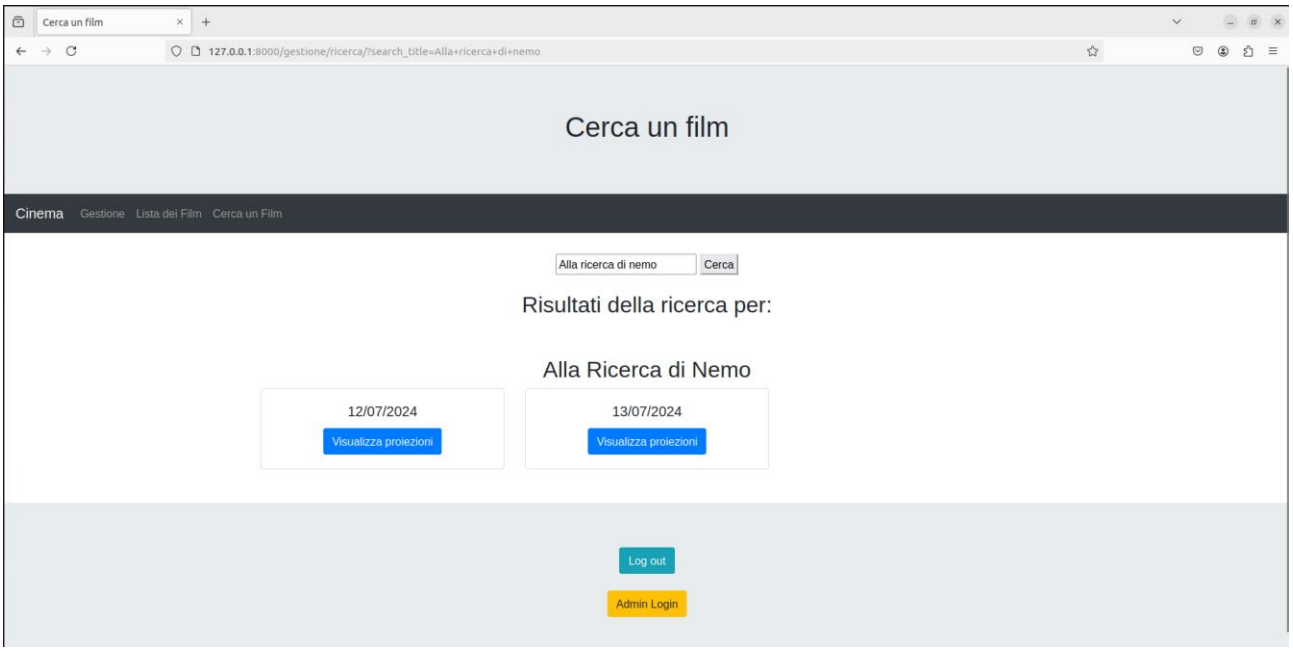
Proiezioni di un film con prenotazione effettuata con successo



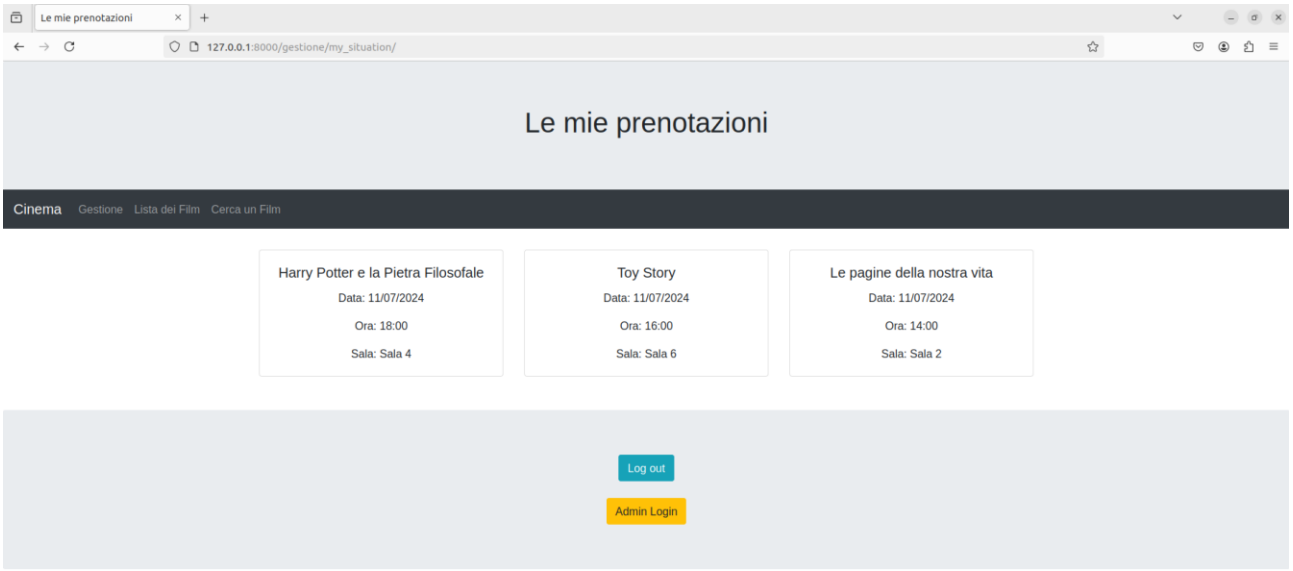
Proiezioni per data di un film selezionato dopo aver filtrato per data



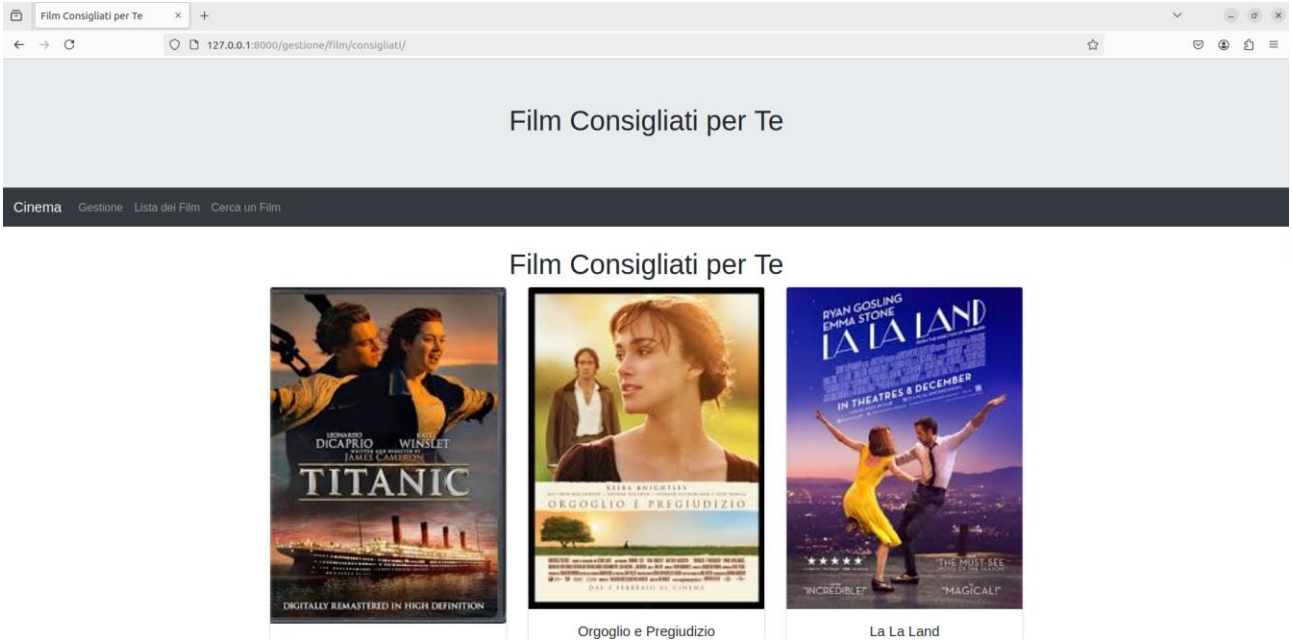
Pagina risultante dopo aver effettuato una ricerca per titolo di un film



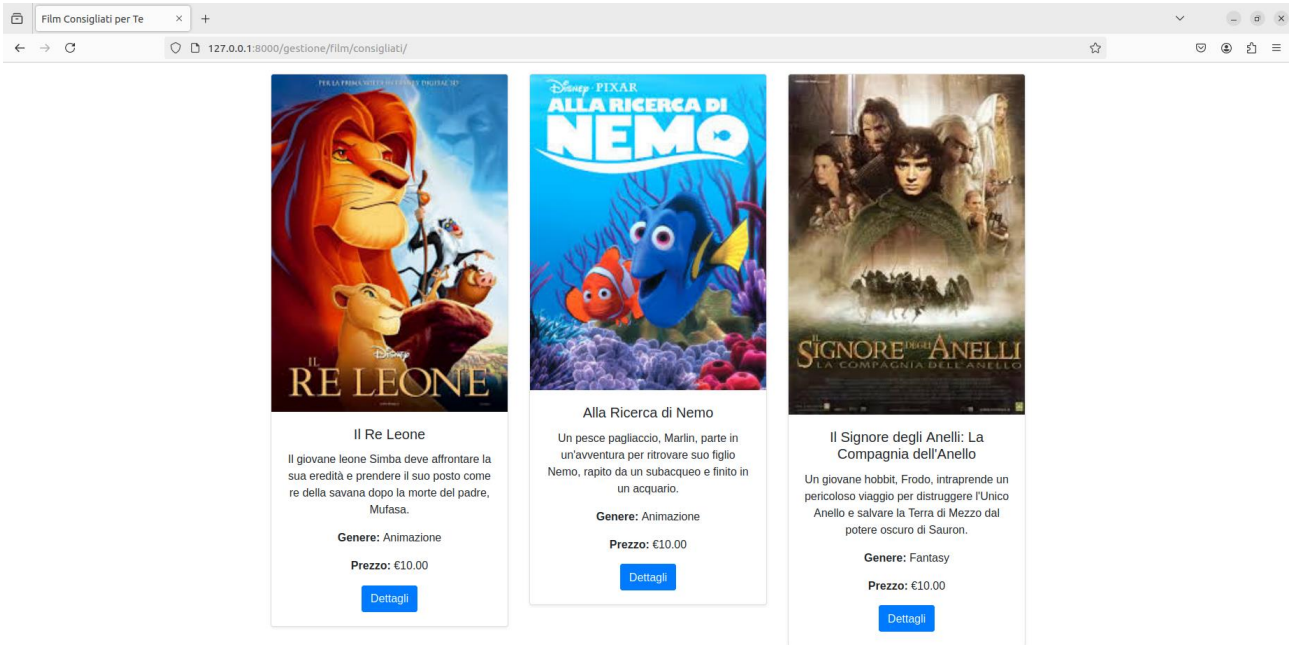
Pagina che mostra le prenotazioni effettuate dall'utente



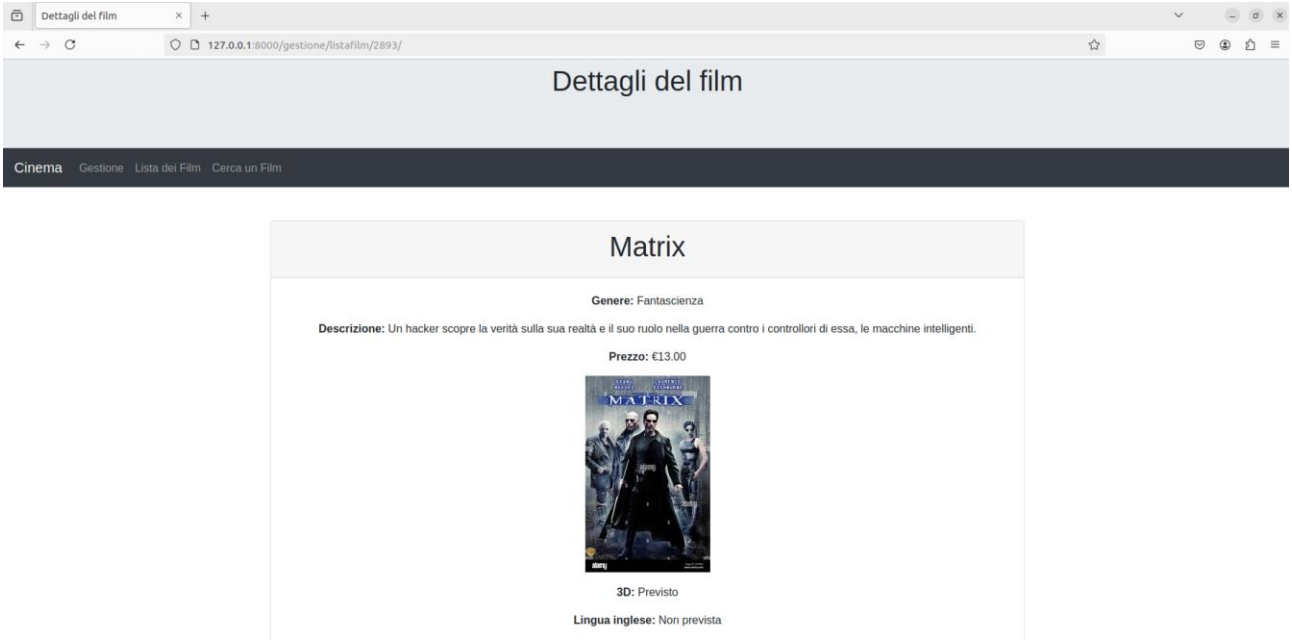
Pagina che mostra i film consigliati per l'utente in base alle sue prenotazioni



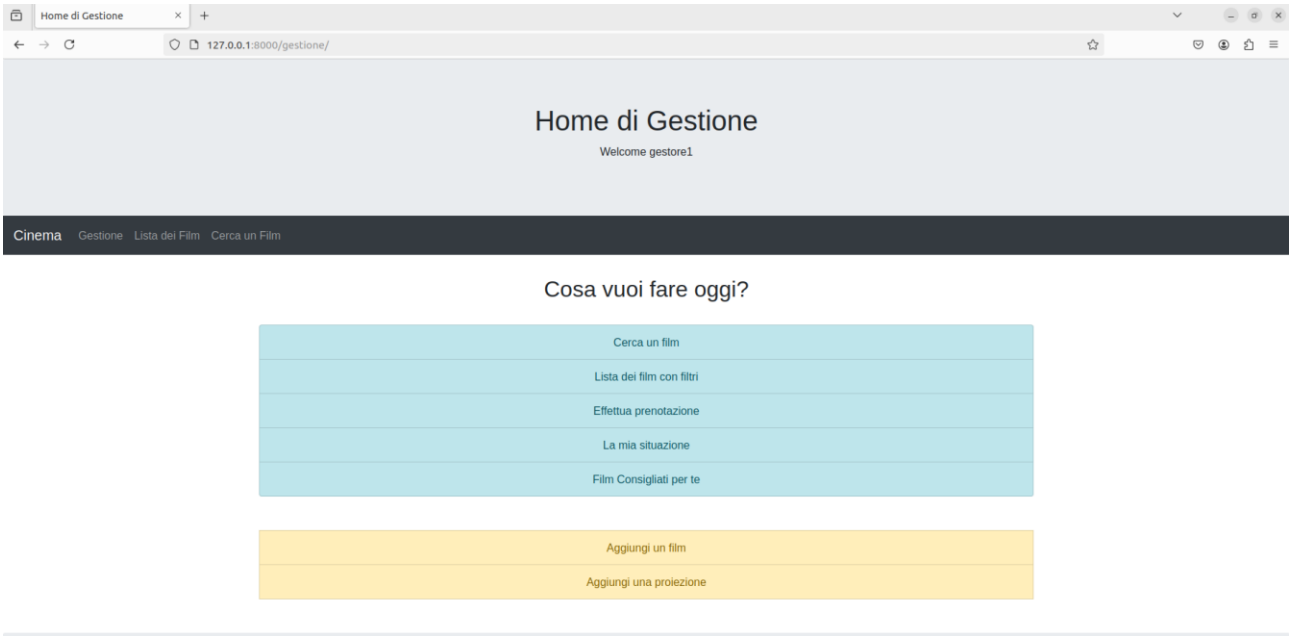
Seconda parte della pagina dei film consigliati



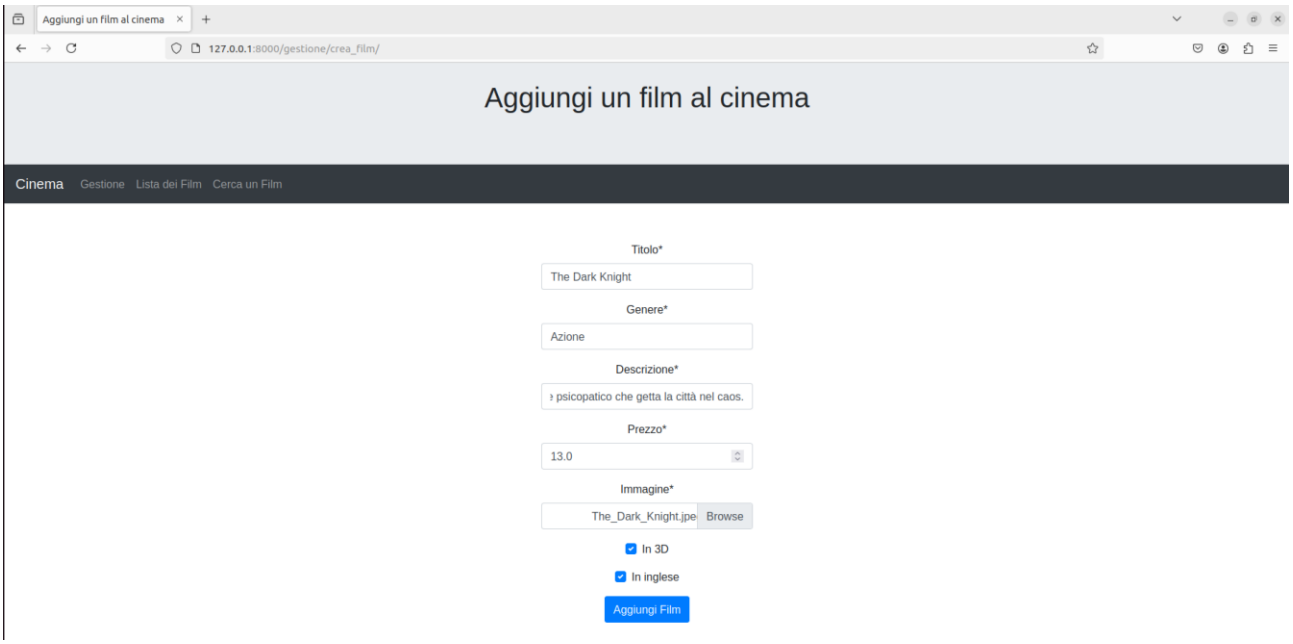
Pagina che mostra i dettagli di un film



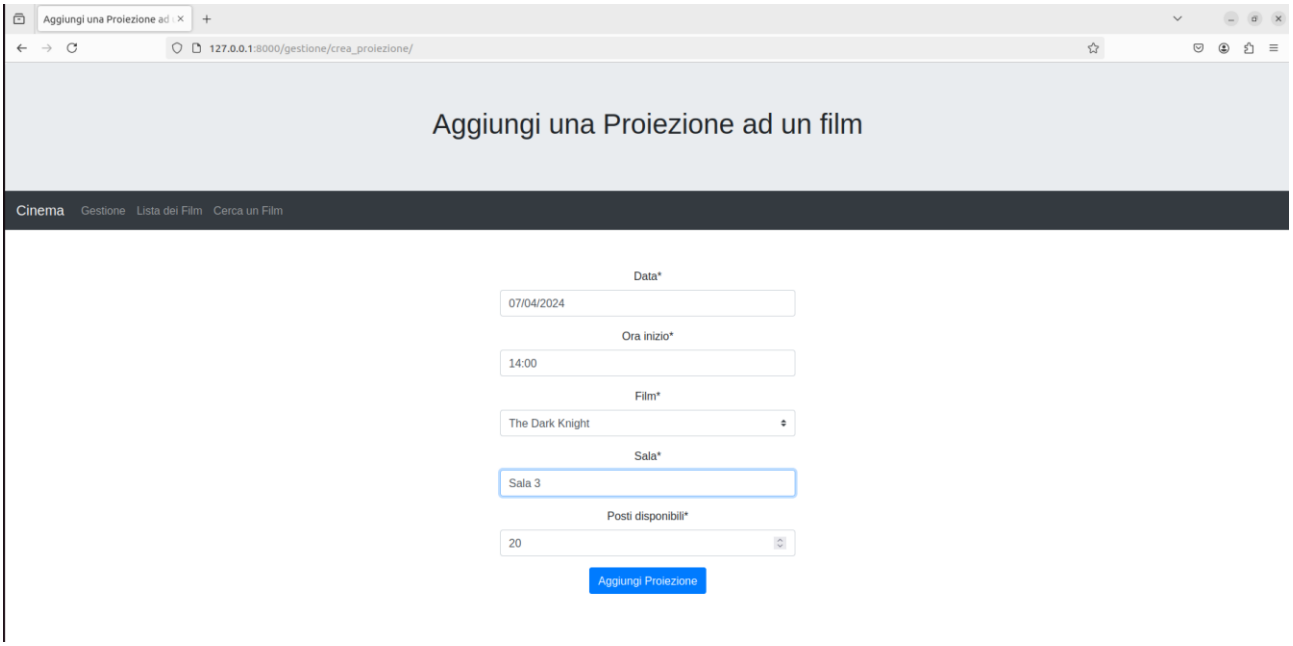
Home dell'app Gestione per un utente gestore



Pagina per aggiungere un nuovo film al Cinema



Pagina per aggiungere una nuova proiezione di un film



Home dell’app Gestione per l’utente admin