Progetto Tecnologie Web

Preti Christian

2022

La relazione ha lo scopo di presentare il progetto svolto per il corso di *Tecnologie Web*, ovvero un sito web per lo streaming di film e serie tv.

Saranno di seguito presentate le funzionalità principali del sito:

- Possibilità di recensire un film, con $1 \sim 5$ stelle e commento scritto.
- Panoramica delle recensioni scritte da un determinato utente.
- Lista di tutte le recensioni di un determinato film, con valutazione media.
- Ranking dei film in base alla valutazione media.
- Possibilità di cercare film in base al genere o con un voto minimo/massimo.
- Inserire un film tra i preferiti.
- Recommendation system operante sui generi preferiti di un utente, basandosi sulle recensioni di altri utenti a film guardati da entrambi.
- Chat di gruppo relativa ad un determinato film.

Nella prossima pagina verranno dati qualche approfondimenti su alcune di queste.

Approfondimenti sui template

Home: Essa contiene i form di login/registrazione. E' possibile anche accedere al sito come utenti anonimi, attraverso il tasto '*Entra come guest*'.

Catalogo: E' il catalogo del sito, al suo interno è possibile trovare la lista di tutti i film, con annessa **valutazione, generi, stato** (guardato/non guardato) e un **pulsante** che simula la possibilità di guardare il film.
Cliccando sul titolo del film si verrà reindirizzati verso una pagina descrittiva più accurata, in cui sarà possibile avere informazioni aggiuntive quali, ad esempio, la trama. Inoltre, sempre nella pagina descrittiva del film, sarà possibile trovare un pulsante chiamato *Chatta*, che appunto fornirà l'accesso, previa visione del film e login nel sito, alla chat di gruppo specifica del film.

Account: Nella pagina *Il mio account* sono presenti i vari *componenti*:

- **Le mie recensioni**: Esso rappresenta un collegamento verso una pagina, in cui sarà possibile avere una panoramica delle recensioni lasciate dall'utente verso tutti i film.
- **Grafico**: E' un grafico di *Chart.js*, è calcolato attraverso il metodo *calcolaGeneri()*, presente nel file *methods.py*.
- **Pulsanti blu**: Rappresentano un film *guardato* dall'utente.
- **Pulsanti viola**: Rappresentano un film *consigliato* dal *recommendation system* all'utente.

Ranking: I parametri su cui è possibile fare ranking sono:

- **Titolo**.
- **Valutazione**, sia essa minima che massima.
- **Genere**.

Chat: Rappresentata come una semplice pagina in cui gli utenti, dopo aver necessariamente guardato il film, avranno modo di commentarlo con altri utenti.

Tecnologie utilizzate

Frontend: Le tecnologie utilizzate lato frontend sono state:

- **Bootstrap**: E' probabilmente il framework più utilizzato in ambito frontend, in grado di ricoprire i vari HTML, CSS e JavaScript. Ho fatto largo uso di Bootstrap, al fine di disegnare in modo facile e veloce le varie pagine del sito.

- **Chart.js**: E' una libreria che permette di importare grafici all'interno del proprio sito. È molto ben fatta, ben documentata e facile da usare. In questo contesto, è stata usata per disegnare un *grafico poligonale* all'interno della pagina di account dell'utente.

Lato **Backend**, invece:

- **SQLite**, data la comodità nel gestire il database in modo single-file e per la sua diretta integrazione con Django.

- **VSCode**, data la comodità nell'avere plugin che renderizzassero live le pagine HTML durante la scrittura, cosa che ha velocizzato molto il workflow. Inoltre, i plugin per la gestione dei test, facilmente avviabili in qualsiasi momento. Oltretutto, la possibilità di passare facilmente da un linguaggio di markup quale HTML o CSS, a Python/JavaScript.

Divisione nelle varie app

Il progetto è stato diviso in 4 app:

- **Streamify**: È l'app principale del progetto, essa contiene le varie pagine, come ad esempio l'homepage, il catalogo dei film, la pagina *'Il mio account'*.
- **Chatify**: Come suggerisce il nome, essa fornisce il servizio di chat all'interno del sito. Ho deciso di separarla da Streamify, al fine di vederlo come un *'servizio ausiliario'*, un po' come se fosse una feature esterna rispetto allo scopo *principale* del sito.
- **Auth**: Anche l'autenticazione è stata separata dal sito principale, dato che ho cercato di tenere le varie logiche quanto più separate possibili, al fine di aumentare la modularità del sito e la sua portabilità.
- **TestApp**: Essa contiene i test di tutte le app citate sopra, creando un'app dedicata a questo scopo, mi è stato più facile gestirli, potendoli raggruppare tutti sotto la stessa directory. Ulteriori dettagli su questo verranno affrontati nel paragrafo dedicato.

Ulteriori dettagli sulle funzionalità/scelte adottate

Recommendation System: Esso è stato pensato come la somiglianza tra la doppia {genere: voto medio} di ogni utente verso ogni genere presente nel database. Inoltre, verranno dati suggerimenti solamente sui due generi preferiti da utente, dato che difficilmente egli vorrà ricevere consigli su generi che non gli interessano.
Questa lista di due generi *preferiti* verrà calcolata per ogni utente e verranno cercate delle corrispondenze tra gli elementi. Se trovati, si guarderà la differenza di voto medio al genere. Più la differenza sarà bassa, più si può presumere che i gusti, quantomeno su quel genere, tra i due utenti siano simili, e verranno consigliati all'utente loggato quelli guardati dall'altro utente, ma non da lui.
Poniamo ad esempio:

```
U1: {
'Azione': 4.5,
'Avventura': 4.8,
}

U2: {
'Azione': 4.4,
'Avventura': 5.0,
}
```

Ipotizzando che i due generi preferiti dei due utenti siano quelle sopra citate, verrà trovato un match per entrambi i generi, dato che entrambi i generi di U1 sono i preferiti anche di U2, a questo punto bisogna calcolare la similarità, ovvero **quanto** il suggerimento può essere inerente, la formula sarà:

$$100 - \frac{100 * |VotoGenereU1 - VotoGenereU2|}{5}$$

In questo esempio, all'utente U1 verranno suggeriti con una precisione del 98% i film di genere *Azione* guardati da U2 (ma ovviamente non da U1) e allo stesso modo con la precisione del 96% i film di genere *Avventura*.

Gestione delle sessioni: L'utente loggato è stato gestito attraverso l'uso di **session**, ovvero un dizionario Python facente parte del parametro *request*, in cui è possibile salvare valori relativi alla sessione corrente. Le sessioni possono poi essere personalizzate in base alle proprie preferenze attraverso vari parametri, personalmente ho fatto uso dei seguenti, impostati nel file *settings.py*:

```
SESSION_EXPIRE_AT_BROWSER_CLOSE = True
SESSION_COOKIE_AGE = 180
```

Con *age* si intende la durata della validità, in secondi, della sessione, prima che il sito chieda di nuovo all'utente di effettuare l'accesso.

Function-based views: La mia scelta riguardo la scrittura delle views è ricaduta su quelle *function-based*. Questo perchè fin da subito mi sono parse una scelta più *ordinata* e *comoda*. Soprattutto, in fase di scrittura mi sembrava di avere di più il controllo sulla logica vera e propria della view.

Tipi di utenti: Gli utenti, all'interno del sito, sono di 3 tipi:

- **Anonimo**: L'utente *anonimo* è un utente coi permessi di base. Gli è concesso entrare nel catalogo attraverso il pulsante *Entra come guest*, può solamente sfogliare il catalogo dei film e consultarne i dettagli, come titolo, valutazione media, anno di uscita...
- **Loggato**: Un utente *loggato* ha invece la possibilità di guardare un film, attraverso un pulsante presente difianco al film, nel catalogo. Ha la possibilità di consultare la propria pagina *Il mio account*, per avere la lista dei film da egli guardati, le recensioni lasciate e i film suggeriti dal recommendation system. Sempre in questa pagina è presente anche il grafico reso disponibile da Chart.js, che mostra in modo user-friendly i generi preferiti.
- **Admin**: L'utente *admin* è inteso come l'amministratore/gestore del sito. Ha il permesso di aggiungere, modificare ed eliminare utenti, film e generi.

La gestione degli utenti è stata nettamente diversificata da quella di *Django*, non è presente infatti la concezione di *AnonymousUser* vero e proprio (fornito per l'appunto da Django stesso). Questo perchè sono sorte alcune problematiche con i form '*puri*' di Django, come ad esempio una possibile ma verbosa estensività, sia nei form che nell'autenticazione.
La soluzione è stata quindi quella di gestire manualmente gli utenti, inserendo nei template dei classici form HTML, i cui valori sarebbero poi stati inviati alle view corrispondenti attraverso **GET** o **POST**.
Tornando alla gestione degli utenti, essi sono stati gestiti, come detto, manualmente, l'utente inserirà (in caso di login), il proprio *username* e *password* e verrà fatto un controllo dalla view corrispondente facendo un check del database. Questa gestione ha sicuramente pro e contro, perchè in certe situazioni era più comodo controllare direttamente il parametro '*user*' della request, per capire se era presente un utente loggato o meno. Al tempo stesso però, una gestione del genere mi ha dato più '*awareness*' su ciò che andavo a controllare.

Status codes: Gli status codes sono una parte fondamentale nello sviluppo di un applicativo web, dato che forniscono il modo per diversificare l'esito di varie operazioni. Essi sono stati utili durante la scrittura dei test, dato che mi è stato possibile usarli per confermare che l'esito di una determinata operazione fosse quello atteso. Collegandomi a loro, ho evitato l'uso del decoratore *@login_required*, perchè come dice il nome esso controlla solamente se è presente un utente loggato in quel momento. Questo porta due problemi:

1. Gestendo manualmente gli utenti e non facendo quindi uso del *Django user*, il decoratore non aveva senso.

2. La maggior parte delle mie views ha più status code. Ad esempio, se viene cliccato il pulsante '*Chatta*', l'utente oltre ad essere loggato deve anche aver guardato il film. Un'implementazione sarebbe comunque stata possibile, ma l'aver gestito manualmente i controlli credo abbia semplificato molto la creazione dei test e la l'ordine della view in sè.

Template tags: Un template tag è una sorta di funzione applicabile ad un template. Nel mio caso ne ho implementata solamente una, chiamata

replace_titles, la sua funzione è quella di prendere in input il titolo di un film, che avrà dei _ al posto dei whitespace (per comodità nei link, quando passato come parametro GET), e sostituirà quest'ultimi con dei whitespace appropriati, si tratta quindi solamente di un workaround *grafico*.

Modelli

All'interno del progetto sono presenti 4 entità:

- **Utente**: Un *utente* è definito dai seguenti *fields*:
  - **username**
  - **email**
  - **password**
  - **nome**
  - **cognome**

- **Film**:
  - **titolo**
  - **generi**
  - **anno_uscita**
  - **trama**

- **Genere**:
  - **name**: Rappresenta il nome del genere, e.g *Azione, avventura...*

- **Recensione**:
  - **voto**: Rappresenta il voto numerico della recensione, con range 1∼5.
  - **film**: *Foreign Key* che specifica il film recensito.
  - **utente**: *Foreign Key* che specifica l'utente autore della recensione.
  - **commento_scritto**: Commento scritto opzionale.

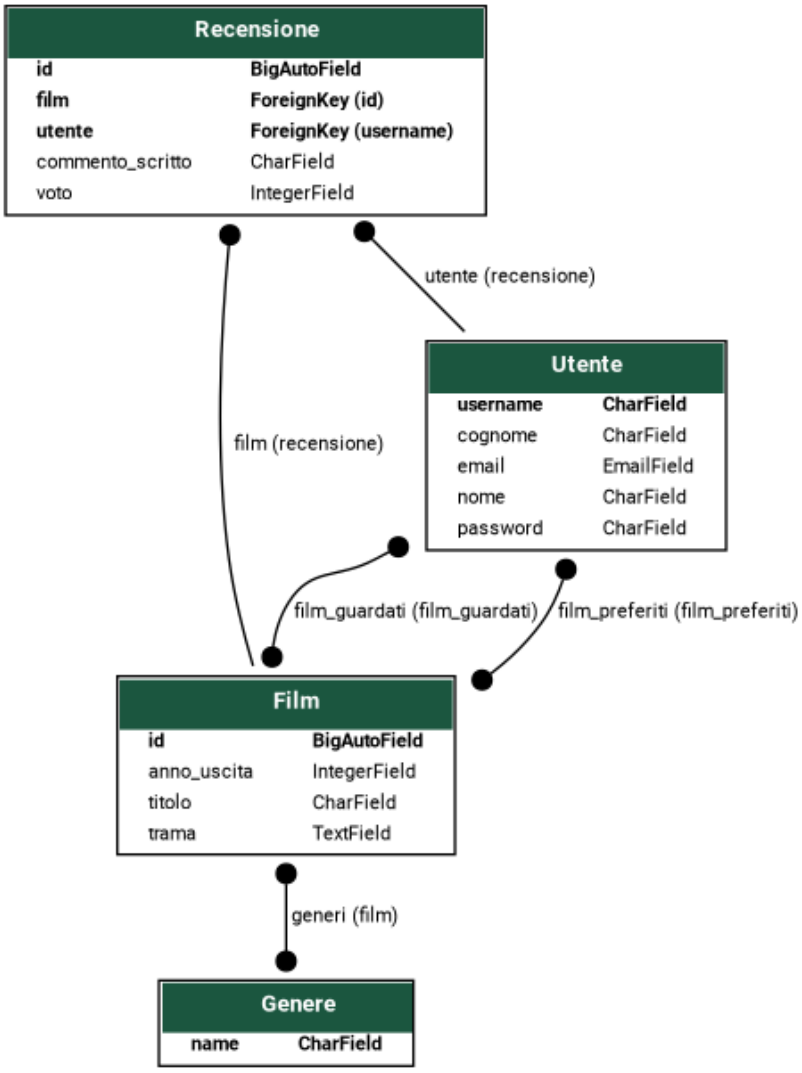
Testing

Come specifica nel paragrafo riguardo alla divisione in app, i test sono stati separati dal resto delle applicazioni, questo ha permesso di facilitare la loro gestione, permettendomi di raggrupparli tutti sotto la stessa app. Dentro il path ./test_apps/tests/, possiamo trovare varie directory.
Ho deciso di separare anche qui i test al fine di tenerli ordinati e poter, su richiesta, eseguire test riguardanti un preciso/solo alcuni argomenti.
Per i test mi sono affidato a *Django Unit Test Framework*, che sfrutta nativamente *unittest*. I test totali sono 33.
Sono presenti più test per ogni singola view, dato che è stato testato ogni possibile aspetto di ogni view, ovvero il caso di *success*, *fail* e *conflict*.

Per orientarmi durante la scrittura di un test, è stato utile assegnare lo status code manualmente ad ogni render delle view:

- Se l'utente effettua l'accesso con successo, lo status code sarà 200 (**Successo**).
- Se l'utente inserisce credenziali errate, lo status code sarà 401 (**Unauthorized**).
- Se l'utente si registra con username o email già in uso, lo status code sarà 409 (**Conflitto**).

Schemi



Schema UML