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 di tutte le recensioni lasciate dall'utente. Sarà inoltre possibile modificare i valori al suo interno, in caso l'utente avesse cambiato idea, oppure eliminarla completamente attraverso il pulsante dedicato.
- **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.

- **AJAX**: Più che una tecnologia può essere vista come un'implementazione (o estensione) di JavaScript, che permette di creare richieste ad un server. E' molto importante perchè è uno strumento molto comodo quando si vuole interconnettere backend e frontend attraverso JavaScript, si pensi ad esempio al fornire al backend valori (più o meno complessi) ottenuti da elementi HTML quali tabelle o div. Facendo qualche esempio, in questo caso è stato utilizzato durante la fase di modifica ed eliminazione di una recensione, recuperando i valori dalla table attraverso JavaScript 'puro' e inviandoli con AJAX ad una Django view.

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.

- **SonarQube**: E' un software dedito all'ispezione di codice al fine di migliorarne la qualità. Permette anche di controllare il coverage, però essendo un po' complicato il link con Django, ho preferito gestire questo fattore attraverso il package 'Coverage.py', mentre ho usato SonarQube per tutti gli altri check sul codice, come il *naming* delle variabili, lo *smell code index*, quantità di *bug* rilevati...

- **Django**: E' il framework *cuore* del progetto, dato che si occupa dell'interfacciamento con il database, della navigazione web, collegando ogni URL ad una view, della creazione dei modelli, creazione del superuser e del suo '*admin panel*' (...)

Divisione nelle varie app

Il progetto è stato diviso in 4 app:

- **Streamify**: È l'app principale del progetto, essa contiene tutti i templates e views, escluse quelle relative all'autenticazione e chat.
- **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 aggiuntivo'*, un po' come se fosse una feature esterna rispetto allo scopo *principale* del progetto.
- **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 tre generi preferiti da utente, dato che difficilmente egli vorrà ricevere consigli su generi che non gli interessano.
Questa lista di tre 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,
'Horror': 4,2
}

U2: {
'Azione': 4.4,
'Avventura': 5.0,
'Fantascienza': 4,1
}
```

Ipotizzando che i tre generi preferiti dei due utenti siano quelli sopra citati, verrà trovato un match per i primi due generi, 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: È stato fatto uso delle sessioni attraverso il dizionario associato a *request*, ovvero *request.session*. In esso sono stati salvati i valori che sarebbero stati utili durante tutta la durata della sessione attuale, al fine di poterli condividere con le varie views. Alcuni esempi sono:

- **recommended_films** –> Contiene la lista di film consigliati dal recommendation system
- **generi** –> Contiene il dizionario con coppie {genere: quantità_film_visti} per il disegno del grafico nell'account page.

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*, o anche *AnonymousUser* 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 in corrispondenza del 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 ripresa da quella *base* di Django, ovvero il modello *Utente* altro non è che un'estensione della classe *AbstractBaseUser*, fornito direttamente da Django.
Le classi da cui un custom user può essere esteso sono AbstractBaseUser e AbstractUser, nel mio caso ho scelto la prima, dato che si tratta di una versione quanto più *basic* possibile, al fine di poterla personalizzare a mio piacimento.
Se si fa uso di AbstractUser/AbstractBaseUser, bisogna specificare a Django quale modello verrà interpretato come utente, ciò è possibile inserendo nel file *settings.py* la seguente riga:

AUTH_USER_MODEL = 'streamify.Utente'

Estendere un utente da AbstractBaseUser porta notevoli benefit rispetto a crearlo *from-scratch*, dato che fornisce la possibilità di usare vari decoratori (anch'essi già forniti da Django) per il controllo sugli utenti. Ad esempio:

- **login_required**: Come ci suggerisce il nome, viene usato per decorare una funzione che necessita di un utente effettivamente loggato per essere eseguita.

- **authenticate**: Altro non è che una funzione che prende in input un set di credenziali (solitamente username/email e password), fa un check nel database cercando nel modello definito come utente e, in caso di match, ritorna l'oggetto desiderato.

- **login**: Una volta aver ottenuto un utente attraverso authenticate() e quindi essersi accertati che, dato il set di credenziali, esista effettivamente un tale utente nel database, sarà possibile *assegnarlo* alla sessione corrente, attraverso appunto questa funzione.

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. Possono essere inseriti manualmente e passati come parametro durante il return del render di una view, con parametro *status*.

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**: Il modello Utente eredita dalla classe padre *built-in* di Django, **AbstractBaseUser**. La particolarità di quest'ultima, rispetto a **AbstractUser**, è l'essere il più minimale possibile, esso infatti è composto solamente di *password* e *last_ login*.

  Un utente è stato definito dai seguenti *fields*:

  - **username**
  - **email**
  - **password**
  - **nome**
  - **cognome**
  - **is_active** –> Alternativa all'eliminazione dal DB di un utente in situazioni come un *blocco temporaneo* dell'account.

- **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.

E' presente poi lo UserManager, ovvero un'estensione della classe padre BaseUserManager, il cui scopo è quello di gestire la creazione di nuovi utenti. Esso verrà chiamato durante la creazione di un utente attraverso la seguente operazione, presente nel modello Utente:
objects = UserManager()

Lo UserManager creerà attraverso il metodo *_create_user(self, email, password, **extra_field)*, l'utente. Inoltre, setterà la password con il metodo *set_password()*, che effettuerà l'hash della password per aumentarne la sicurezza.

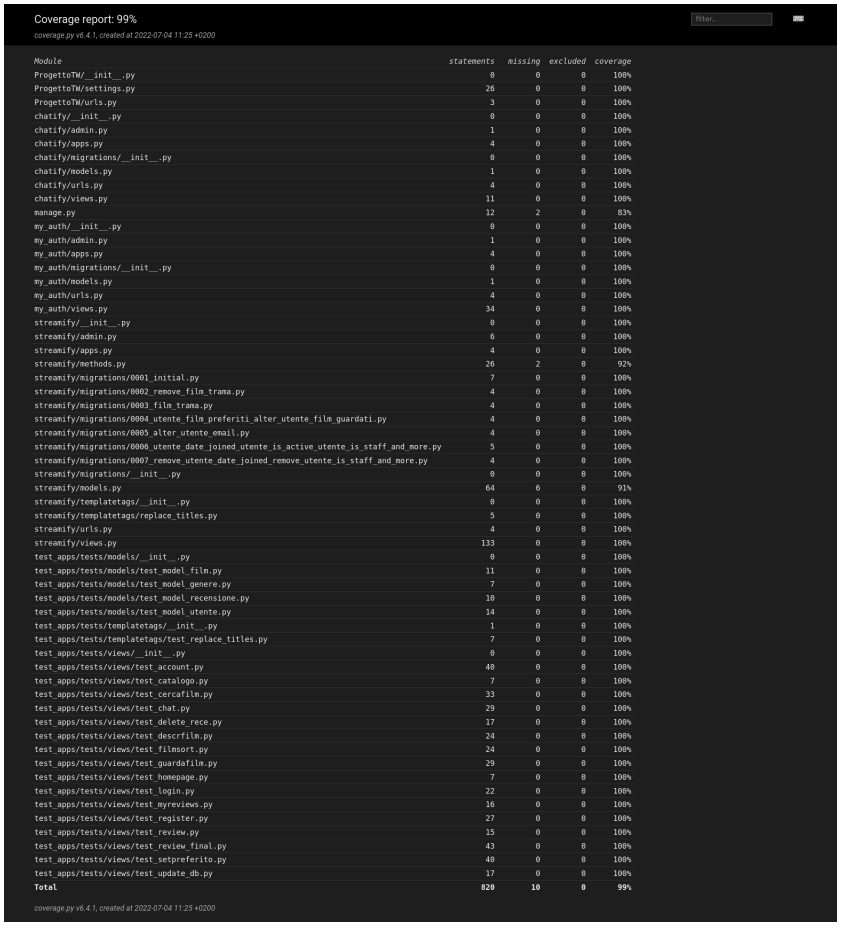
Testing

Come specificato nel paragrafo riguardo alla divisione in più 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/alcuni argomenti.
Per i test mi sono affidato a *Django Unit Test Framework*, che sfrutta nativamente *unittest*. 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, alcuni esempi:

- 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**).

I test verranno *eseguiti* da **coverage.py** un package fornito da PIP, con cui Django si è dimostrato veramente ben integrato. Coverage può cercare, attraverso il parametro '–source' all'interno di path specifici, motivo per cui ho tenuto a separare i vari tipi di test. Di seguito, verrà riportato un resoconto del coverage dell'intero progetto:



Coverage report: 99%

coverage.py v6.4.1, created at 2022-07-04 11:25 +0200

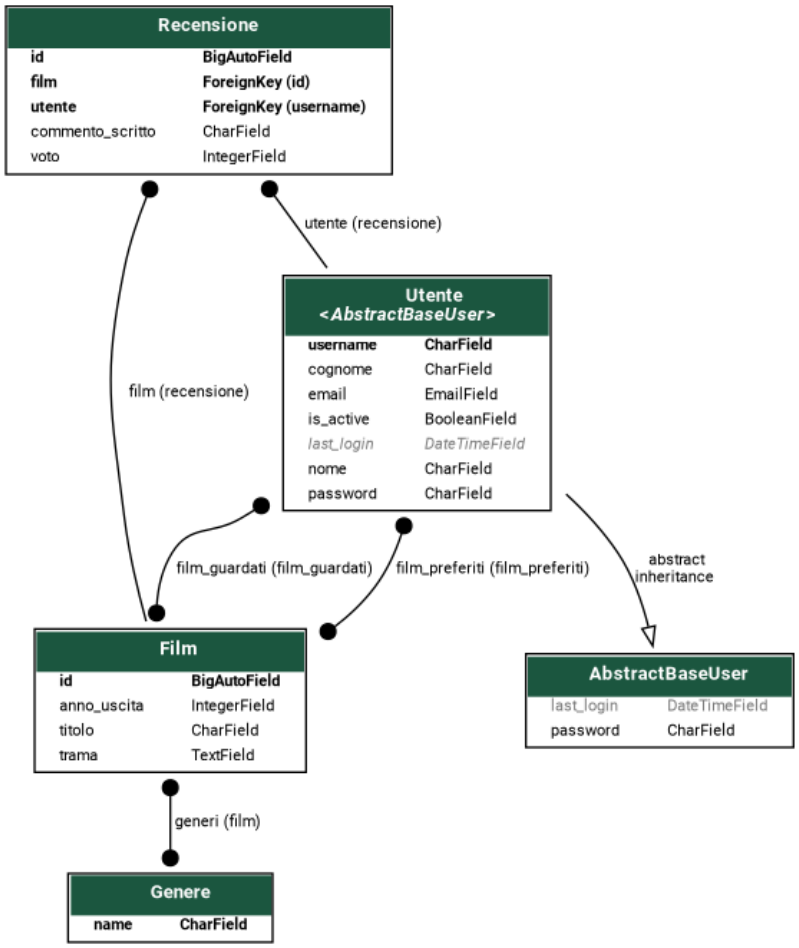
filter...

Module	statements	missing	excluded	coverage
ProgettoTW/__init__.py	0	0	0	100%
ProgettoTW/settings.py	26	0	0	100%
ProgettoTW/urls.py	3	0	0	100%
chatify/__init__.py	0	0	0	100%
chatify/admin.py	1	0	0	100%
chatify/apps.py	4	0	0	100%
chatify/migrations/__init__.py	0	0	0	100%
chatify/models.py	1	0	0	100%
chatify/urls.py	4	0	0	100%
chatify/views.py	11	0	0	100%
manage.py	12	2	0	83%
my_auth/__init__.py	0	0	0	100%
my_auth/admin.py	1	0	0	100%
my_auth/apps.py	4	0	0	100%
my_auth/migrations/__init__.py	0	0	0	100%
my_auth/models.py	1	0	0	100%
my_auth/urls.py	4	0	0	100%
my_auth/views.py	34	0	0	100%
streamify/__init__.py	0	0	0	100%
streamify/admin.py	6	0	0	100%
streamify/apps.py	4	0	0	100%
streamify/methods.py	26	2	0	92%
streamify/migrations/0001_initial.py	7	0	0	100%
streamify/migrations/0002_remove_film_trama.py	4	0	0	100%
streamify/migrations/0003_film_trama.py	4	0	0	100%
streamify/migrations/0004_utente_film_preferiti_alter_utente_film_guardati.py	4	0	0	100%
streamify/migrations/0005_alter_utente_email.py	4	0	0	100%
streamify/migrations/0006_utente_date_joined_utente_is_active_utente_is_staff_and_more.py	5	0	0	100%
streamify/migrations/0007_remove_utente_date_joined_remove_utente_is_staff_and_more.py	4	0	0	100%
streamify/migrations/__init__.py	0	0	0	100%
streamify/models.py	64	6	0	91%
streamify/templatetags/__init__.py	0	0	0	100%
streamify/templatetags/replace_titles.py	5	0	0	100%
streamify/urls.py	4	0	0	100%
streamify/views.py	133	0	0	100%
test_apps/tests/models/__init__.py	0	0	0	100%
test_apps/tests/models/test_model_film.py	11	0	0	100%
test_apps/tests/models/test_model_genere.py	7	0	0	100%
test_apps/tests/models/test_model_recensione.py	10	0	0	100%
test_apps/tests/models/test_model_utente.py	14	0	0	100%
test_apps/tests/templatetags/__init__.py	1	0	0	100%
test_apps/tests/templatetags/test_replace_titles.py	7	0	0	100%
test_apps/tests/views/__init__.py	0	0	0	100%
test_apps/tests/views/test_account.py	40	0	0	100%
test_apps/tests/views/test_catalogo.py	7	0	0	100%
test_apps/tests/views/test_cercafilm.py	33	0	0	100%
test_apps/tests/views/test_chat.py	29	0	0	100%
test_apps/tests/views/test_delete_rece.py	17	0	0	100%
test_apps/tests/views/test_descrfilm.py	24	0	0	100%
test_apps/tests/views/test_filmsort.py	24	0	0	100%
test_apps/tests/views/test_guardafilm.py	29	0	0	100%
test_apps/tests/views/test_homepage.py	7	0	0	100%
test_apps/tests/views/test_login.py	22	0	0	100%
test_apps/tests/views/test_myreviews.py	16	0	0	100%
test_apps/tests/views/test_register.py	27	0	0	100%
test_apps/tests/views/test_review.py	15	0	0	100%
test_apps/tests/views/test_review_final.py	43	0	0	100%
test_apps/tests/views/test_setpreferito.py	40	0	0	100%
test_apps/tests/views/test_update_db.py	17	0	0	100%
Total	**820**	**10**	**0**	**99%**

coverage.py v6.4.1, created at 2022-07-04 11:25 +0200

Coverage report

Schemi



Schema UML