



---

# MOVIEMORE

---

Progetto Tecnologie Web



4 LUGLIO 2024  
NERI RICCARDO  
Mat. 165343

# Introduzione

## Idea generale

Il progetto simula una piattaforma di un **cinema** per la visualizzazione, amministrazione e generazione delle diverse proiezioni programmate ed il loro conseguente acquisto da parte di eventuali clienti.

Ovviamente è stato utilizzato il web framework **Django**. Grazie a quest'ultimo hanno preso forma il backend ed il frontend, con annesse funzionalità logiche e grafiche, rendendo l'implementazione consona ai requisiti richiesti.

## Requisiti

La traccia dell'applicazione prevedeva diversi punti su cui focalizzarsi per trovare soluzioni adatte alla loro realizzazione:

- Applicazione Web dinamica basata sull'interazione con un database  
Utilizzo di un DB su cui eseguire operazioni CRUD sulle entità presenti (utenti, sale del cinema, film, proiezioni programmate e prenotazioni/acquisti da parte degli utenti)
- Richiesta presenza di utenti anonimi e registrati con diritto di accesso differenziato  
L'applicazione denota la presenza di quattro tipologie di utenti:
  - *Admin (superuser)*: controlla l'intero sistema e ha il permesso di eseguire tutte le operazioni degli utenti sottostanti. È l'unico che può creare utenti di tipo "Amministratori"
  - *Amministratori*: possono aggiungere, modificare ed eliminare film, sale cinematografiche e proiezioni. Sono dotati di un pannello di amministrazione tramite cui eseguire le azioni sopracitate
  - *Utenti registrati*: utilizzando un sistema di login possono accedere al proprio profilo personale e personalizzarlo a proprio piacimento. Una volta effettuato l'accesso hanno la possibilità di visualizzare ed acquistare le proiezioni (implementato un sistema che simula un pagamento che richiede l'inserimento delle credenziali della carta di credito per effettuare la transazione)
  - *Utenti anonimi*: tramite un sistema di registrazione possono creare un proprio profilo personale, trasformandosi a tutti gli effetti in user di tipo "Utenti Registrati". Possono visualizzare le proiezioni ma non acquistarle (sono necessari la registrazione o il login).

Tramite la suddivisione dei **permessi** e la **protezione** delle risorse è stato quindi possibile creare un sito web che contempla l'esistenza di più tipologie di utente, ognuno con i propri ruoli e scopi.

(vedere “Figura 1”, “Figura 2”, “Figura 3”, “Figura 4” e “Tabella 1” per ulteriori informazioni)

- *Piattaforma di admin riservata all'amministratore*

L'Admin dell'applicazione può accedere al proprio pannello riservato visitando l'indirizzo `localhost:8000/admin/`

Tale piattaforma è fornita automaticamente da Django e permette al superuser di aggiungere/rimuovere gruppi, users ed i relativi permessi associati. Nello specifico sono stati creati due gruppi:

- `AmministratoriCinema`: popolato dagli users di tipo “Amministratori” direttamente creati dall'Admin
- `UtentiRegistrati`: popolato dagli users che creano un proprio profilo tramite il sistema di registrazione fornito dal sito web

La suddivisione dei permessi si basa sull'appartenenza ai gruppi (l'Admin è compreso in quello dedicato agli amministratori) ed in base al flag `is_superuser`. In questo modo tutte le viste implementate sono protette da accessi non autorizzati. (vedere “Tabella 2” e per ulteriori informazioni)

- *Richieste possibilità di caricamento immagini da utente*

Il caricamento di immagini scelte dall'utente può avvenire in due casi: il primo prevede l'upload di un'eventuale immagine del profilo quando si desidera personalizzarlo, il secondo prevede l'upload di un'eventuale copertina/poster di un nuovo film che si intendere inserire nel database. Il caricamento è implementato tramite un *file chooser* che permette di sfogliare i vari file locali.

- *Richiesta produzione di test*

Django include un framework di testing basato su **unittest**, una libreria standard di Python, che rende facile creare e gestire test per la propria applicazione. Come da richiesta sono stati implementati due test, uno su una funzione e uno su una view.

- *Notifiche/avvisi di conferma/errore durante la progettazione*

A seguito di operazioni da parte dell'utente (registrazione, login, logout, aggiunta/modifica/eliminazione di un oggetto, modifica profilo personale, azioni non concesse, errori) è presente un sistema di pop-up per segnalare i loro esiti. Per implementare questa funzionalità è stato necessario ricorrere alla libreria JavaScript **Toastr**.

Nei successivi paragrafi vedremo alcuni dei punti appena descritti più nello specifico.

## Permessi

### Autorizzazioni ed Use Case UML

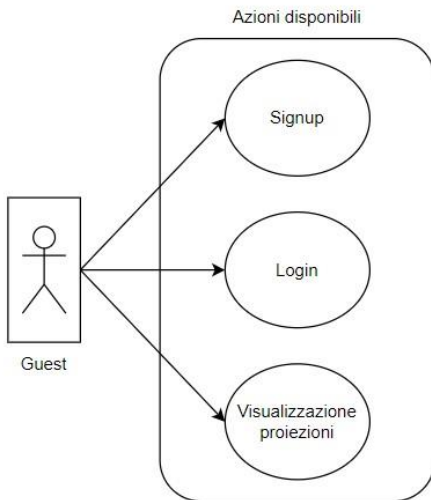
Focalizziamoci maggiormente sulla parte relativa alle autorizzazioni: come detto in precedenza esistono quattro tipologie diverse di utente, ognuna con i propri permessi e funzionalità. Ciò che rende possibile la loro distinzione è l'appartenenza ai gruppi citati sopra, oltre ad eventuali flag forniti di default da Django.

Di seguito è possibile trovare la rappresentazione della distribuzione dei permessi ed i vari Use Case UML per ogni varietà di user:

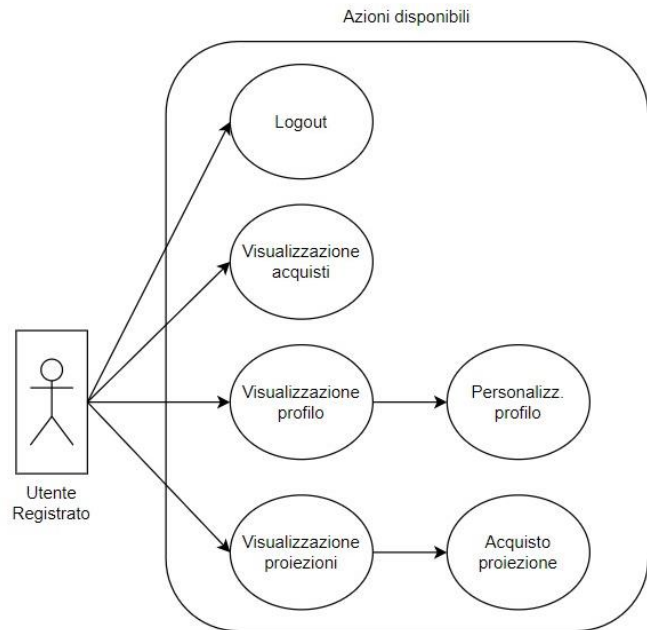
Tipo	movie	screening room	screening	booking	user	userprofile
<i>Guest</i>	Ø	Ø	<u>View</u> della programmazione del cinema	Ø	<u>Create</u> di un nuovo "Utente Registrato"	Ø
<i>Utente Registrato</i>	Ø	Ø	<u>View</u> della programmazione del cinema	<u>View</u> dei propri acquisti, <u>Create</u> nuovo acquisto	Ø	<u>View</u> , <u>Upload</u> del proprio profilo
<i>Amministratore Cinema</i>	<u>View</u> <u>Create</u> <u>Update</u> <u>Delete</u>	<u>View</u> <u>Create</u> <u>Update</u> <u>Delete</u>	<u>View</u> <u>Create</u> <u>Update</u> <u>Delete</u>	<u>View</u> dei propri acquisti, <u>Create</u> nuovo acquisto	Ø	<u>View</u> , <u>Upload</u> del proprio profilo
<i>Admin (superuser)</i>	<u>View</u> <u>Create</u> <u>Update</u> <u>Delete</u>	<u>View</u> <u>Create</u> <u>Update</u> <u>Delete</u>	<u>View</u> <u>Create</u> <u>Update</u> <u>Delete</u>	<u>View</u> dei propri acquisti, <u>Create</u> nuovo acquisto	<u>View</u> <u>Create</u> <u>Update</u> <u>Delete</u>	<u>View</u> , <u>Upload</u> del proprio profilo

N.B. → "Guest" e "Utente Registrato" possono visualizzare SOLO le informazioni essenziali di una certa proiezione (contenenti anche il nome della sala e dati sul film proiettato)  
 "Admin" eliminando un user eliminerà automaticamente anche il suo *profile*

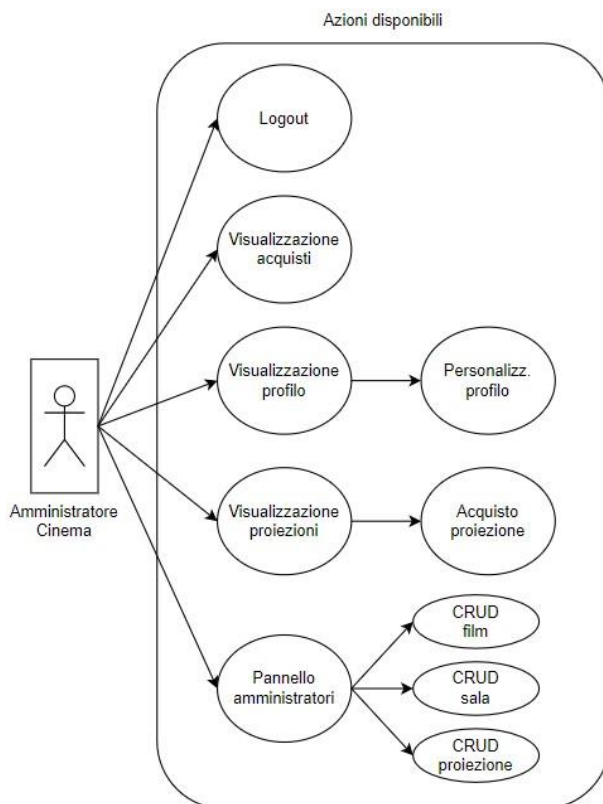
**Tabella 1:** Autorizzazioni in base al tipo di utente



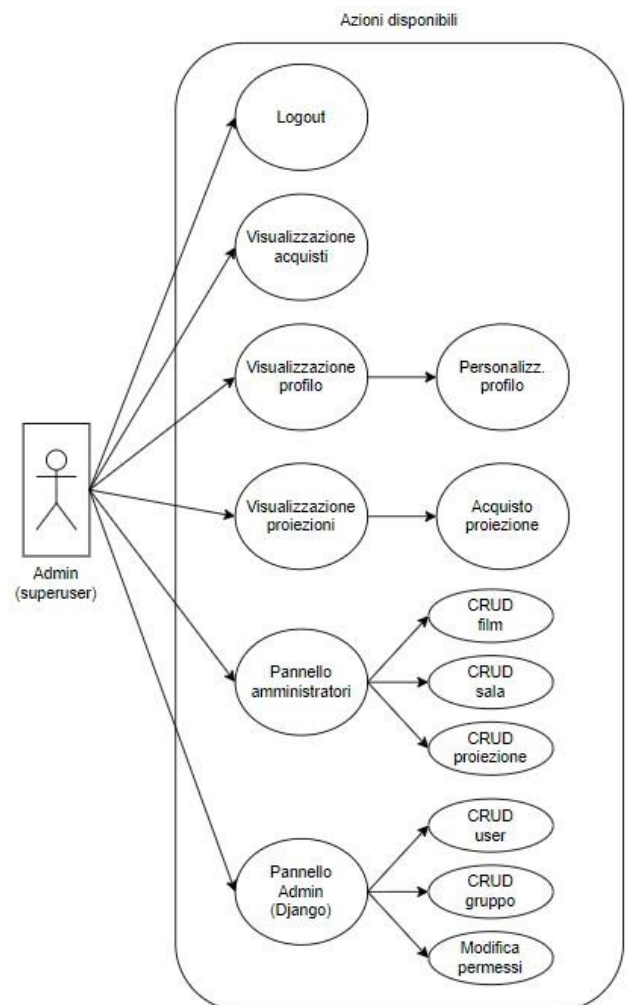
**Figura 1: Use Case "Guest"**



**Figura 2: Use Case "Utente Registrato"**



**Figura 1: Use Case "Amministratore Cinema"**



**Figura 2: Use Case "Admin"**

Tipo	Condizioni
Guest	$\text{!is\_authenticated}$
Utente Registrato	$\text{is\_authenticated}$
Amministratore Cinema	$\text{group\_required} \cap \text{user\_groups}$
Admin (superuser)	$\text{is\_superuser} + \text{is\_staff} + (\text{group\_required} \cap \text{user\_groups})$

N.B. →  $\text{group\_required} = \text{AmministratoriCinema}$   
 $\text{user\_groups}$  equivale ai gruppi di cui fa parte un certo utente  
 $\text{group\_required} \cap \text{user\_groups}$  intersezione tra gli insiemi, se ritorna un insieme NON vuoto allora soddisfa la condizione

**Tabella 2:** Condizioni di appartenenza in base al tipo di utente

## Database (DB)

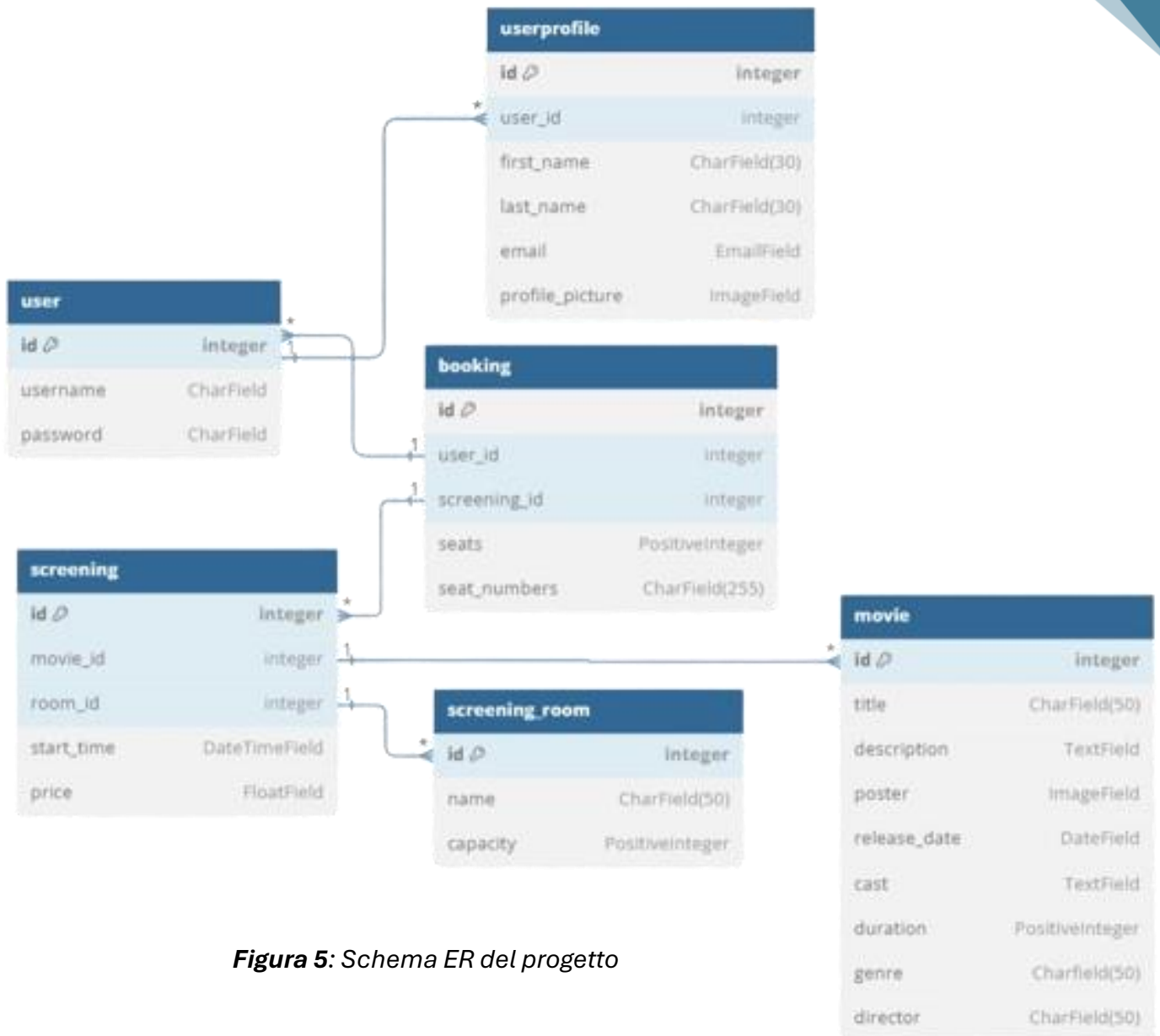
Nel contesto del progetto è stato scelto di utilizzare SQLite come Database Management System (DBMS). La scelta è stata guidata da una serie di considerazioni pratiche e tecniche che ne fanno una soluzione ideale per lo sviluppo di applicazioni web di piccola/media scala.

SQLite si è dimostrato essere una soluzione ideale per lo sviluppo iniziale e il testing dell'applicazione, fornendo un ambiente di sviluppo rapido ed efficiente. Tuttavia, per progetti con requisiti di scalabilità più elevati, potrebbe essere necessario considerare l'adozione di un DBMS più robusto e scalabile.

## Diagramma ER

Lo schema ER sottostante (vedi “Figura 5”) denota la presenza di 6 entità:

- user: rappresenta gli utenti che possono accedere al sistema. Contiene le informazioni necessarie per l'autenticazione
- userprofile: memorizza informazioni aggiuntive sul profilo dell'utente
- movie: contiene informazioni dettagliate sui film disponibili per la proiezione
- screening\_room: rappresenta le sale del cinema dove vengono proiettati i film
- screening: memorizza le informazioni sulle proiezioni dei film, comprese le sale e gli orari di proiezione
- booking: gestisce le prenotazioni/acquisti effettuate dagli utenti per le proiezioni, compresi i dettagli sui posti prenotati/acquistati



**Figura 5:** Schema ER del progetto

## Classi

A livello implementativo lo schema delle classi si discosta leggermente dalla figura sovrastante. Le entità *userprofile*, *screening*, *booking*, *screening\_room* e *movie* corrispondono completamente a classi definite nei vari files *models.py* del progetto. La differenza principale è quella relativa ad *user* poiché non è stata definita nessuna classe di tale tipo.

Infatti, è stato utilizzato l'utente (*User*) che viene già fornito da Django in automatico. Django include un modello utente predefinito (`django.contrib.auth.models.User`) con lo scopo di gestire gli utenti.

Questo modello predefinito offre tutte le funzionalità necessarie per tale obiettivo rendendo non necessario definire un modello utente personalizzato. Ciò permette di sfruttare una soluzione robusta e ben integrata, riducendo la complessità del codice e garantendo una migliore manutenibilità e sicurezza del sistema. Questo approccio, combinato con la definizione di classi personalizzate per le altre entità, fornisce una base solida e flessibile per lo sviluppo dell'applicazione.

## Tecnologie

Nel progetto Django sviluppato, è stato impiegato un insieme variegato di tecnologie sia lato frontend che backend, ognuna delle quali è stata scelta per soddisfare specifiche esigenze funzionali e di design.

Di seguito viene presentata una panoramica dettagliata delle tecnologie utilizzate e delle motivazioni alla base della loro scelta.

### Backend

#### Django Framework

Django è stato scelto come framework principale per il back-end grazie alla sua robustezza e versatilità. Fornisce un'architettura strutturata che facilita lo sviluppo rapido di applicazioni web sicure e scalabili. Django segue il principio "Don't Repeat Yourself" (DRY), riducendo la ridondanza e aumentando l'efficienza del codice. Inoltre, offre un potente ORM (Object-Relational Mapping) che permette di interagire con il database in modo intuitivo e senza dover scrivere query SQL manuali.

#### SQLite Database

Per la gestione del database, è stato utilizzato SQLite, il database di default fornito da Django. SQLite è leggero e facile da configurare, rendendolo ideale per lo sviluppo e il testing. Sebbene sia adatto principalmente per applicazioni di piccola e media scala, la sua semplicità di utilizzo lo rende una scelta pratica per lo sviluppo iniziale.

(Per maggiori informazioni vedi paragrafo "[\*Database \(DB\)\*](#)")

#### Django Signals

Django Signals è una potente funzionalità che consente di eseguire azioni in risposta a determinati eventi all'interno dell'applicazione. Nel progetto, i segnali sono stati utilizzati per creare automaticamente un profilo utente (*userprofile*) ogni volta che viene creato un nuovo utente. In particolare, è stato utilizzato il segnale `post_save` per assicurare che ogni utente abbia un profilo associato, garantendo così la consistenza dei dati.

La configurazione di questa tecnologia si trova nel file *signals.py* all'interno dell'app *Users*.



## Django Crispy Forms

Django Crispy Forms è stato utilizzato per migliorare la gestione e il rendering dei form. Questo pacchetto permette di stilizzarli facilmente utilizzando framework CSS come Bootstrap, migliorandone l'estetica e l'usabilità all'interno dell'applicazione.

È stato necessario includere nelle “INSTALLED\_APPS” (in *settings.py*) “*crispy\_forms*” e “*crispy\_bootstrap4*” (CRISPY\_TEMPLATE\_PACK = 'bootstrap4').

(Per maggiori informazioni relative a dove sono stati utilizzati vedi paragrafo “*Pagine ed urls*”)

## Django Messages

Django Messages è un framework integrato che permette di gestire messaggi temporanei tra le richieste. Questo è stato utilizzato per fornire feedback agli utenti riguardo alle loro azioni, come la conferma di avvenute operazioni o avvisi di errori:

- login, logout e registrazione
- aggiunta/modifica/rimozione di un film, sala, proiezione
- modifica del profilo personale
- segnalazione di errori nel caso in cui si voglia acquistare una proiezione sold out, si tenti di accedere se si è già “loggati”, si tenti di registrare un nuovo profilo se si è “loggati”, tentare di “sloggarsi” se non si è “loggati”

## JSON

Durante l'inizializzazione del database, è stato utilizzato JSON per popolare i dati. Questo formato è stato scelto per la sua leggibilità e facilità di manipolazione, permettendo di importare dati strutturati in modo efficiente e organizzato. In particolare, all'interno del file *Initialization.py* sono presenti due funzioni che generano i file json per le proiezioni e gli acquisti ogni volta che viene eseguito *setup.py*, rendendo il sito web “diverso” ogni qualvolta che lo si desidera.

Tutti i file json sono raggruppati all'interno della cartella `static/json/`.

## Frontend

### HTML

HTML è stato utilizzato per strutturare i template dell'applicazione. I template HTML servono come scheletri delle pagine web, definendo la disposizione degli elementi e integrando i dati dinamici provenienti dal backend tramite le richieste GET e POST. L'uso dei template HTML di Django ha permesso di creare pagine web dinamiche e interattive.

### CSS e Bootstrap 5

Il CSS è stato impiegato per stilizzare i template HTML, assicurando che l'interfaccia utente fosse visivamente attraente e coerente. Bootstrap 5, un framework CSS molto popolare, è stato utilizzato per sfruttare componenti predefiniti e responsivi. Tuttavia, i componenti di Bootstrap sono stati personalizzati per adattarsi alle esigenze specifiche del progetto, sia dal punto di vista grafico che funzionale.

(Per maggiori informazioni vedi i paragrafi “*Interfaccia*” e “*Design*”).

## JavaScript

JavaScript è stato utilizzato all'interno di alcuni template per aggiungere interattività e migliorare l'esperienza utente. Script personalizzati hanno permesso di gestire eventi, validare dati lato client e manipolare il DOM, rendendo l'applicazione più dinamica e reattiva. Questa tecnologia è presente nei template che richiedono la visualizzazione di messaggi (vedi *Toastr* sotto) ed all'interno del template per l'acquisto. Al suo interno infatti è presente un breve script che permette di aggiornare in numero di biglietti che si intendono acquistare tramite due bottoni ( - e + ), modificando dinamicamente tale quantità.

## Toastr

Toastr è una libreria JavaScript per notifiche non bloccanti. È stata integrata nei template che richiedono la visualizzazione di avvisi, fornendo un metodo elegante e user-friendly per notificare gli utenti riguardo a varie azioni e stati dell'applicazione, come successi, errori o informazioni.

Si possono trovare script che usano tale tecnologia all'interno dei template dell'home page, del proprio profilo e nel pannello degli amministratori.

## Struttura del codice

Affinché il progetto sfrutti il più possibile una migliore organizzazione del codice, maggiore riusabilità e facilità di manutenzione, esso è stato suddiviso in cinque app principali:

- **cinema:** è l'app principale, al suo interno sono presenti i *settings* del progetto, i vari *urls* per raggiungere le pagine implementate e tutte le *views* principali relative alla "home page", al pannello degli amministratori ed alcune funzioni per la gestione dei permessi e degli errori. Inoltre, è presente un file (*initialization.py*) che ha lo scopo di definire le funzioni *init\_db()* ed *erase\_db()*, fondamentali per l'inizializzazione ed il "reset" del database ogni volta che viene richiamato lo script *setup.py*
- **movies:** app per la rappresentazione/gestione dei film, contiene tutte le *views*, i *forms*, i *models* ed i *templates* relativi ad essi
- **screenings:** app per la rappresentazione/gestione delle proiezioni e delle sale del cinema, contiene tutte le *views*, i *forms*, i *models* ed i *templates* relativi ad esse. Inoltre, va considerato anche *tests.py* in cui è stato definito un setup per testare una funzione implementata nel corso dello sviluppo del progetto (per ulteriori informazioni vedi il paragrafo "*Test*")
- **bookings:** app per la rappresentazione/gestione delle prenotazioni (acquisti), contiene tutte le *views*, i *forms*, i *models* ed i *templates* relativi ad esse. Inoltre, va considerato anche *tests.py* in cui è stato definito un setup per testare una vista

implementata nel corso dello sviluppo del progetto (per ulteriori informazioni vedi il paragrafo “*Test*”)

- **users:** app per la rappresentazione/gestione degli user e dei profili associati, contiene tutte le *views*, i *forms*, i *models* ed i *templates* relativi ad essi. È presente anche il file *signals.py* citato precedentemente nel paragrafo relativo alle tecnologie utilizzate

Oltre alle app vi sono anche due cartelle, *static* e *templates*, che rispettivamente contengono tutti i file css, png/jpg, json utili ai fini del progetto e tutti i template relativi all'app *cinema* descritta sopra.

Inoltre è anche presente il file *setup.py* citato sopra, esso ha lo scopo di cancellare i vari record dal DB e successivamente di ripopolarlo.

Per concludere, la totalità delle interazioni tra client e server avviene mediante richiesta-risposta HTTP.

## Interfaccia

La GUI del sito web è stata pensata per essere il più user friendly possibile, garantendo navigabilità ed intuitività.

Alcune implementazioni sono mostrate nel paragrafo relativo (“*Risultati finali*”).

## Pagine ed urls

Sostanzialmente, le pagine del sito sono suddivise in quattro categorie:

- *Pagine principali*  
Composto dalla “home”, dalla “programmazione” e dalla “collezione”.  
Esse prevedono una *navbar* nella porzione superiore dello schermo in cui sono presenti:
  - Titolo del progetto, se cliccato riporta alla “home”
  - Tre pulsanti di navigazione, in ordine “Home” – “Programmazione” – “Collezione”
  - Immagine del profilo cliccabile, in tal caso verrà aperta una piccola finestra con diverse opzioni (visualizzazione profilo, visualizzazione acquisti, logout, pannello amministratori se l'utente è un amministratore)
  - Pulsanti per accedere e registrarsi, presenti solo se l'utente è un *Guest*

In “Home” è presente un carosello scorrevole contenente dei bottoni che portano ad altre parti del sito.

In “Programmazione” sono presenti dei bottoni per scegliere il giorno di cui

visualizzare la programmazione (settimanale). Se vi sono proiezioni verranno visualizzate delle *card* (ogni card è una proiezione diversa) contenenti la copertina del film, le corrispettive informazioni e un pulsante acquista (nel caso di un *Guest* esegue una redirectione alla pagina di login).

In “Collezione” è possibile ricercare tra tutte le proiezioni esistenti tramite dei filtri. I risultati verranno mostrati in forma tabellare.

- Operazioni utente

Composto dalle pagine utilizzate per accedere, loggarsi ed effettuare il logout. Esse contengono i relativi *forms*, generati tramite *crispy forms*, oltre ai bottoni di conferma ed annullamento. Inoltre, è presente la pagina per l'acquisto di una proiezione, contenente diversi campi da compilare per confermare il pagamento.

- Profilo utente

Composto dalle pagine utilizzate per visualizzare e modificare il proprio profilo. Per la modifica sono stati utilizzati i *crispy forms*, mentre per la visualizzazione è stato creato un semplice template in cui vengono mostrati i dati attuali dell'utente. È inoltre presente un'altra pagina utilizzata per mostrare i vari acquisti effettuati (ordinati in base alla data ed ora della proiezione acquistata).

- Operazioni amministratore

Composto dalle pagine utilizzate dagli amministratori per accedere al loro pannello, aggiungere/modificare/rimuovere film, sale del cinema e proiezioni. Per la modifica e la rimozione vengono mostrati tutti gli oggetti (film, sala o proiezione in base a ciò che si vuole fare) filtrabili tramite dei campi di ricerca, per ogni oggetto vi sono dei bottoni per scegliere se modificarlo o rimuoverlo.

I *forms* per l'aggiunta e la modifica di un oggetto vengono generati tramite *crispy forms*, la rimozione invece reindirige ad una pagina di conferma.

## Design

Nel progetto in esame, è stata adottata una metodologia che combina l'uso di componenti predefiniti di Bootstrap 5 con la creazione di componenti personalizzati. Questa scelta ha consentito di sfruttare la robustezza e la flessibilità offerte da Bootstrap 5, garantendo al contempo che ogni elemento visivo fosse adeguatamente adattato alle specifiche esigenze del progetto.

Dal punto di vista grafico, sia i componenti di Bootstrap che quelli personalizzati sono stati modificati per adeguarsi all'estetica complessiva dell'app. Questa personalizzazione ha incluso la modifica di stili CSS e l'implementazione di script JavaScript per migliorare l'interattività e la responsività dell'interfaccia utente.

## Test

Era richiesta la produzione di test che verificassero la funzionalità di parti del software dell'applicazione. Per implementare questo punto si è ricorso al framework Django di testing basato su **unittest**.

All'interno del progetto sono state testate due funzionalità, più precisamente una funzione ed una view (rispettivamente *test.py* all'interno delle app *screenings* e *bookings*)

### Testing funzione

La funzione presa in analisi è denominata `is_overlapping()` e si trova all'interno del file *forms.py* dell'app *screenings*. Essa viene chiamata ogni volta che si tenta di aggiungere o aggiornare una proiezione e ha lo scopo di controllare che non vi siano sovrapposizioni. Quest'ultime possono verificarsi nel caso in cui nello stesso giorno e sala vi siano due proiezioni, *s1* e *s2*, che hanno un orario di inizio coincidente:

- orario di inizio di *s1* è prima dell'orario di fine della nuova proiezione *s2*
- orario di fine di *s1* è dopo l'orario di inizio della nuova proiezione *s2*
- *s2* completamente inclusa all'interno della proiezione esistente *s1* o *s2* coincide perfettamente con *s1*
- modifica dell'orario di inizio di una proiezione esistente crea "overlap" con un'altra proiezione esistente

Controllando le precedenti condizioni si ha un criterio che assicura che qualsiasi sovrapposizione venga rilevata.

Un controllo simile, non testato però tramite *unit testing*, è stato implementato anche all'interno dell'app *movies*. Esso ha lo scopo di evitare che la modifica della durata di un film esistente vada a creare problemi di "overlapping" tra le proiezioni esistenti che proiettano il film considerato.

### Testing vista

La vista presa in analisi è denominata `PurchaseTicketsView()` e si trova all'interno del file *views.py* dell'app *bookings*. Questa vista ha lo scopo di gestire l'acquisto dei biglietti per una certa proiezione mostrando i relativi template e facendo le dovute operazioni.

Viene richiamata quando l'utente clicca il bottone "Acquista" all'interno delle *card* che rappresentano le proiezioni, oppure all'interno della tabella che mostra i risultati a seguito della ricerca di proiezioni specifiche nella *collection* (`localhost:8000/collection/`).


I test effettuati si concentrano soprattutto sui codici di risposta *HTTP*, sui corretti “render” e “redirect” dei template, sul rilevamento di errori o problemi nel *form* d’acquisto e sul controllo che una proiezione “sold out” non possa essere acquistata.

## Risultati finali




**Figura 6:** Home

**Figura 7:** Acquisto proiezione



**MovieMore**

[Home](#)
[Programmazione](#)
[Collezione](#)



[03-07-2024](#)
[04-07-2024](#)
[05-07-2024](#)
[06-07-2024](#)
[07-07-2024](#)
[08-07-2024](#)
[09-07-2024](#)

### Programmazione giornaliera del 09-07-2024



**The Dark Knight**

Quando la minaccia conosciuta come il Joker emerge dal suo misterioso passato, semina il caos e l'anarchia tra la gente di Gotham.

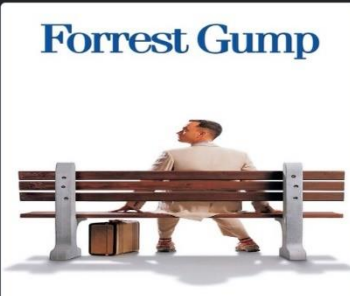
**Cast:** Christian Bale, Heath Ledger, Aaron Eckhart

**Durata:** 152 minuti

**Regista:** Christopher Nolan

**Genere:** Azione

[Acquista](#) 09-07-2024 22:30 **Sala:** Sala 8



**Forrest Gump**

Le presidenze di Kennedy e Johnson, la guerra del Vietnam, lo scandalo Watergate e altri eventi storici si svolgono dalla prospettiva di un uomo dell'Alabama con un QI di 75, il cui unico desiderio è riuniti con il suo amore d'infanzia.


**Cast:** Tom Hanks, Robin Wright, Gary Sinise

**Durata:** 142 minuti

**Regista:** Robert Zemeckis

**Genere:** Drammatico

[Acquista](#) 09-07-2024 19:00 **Sala:** Sala 5



**La Lista di Schindler**

Nella Polonia occupata dai tedeschi durante la seconda guerra mondiale, l'industriale Oskar Schindler gradualmente si preoccupa per la sua forza lavoro ebraica dopo averne testimoniato la persecuzione da parte dei nazisti.

**Cast:** Liam Neeson, Ben Kingsley, Ralph Fiennes

**Durata:** 195 minuti

**Regista:** Steven Spielberg

**Genere:** Biografia

[Acquista](#) 09-07-2024 22:15 **Sala:** Sala 9

[1](#)
[2](#)
[»](#)
[»»](#)

**Figura 8: Programmazione**

**Pannello amministratori**

[Torna al sito](#)
[Logout](#)

**Film**

Clicca qui per aggiungere, modificare o eliminare un film

[> Aggiungi film](#)  
[> Modifica / Elimina film](#)

**Sale**

Clicca qui per aggiungere, modificare o eliminare una sala

[> Aggiungi sala](#)  
[> Modifica / Elimina sala](#)


**Proiezioni**

Clicca qui per aggiungere, modificare o eliminare una proiezione


[> Aggiungi proiezione](#)  
[> Modifica / Elimina proiezione](#)

**Figura 9: Pannello amministratori**




**MovieMore**

[Home](#)
[Programmazione](#)
[Collezione](#)



## Ricerca proiezioni

Titolo

Data iniziale

Genere

Data finale

Ordina per

Cerca

### Risultati della ricerca

Titolo	Genere	Data proiezione	Ora proiezione	Azione
Matrix	Fantascienza	05-07-2024	17:30	<a href="#">Acquista</a>
Interstellar	Fantascienza	06-07-2024	18:00	<a href="#">Acquista</a>
Inception	Fantascienza	08-07-2024	18:15	<a href="#">Acquista</a>
Matrix	Fantascienza	08-07-2024	18:30	<a href="#">Acquista</a>
Matrix	Fantascienza	07-07-2024	19:15	<a href="#">Acquista</a>
Inception	Fantascienza	08-07-2024	21:15	<a href="#">Acquista</a>
Inception	Fantascienza	12-07-2024	22:00	<a href="#">Acquista</a>
Interstellar	Fantascienza	03-07-2024	23:30	<a href="#">Acquista</a>

1
2
3
4
5

Figura 10: Collezione

### Elenco delle Proiezioni

Titolo: 
Genere: 
Data di inizio proiezione: 
Data di fine proiezione:

Ordina per:

Filtro
Indietro

Film	Genere	Sala	Data e Ora	Prezzo	Azioni
Inception	Fantascienza	Sala 10	July 12, 2024, 10 p.m.	1.64	<a href="#">Modifica</a> <a href="#">Elimina</a>
Inception	Fantascienza	Sala 7	July 8, 2024, 9:15 p.m.	1.86	<a href="#">Modifica</a> <a href="#">Elimina</a>
Inception	Fantascienza	Sala 7	July 19, 2024, 11 p.m.	5.21	<a href="#">Modifica</a> <a href="#">Elimina</a>
Inception	Fantascienza	Sala 9	July 8, 2024, 6:15 p.m.	3.53	<a href="#">Modifica</a> <a href="#">Elimina</a>
Inception	Fantascienza	Sala 8	July 23, 2024, 6 p.m.	2.5	<a href="#">Modifica</a> <a href="#">Elimina</a>
The Dark Knight	Azione	Sala 7	July 10, 2024, 7 p.m.	7.21	<a href="#">Modifica</a> <a href="#">Elimina</a>
The Dark Knight	Azione	Sala 3	Aug. 1, 2024, 7:30 p.m.	6.99	<a href="#">Modifica</a> <a href="#">Elimina</a>
The Dark Knight	Azione	Sala 3	July 15, 2024, 5:30 p.m.	7.62	<a href="#">Modifica</a> <a href="#">Elimina</a>
The Dark Knight	Azione	Sala 8	July 9, 2024, 10:30 p.m.	1.1	<a href="#">Modifica</a> <a href="#">Elimina</a>
The Dark Knight	Azione	Sala 6	July 5, 2024, 9:15 p.m.	3.17	<a href="#">Modifica</a> <a href="#">Elimina</a>

1
2
3
4
5

Figura 11: Visualizzazione proiezioni



## Problemi riscontrati

L'unico problema degno di nota è relativo alle immagini profilo ed ai poster dei film (ma anche alle altre risorse statiche). Sostanzialmente, dopo aver caricato un'immagine personalizzata tramite il *file chooser*, un'eventuale nuova modifica mostra nel form un campo del tipo:

Currently: <percorso dell'immagine locale>

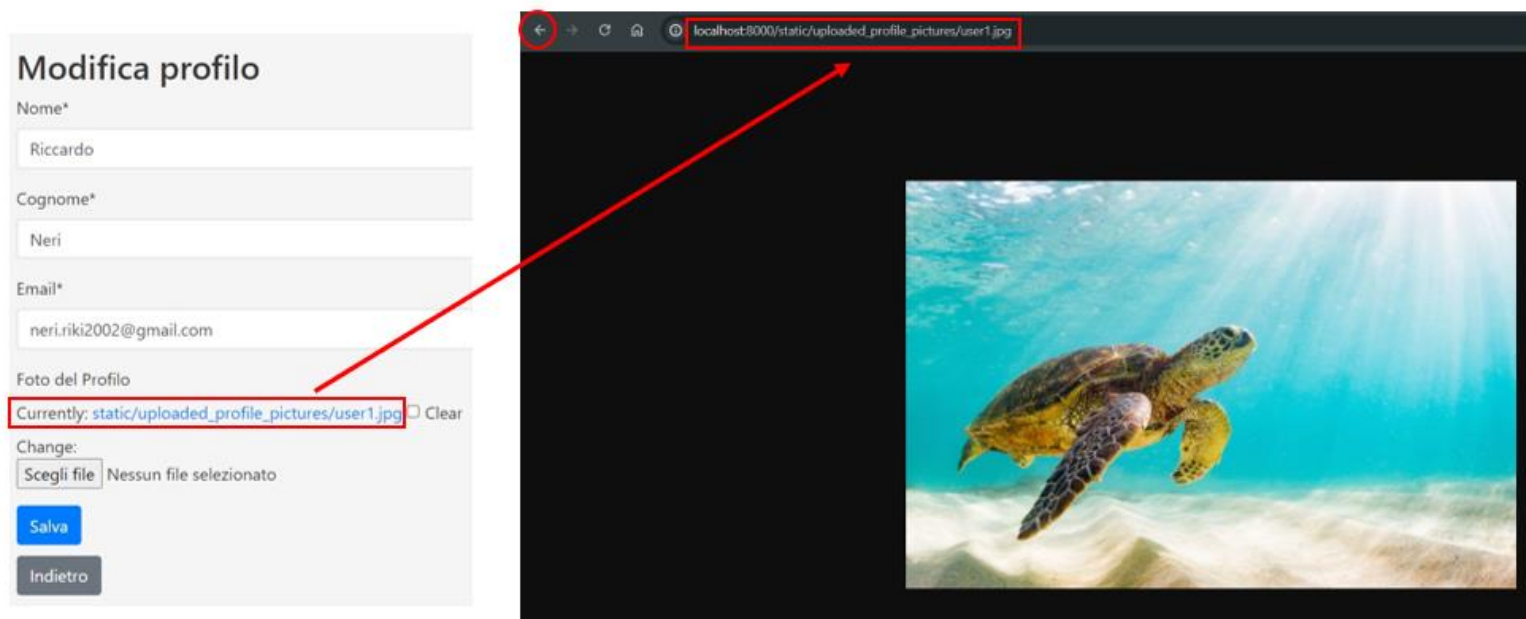
Il problema risiede nel fatto che il percorso (link) è cliccabile, in tal caso si viene indirizzati alla risorsa locale e gli unici modi per tornare al sito sono:

- inserire un url che porta ad una pagina del sito web
- cliccare la “freccia indietro” del browser

Dalle direttive del progetto era esplicitamente citato il fatto che la navigazione tra le varie pagine doveva essere intuitiva e che, soprattutto, non si potesse usare il metodo del secondo punto.

Nonostante numerosi tentativi non è stato possibile implementare una soluzione congrua per rendere il percorso NON cliccabile oppure eliminare direttamente il campo “Currently”. Tale form, infatti, viene direttamente generato da *crispy forms* e non è stato trovato un metodo idoneo per personalizzarlo nei modi appena descritti.

Di seguito vi sono le figure che rappresentano visivamente ciò che è stato appena descritto.



**Figura 12:** Problema risorse statiche

Si può quindi notare che le risorse statiche vengono servite direttamente da Django poiché si è in fase di sviluppo (DEBUG = True). Perciò, se si conoscono gli urls di tali risorse, sarà possibile accedervi (nonostante non siano dati importanti o critici per la sicurezza).