



Università  
Ca' Foscari  
Venezia

# Cineplex

Progetto di Basi di Dati

Anno Accademico 2019/2020

## **Gruppo**

Paolo Concolato, 876700  
Francesco Angiolini, 875339  
Filippo Cigana, 876887

# Indice

Introduzione	3
Funzionalità	4
Struttura file	7
Base di dati	11
Query interessanti	16
Scelte progettuali effettuate	17
Eventuali altre informazioni utili	18

# Introduzione

Cineplex è un'applicazione, principalmente scritta in Python, che utilizza il framework Flask ed SQLAlchemy per interfacciarsi con il DBMS, con la scelta di utilizzare PostgreSQL, in quanto più familiare a noi.

La parte di front-end è responsive, chiara e funzionale, creata utilizzando HTML, CSS e Javascript. Inoltre, è stato utilizzato il template Jinja, il più utilizzato con Python ed ottimo per applicazioni web che utilizzano Flask.

Lo scopo del documento è quello di descrivere il funzionamento di Cineplex parlando delle query effettuate, della struttura dell'applicazione web, della base di dati progettata e di ciò che Cineplex stessa permette di fare sia ai gestori che ai clienti.

Il tema selezionato tra quelli proposti è quindi il seguente:

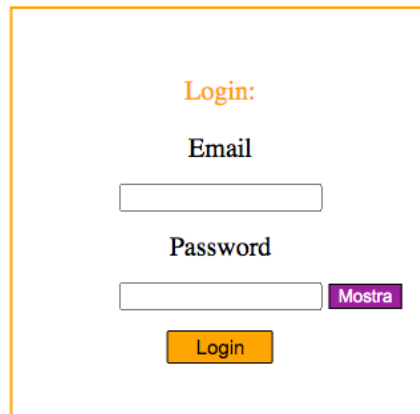
“Cinema: gli utenti possono avere un ruolo fra Cliente e Gestore. I Clienti possono cercare film in base ai loro gusti personali, comprare i biglietti e selezionare un posto a sedere, mentre i Gestori hanno il compito di amministrare l'elenco dei film disponibili. I Gestori hanno anche interesse a collezionare statistiche a fini commerciali.”

# Funzionalità

Cineplex è un'applicazione pensata per gestire un cinema, partendo dalla creazione di proiezioni per arrivare all'acquisto di biglietti, passando per la visualizzazione di statistiche relative ai guadagni.

Gli utilizzatori possono avere il ruolo di gestori o di clienti.

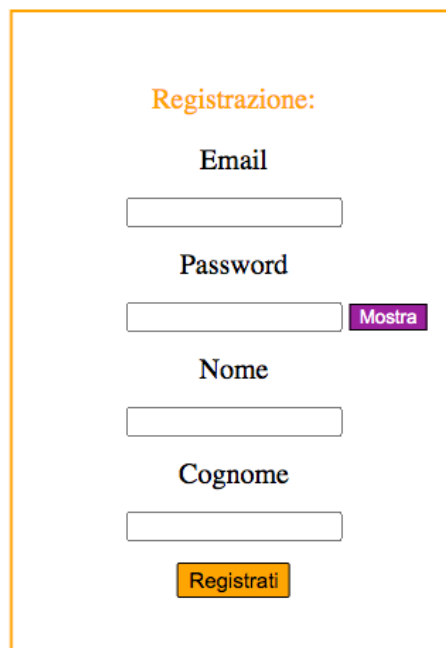
L'autenticazione avviene attraverso una pagina di login nella quale è richiesto il proprio indirizzo mail e la propria password per accedere all'area personale, prima di effettuare un acquisto.



The login form is enclosed in an orange border. It features the title 'Login:' in orange text. Below it are two input fields: 'Email' and 'Password'. The 'Password' field has a purple 'Mostra' button to its right. At the bottom is an orange 'Login' button.

Fig.1 Form di login

Se non si è già registrati, è possibile creare un account in modo semplice attraverso la pagina di signup: viene richiesto di inserire nome, cognome, un indirizzo mail (univoco) e una password. Gli utenti creati in questo modo acquisiscono il ruolo di tipo cliente di default. Potranno essere promossi al ruolo di tipo gestore solo tramite l'azione di un utente che detiene già questo privilegio.



The registration form is enclosed in an orange border. It features the title 'Registrazione:' in orange text. Below it are four input fields: 'Email', 'Password', 'Nome', and 'Cognome'. The 'Password' field has a purple 'Mostra' button to its right. At the bottom is an orange 'Registrati' button.

Fig.2 Form di registrazione

## Funzionalità comuni

Un utente non registrato può solamente vedere i film presenti in catalogo, effettuando eventualmente una ricerca per genere e tra i registi messi a scelta.

Una volta effettuato il login, che sia gestore o cliente, l'utente loggato ha inoltre la possibilità di vedere se ci sono o meno spettacoli in programma per il film da lui selezionato.

Per gli spettacoli presentati, vi è la possibilità di scegliere la fila e il posto a sedere tra quelli ancora disponibili, per poi procedere all'acquisto.

Nella propria area personale, oltre a poter modificare i propri dati, ad eccezione della mail che è univoca, si possono vedere i biglietti acquistati e le relative informazioni sul film.

### Area Privata:

Email

Password

Mostra

Nome

Cognome

Modifica

### Biglietti acquistati:

- Spettacolo: 13/09/20 21:00 -- Sala: 1 -- Fila: 1 -- Posto: 5 -- Film: Avatar

Fig.3 Area Privata

## Funzionalità Gestore

Le funzioni del cliente sono limitate a quelle sopra citate, mentre il gestore ha anche le seguenti.

Lista utenti con la possibilità di:

- eliminare gli utenti presenti, a condizione che non abbiano comprato biglietti o che i biglietti acquistati siano di proiezioni passate;
- promuovere a gestore un cliente;
- rendere cliente un gestore.

Per i film:

- Aggiungere un film;
- Eliminare un film, a condizione che non vi siano biglietti acquistati in sue proiezioni;
- Aggiungere una proiezione, rispettando la sovrapposizione di data, orario e sala;
- Rimuovere uno spettacolo a condizione che non vi siano biglietti acquistati per quella proiezione.

Statistiche:

- Report incassi suddivisi per genere, film e regista; dal più redditizio al meno.

Film	Incassi	N. Biglietti
Django Unchained	35.00 €	7
Avatar	25.00 €	5
Quei bravi ragazzi	15.00 €	3
The Irishman	15.00 €	3

Regista	Incassi	N. Biglietti
Quentin Tarantino	35.00 €	7
Martin Scorsese	30.00 €	6
James Cameron	25.00 €	5

Genere	Incassi	N. Biglietti
Western	35.00 €	7
Drammatico	30.00 €	6
Fantascienza	25.00 €	5

Fig.4 Report statistiche

# Struttura File

Il progetto è composto da due file principali e due cartelle.

Nella cartella *static* troviamo le sottocartelle *css* e *img* che contengono rispettivamente i file *css* per la parte estetica e le immagini.

Nella cartella *templates* ci sono invece tutti i file *html* che vengono invocati dalle varie route presenti nel file principale *app.py*.

## Html

File *html* con rispettive caratteristiche:

- *base*, template del sito contenente l'header standard di ogni pagina con i bottoni per l'accesso all'area privata, al login, al signup e di logout. Contiene anche la funzione in js richiamata da più file per la visualizzazione del campo password nei form;

The screenshot shows the homepage of a website named 'Cineplex'. At the top is a dark navigation bar with a 'Home' button on the left, the 'Cineplex' logo in the center, and 'Area Gestione' and 'Logout' buttons on the right. Below the navigation bar is a search section with two dropdown menus for 'Genere' (set to 'Tutti') and 'Regista' (set to 'Tutti'), followed by 'Cerca' and 'Reset' buttons. The main content area displays a list of movies, each with its author, genre, duration, title, and a 'Compra' button. The movies listed are: Alice in Wonderland (Tim Burton, Fantasy, 108 min), Avatar (James Cameron, Fantascienza, 178 min), Bastardi senza gloria (Quentin Tarantino, Guerra, 153 min), Bohemian Rhapsody (Bryan Singer, Biografico, 133 min), Django Unchained (Quentin Tarantino, Western, 165 min), E. T. (Steven Spielberg, Fantascienza, 121 min), Gran Torino (Clint Eastwood, Thriller, 120 min), and Il pianista (Roman Polański, Guerra, 150 min).

Autore	Genere	Durata	Titolo	Compra
Tim Burton	Fantasy	108 minuti	Alice in Wonderland	Compra
James Cameron	Fantascienza	178 minuti	Avatar	Compra
Quentin Tarantino	Guerra	153 minuti	Bastardi senza gloria	Compra
Bryan Singer	Biografico	133 minuti	Bohemian Rhapsody	Compra
Quentin Tarantino	Western	165 minuti	Django Unchained	Compra
Steven Spielberg	Fantascienza	121 minuti	E. T.	Compra
Clint Eastwood	Thriller	120 minuti	Gran Torino	Compra
Roman Polański	Guerra	150 minuti	Il pianista	Compra

Fig.5 Homepage

- *index*, homepage (fig.5) del sito con lista film e accesso all'acquisto dei biglietti;
- *user*, pagina contenente il form di login;
- *registra*, pagina contenente il form di registrazione. In caso di successo, redirect alla pagina di login;
- *private*, area privata dell'utente con form per modifica dati personali e lista degli eventuali biglietti acquistati;

- *gestione*, presenta tutte le funzionalità dell'utente con ruolo gestore;
- *compra*, (fig.6) mostra per il film selezionato le proiezioni disponibili;

### Spettacoli del film: Avatar

11/09/20

19:30 - Sala 1

13/09/20

17:00 - Sala 1

17:00 - Sala 2

21:00 - Sala 1

Fig.6 compra.html

- *posto*, (fig.7) scelta del posto da acquistare e pagamento;

Indietro

---- Schermo ----

Fila 1 | ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ |

Fila 2 | ☐ ☒ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ |

Totale: 5,00 €

Acquista

Fig.7 posto.html

- *pagato*, riepilogo e conferma di pagamento avvenuto;
- *confirmedOperation*, conferma generica di operazione effettuata andata a buon fine;
- *insertError*, warning generico di gestione errori nell'operazione effettuata (solitamente parametri in input sbagliati passati alle query).



## app.py

File *app.py* è il cuore del programma:

- import dei moduli;
- dichiarazione dell'app
- creazione e connessione al database
- configurazione flask-login e della classe User per l'autenticazione
- handler errori di navigazione
- route e relative query

## **Route**

'/'

Homepage

('/cerca'

Metodo per cerca film

('/user'

Pagina di login, se non autenticato

('/acquista'

Lista proiezioni del film, solo se autenticato

('/login'

Metodo per gestione login

('/logout'

Metodo logout

('/private'

Pagina area gestore con relative query, solo se autorizzato

('/signup'

Pagina di registrazione, solo se autenticato

('/registrato'

Metodo per registrare nuovo utente

('/eliminaFilm'

Metodo per eliminare film, solo se autorizzato

('/eliminaSpettacolo'

Metodo per eliminare spettacolo, solo se autorizzato

('/filmInserito'

Metodo per inserire film, solo se autorizzato

('/spettacoloInserito'

Metodo per aggiungere spettacolo, solo se autorizzato

('/aggUtente'

Metodo per aggiornare dati utente, solo se autorizzato

‘/prenotaPosto’

Pagina visualizzazione posti dello spettacolo, solo se autenticato

‘/acquistaPosto’

Pagina conferma acquisto biglietto, solo se autenticato

‘/gestUtente’

Metodo eliminazione utente, solo se autorizzato

‘/promuoviAdmin/<index>’

Metodo modifica ruolo utente, solo se autorizzato

‘/pp’

Pagina area privata singolo utente, solo se autenticato

# Base di dati

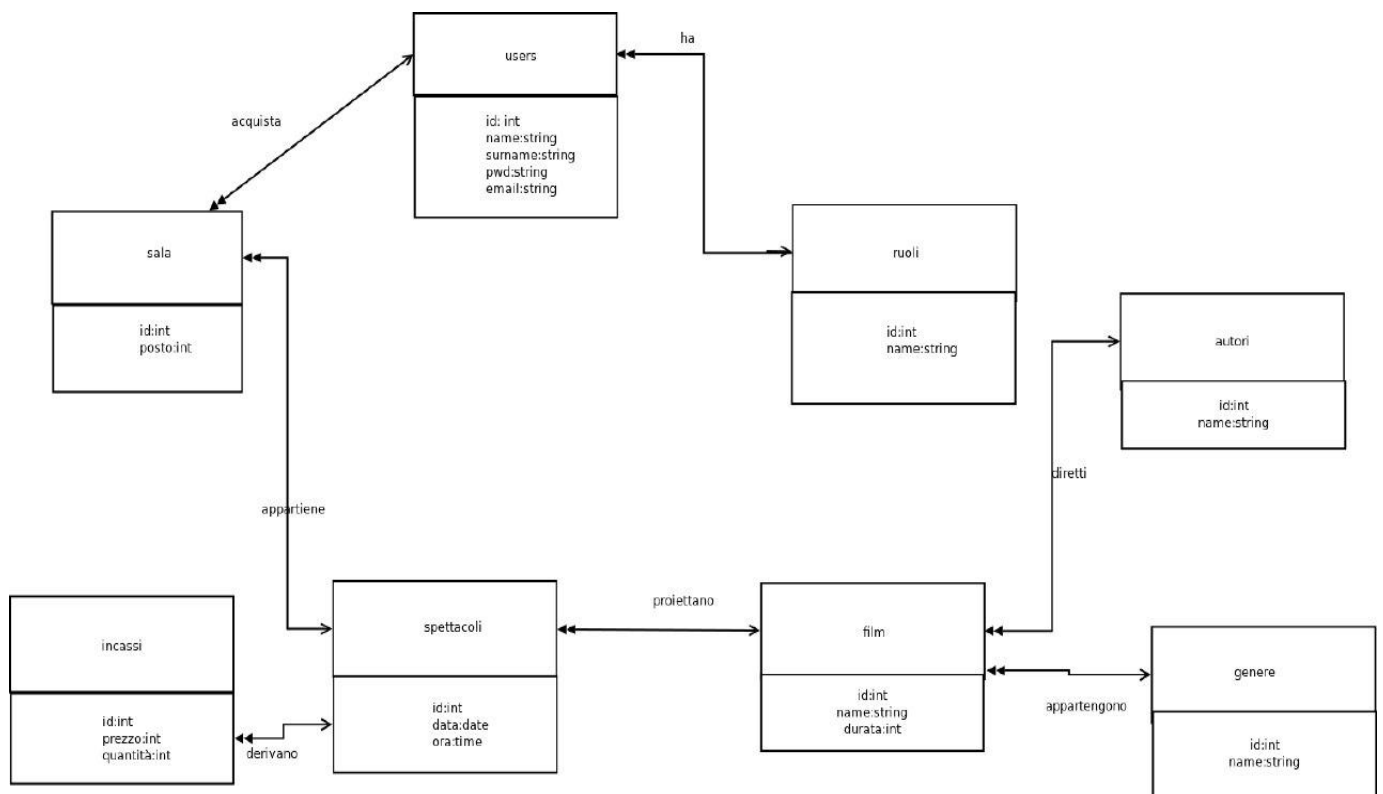


Fig.8 Schema concettuale

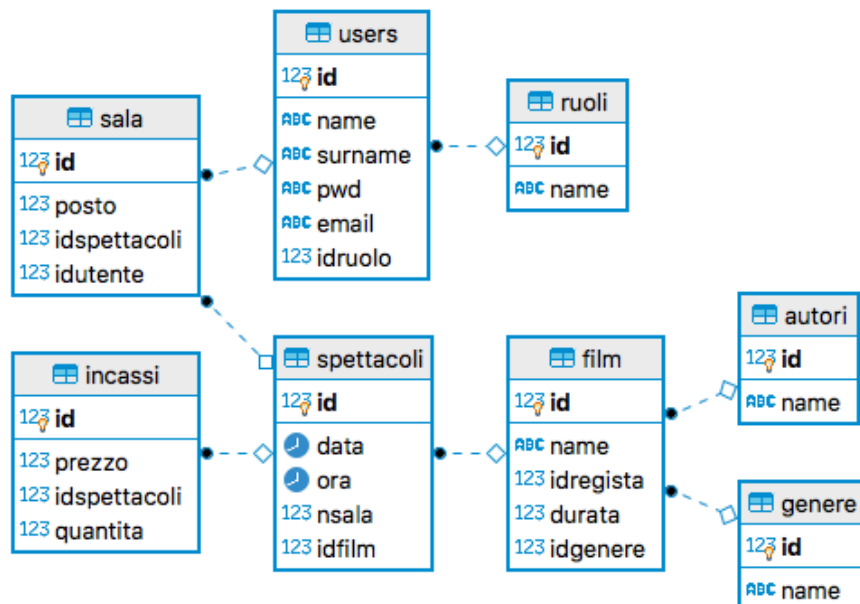


Fig.9 Schema relazionale

## models.py

Possiamo suddividerlo in due sotto parti: inizializzazione del database e popolazione di quest'ultimo.

### **Prima parte: creazione**

(Fig.10) Richiamato da *app.py*, *models.py* principalmente sfrutta SQLAlchemy. La connessione al DB avviene da *app.py* come detto precedentemente, creando l'engine che è il punto di partenza dell'applicazione. L'engine è caratterizzato dall'uso di un dialect per comunicare con il DBMS sottostante, nel nostro caso Postgres, e dal driver specifico Psycopg2. La connessione avviene specificando l'username e la password dell'utente del DBMS che esegue il file. Il database con nome "movie" viene istanziato in *app.py*, solamente se esso non esiste già e successivamente viene inizializzato e popolato da *model.py* per creare le sue relative tabelle che sono:

- users: un utente ha un id(primary key), nome, cognome, email(unico), password e l'id del ruolo(foreign key[ruoli]). Identifica gli utenti registrati.
- film: un film ha un id(primary key), nome, id del regista(foreign key[autori]), durata e l'id del genere(foreign key[genere]). Identifica i film in catalogo.
- ruoli: ogni ruolo ha un id(primary key) e un nome. Rappresenta i ruoli che può assumere un utente.
- spettacoli: ogni spettacolo ha un id(primary key), data di proiezione, ora, numero della sala dove è proiettato e id del film(foreign key[film]). Le proiezioni di un film.
- genere: formato dal suo id(primary key) e dal nome. Rappresenta i generi di film.
- autori: corrispondenti ai registi, hanno un id(primary key) e un nome.
- incassi: rappresenta il botteghino totale di ogni film. Composto da id(primary key), prezzo del biglietto(default a 5€), spettacolo di riferimento(foreign key[spettacoli]) e la quantità, ovvero il numero di biglietti acquistati da quella persona in quel momento.
- sala: l'insieme dei posti acquistati per ogni proiezione. Ha un id(primary key), il numero del posto nella sala, il riferimento allo spettacolo(foreign key[spettacoli]) e l'utente che l'ha acquistato(foreign key[users]).

### **Seconda parte: popolamento**

La seconda parte del file *models.py* inizializza il database per eliminare eventuali doppioni nelle tabelle per poi inserisce un paio di utenti (uno gestore ed uno cliente), i maggiori generi di film, almeno un film per genere con relativo regista, alcune proiezioni e dei posti acquistati con relativi incassi.

L'inizializzazione con successiva creazione dell'engine, antecedente alle operazioni di insert si può vedere in fig.11.

### Osservazioni

Per fornire una maggiore scalabilità è stata creata la tabella ruoli in modo che l'aggiunta di un nuovo ruolo non costringa l'aggiornamento di tutte le righe delle tabelle coinvolte o una modifica onerosa del programma generale. Tutto il database è stato progettato seguendo questa filosofia, garantendo una maggiore flessibilità in caso di modifiche sostanziali.

Per quanto riguarda il parsing degli input passati alle query, si è deciso di operare più a livello software(python) che database(sql) per una questione di familiarità e dimestichezza, a favore anche della sicurezza con l'utilizzo di regex.

Si è utilizzato SQLAlchemy in formato Expression Language per lo più in *models.py* per istanziare le tabelle del database e fare le operazioni di insert più elegantemente.

Mentre è stato preferito Textual SQL per quasi il resto delle query perché avendo la possibilità di scegliere il DBMS sottostante, abbiamo ritenuto più pratico e semplice scrivere le query nel 'classico' formato SQL.

```

# creazione tabelle
users = Table('users', metadata,
    Column('id', Integer, primary_key=True),
    Column('name', String),
    Column('surname', String),
    Column('pwd', String),
    Column('email', String, unique=True),
    Column('idruolo', Integer, ForeignKey('ruoli.id'))
)

film = Table('film', metadata,
    Column('id', Integer, primary_key=True),
    Column('name', String, unique=True),
    Column('idregista', Integer, ForeignKey('autori.id')),
    Column('durata', Integer),
    Column('idgenere', Integer, ForeignKey('genere.id'))
)

ruoli = Table('ruoli', metadata,
    Column('id', Integer, primary_key=True),
    Column('name', String, unique=True)
)

spettacoli = Table('spettacoli', metadata,
    Column('id', Integer, primary_key=True),
    Column('data', Date),
    Column('ora', Time),
    Column('nsala', Integer),
    Column('idfilm', Integer, ForeignKey('film.id'))
)

genere = Table('genere', metadata,
    Column('id', Integer, primary_key=True),
    Column('name', String, unique=True)
)

autori = Table('autori', metadata,
    Column('id', Integer, primary_key=True),
    Column('name', String, unique=True)
)

incassi = Table('incassi', metadata,
    Column('id', Integer, primary_key=True),
    Column('prezzo', Integer, default=5),
    Column('idspettacoli', Integer, ForeignKey('spettacoli.id')),
    Column('quantita', Integer)
)

sala = Table('sala', metadata,
    Column('id', Integer, primary_key=True),
    Column('posto', Integer),
    Column('idspettacoli', Integer, ForeignKey('spettacoli.id')),
    Column('idutente', Integer, ForeignKey('users.id'))
)

```

Fig.10 Creazione tabelle

```

#inizializzo database e creo tabelle
metadata.drop_all(engine)
metadata.create_all(engine)

# popolo il database con le insert
ins = users.insert()
insF = film.insert()
insR = ruoli.insert()
insA = autori.insert()
insG = genere.insert()
insS = spettacoli.insert()
insI = incassi.insert()
insP = sala.insert()
conn = engine.connect()

#insert
conn.execute(insG,[
    {'name': 'Animazione'},
    {'name': 'Avventura'},
    {'name': 'Biografico'},
    {'name': 'Commedia'},
    {'name': 'Documentario'},
    {'name': 'Drammatico'},
    {'name': 'Fantascienza'},
    {'name': 'Fantasy'},
    {'name': 'Guerra'},
    {'name': 'Horror'},
    {'name': 'Thriller'},
    {'name': 'Western'}
])

conn.execute(insA,[
    {'name': 'James Cameron'},
    {'name': 'Quentin Tarantino'},
    {'name': 'Martin Scorsese'},
    {'name': 'Steven Spielberg'},
    {'name': 'Woody Allen'},
    {'name': 'Tim Burton'},
    {'name': 'Roman Polański'},
    {'name': 'Sergio Leone'},
    {'name': 'Christopher Nolan'},
    {'name': 'Clint Eastwood'},
    {'name': 'Paolo Sorrentino'},
    {'name': 'Bryan Singer'},
    {'name': 'Stanley Kubrick'},
    {'name': 'Morgan Matthews'}
])

conn.execute(insR,[
    {'name': 'Gestore'},
    {'name': 'Cliente'}
])

```

Fig.11 Insert

## Query interessanti

```
if(request.form['genere'] != '' and request.form['registra'] != ''):
    s = select([film.c.name.label('film'), film.c.id.label('id'), film.c.durata.label('durata'), genere.c.name.label('genere'), autori.c.name.label('autore')]). \
    where(and_(film.c.idregistra == autori.c.id, genere.c.id == film.c.idgenere, genere.c.name == bindparam('gen'), autori.c.name == bindparam('reg')))
    films = conn.execute(s, gen=request.form['genere'], reg=request.form['registra'])
```

- Ricerca dei film secondo il genere e il regista selezionato dall'utente. Questa query prende in input due parametri esterni, ecco l'utilizzo di bindparam(). Vengono fatte due join tra autori e film e tra genere e autori. Scritta in Expression Language.

```
ticket = conn.execute('SELECT * FROM spettacoli WHERE idfilm = %s and %s <= data group by data, spettacoli.id order by data, ora', request.form["idfilm"], datetime.date.today())
```

- Una volta selezionato il film da guardare, questa query restituisce gli spettacoli che riguardano il film e la cui data di proiezione lo stesso giorno o nei giorni successivi a quello attuale. In questo modo si evitano acquisti di biglietti che riguardano spettacoli già avvenuti. I parametri esterni sono di tipo string e date. Viene fatto il group by perché al momento di stampa del risultato sul lato front-end viene controllato se le date degli spettacoli sono uguali tra le righe successive in modo da evitarne la visualizzazione in modo ridondante. Scritta in Textual SQL.

```
moneyF = conn.execute('SELECT sum(prezzo*quantita) as ticket, name, sum(quantita) as n \
from incassi i join spettacoli s on i.idspettacoli = s.id join film on s.idfilm = film.id \
group by name \
order by ticket desc')
```

- Questa query viene usata per il calcolo delle statistiche. Infatti tra i tre report disponibili, la sintassi differisce di poco. Vengono fatte due sum; la prima per il calcolo dell'incasso in € dove avviene al suo interno una pre operazione perché il valore nella tabella è salvato in due colonne, una con il numero di biglietti e l'altra con il prezzo del biglietto. La seconda sum conta invece il totale dei biglietti. Vengono fatte due join perché ci si vuole ricavare anche il nome del film, in questo caso. La differenza con le altre riguarda il nome del genere o del regista. Il group by serve per unire le righe appartenenti, sempre in questo caso, allo stesso film. La scelta di ordinare in modo decrescente è per visualizzare prima il film col maggior incasso.

```
rs = conn.execute('INSERT INTO users(pwd,email,name,surname,idruolo) \
VALUES (%s, %s, %s, %s, %s)', \
[request.form['pwd'], request.form['email'], request.form['name'], request.form['surname'], request.form['idruolo']])
```

- Classico esempio di insert che viene usata ogni volta che aggiungiamo un film, un utente o uno spettacolo. Nella maggior parte dei casi tutti i parametri sono presi dal form presente nella pagina e il parsing, come detto precedentemente, non avviene nella query ma a livello di python prima di passare i dati alla query.

```
ch = conn.execute('SELECT * from sala join spettacoli on sala.idspettacoli = spettacoli.id where idutente = %s and %s <= data ', request.form['idutente'], datetime.date.today())
ch = list(ch)
if len(ch)>0 :
    return render_template('insertError.html')
else:
    conn.execute('DELETE from sala where idutente = %s', request.form['idutente'])
    conn.execute('DELETE FROM users WHERE id = %s and id NOT IN (select idutente from sala ) ', request.form['idutente'])
```

- Dopo aver controllato che non ci siano posti acquistati dall'utente in questione per spettacoli che devono ancora essere proiettati, si passa alla sua eliminazione. Prima si elimina l'utente dall'elenco dei posti, poiché il film per cui aveva comprato il biglietto non è più rilevante in quanto evento passato. Infine si elimina l'utente dalla tabella utenti, in modo che non vi sia più permesso l'accesso all'area privata.



## Scelte progettuali effettuate

### CSS

Utilizzato per rendere più gradevole la visione delle varie pagine dell'applicazione. Abbiamo preferito evitare l'utilizzo di librerie esterne per rendere il programma meno dipendente dall'esterno.

### JavaScript

Utilizzato, insieme a Jinja, per gestire alcune accortezze grafiche. Inoltre, l'utilizzo della libreria jQuery è stata utile per la selezione delle date e alcune dinamiche estetiche più articolate.

### Jinja

Jinja è il template più usato da Python. Permette di creare file in vari formati di markup (ad esempio HTML) e di restituirli all'utente tramite risposta HTTP. Inoltre, permette l'ereditarietà di template utilizzando il tag `{% extends %}` che determina il template genitore. Tra i delimitatori più utili si possono notare `{% istruzione %}` ed `{{ espressioni e variabili }}`, è permesso l'uso di if ma anche di cicli come il for-in. In Cineplex, Jinja è utile per stampare valori passati da Flask controllando che ve ne siano di stampabili ed eventualmente scorrendo tramite un for ciò che è stato ricevuto. La possibilità di usare gli if-else è utile per controllare i valori di condizioni che possono indicare errori di inserimento tramite degli alert.

### Transazioni

Tutte le operazioni SQLAlchemy effettuano un auto-commit di default, cioè le singole istruzioni sono tutte transazioni "banali". Non abbiamo ritenuto opportuno l'uso di altre transazioni poiché assenti casi dove si necessita rollback.

### Vincoli

I vincoli utilizzati sono principalmente Unique e Primary key presenti all'interno del file models.py. Gli attributi id sono indicati come Primary key mentre l'attributo email ed alcuni attributi name sono indicati come Unique per rendere univoci i campi.

## Eventuali altre informazioni utili

### Come eseguire Cineplex da terminale

In `app.py` (a riga 18) sostituire i tre parametri con il proprio username seguito da password (la propria password) se è presente una password e l'indirizzo di host. Infatti, la forma del database URL deve essere la seguente: `dialect+driver://username:password@host:[port]`

In questo modo verrà creato l'engine.

Da terminale, dirigersi nella cartella Cineplex ed eseguire l'applicazione con: `python3 app.py`.

Ovviamente bisogna aver installati tutti i moduli presenti in `app.py`.

Aprire il browser recarsi su <http://127.0.0.1:5000/> nel nostro caso.

L'utente di tipo gestore ha come indirizzo mail `dev@test.it` e come password `dev` mentre l'utente di tipo cliente ha come indirizzo mail `test@test.it` e come password `test` ma è possibile creare altri clienti utilizzando la pagina di registrazione.

Testato su browser Google Chrome e Firefox.