



Università  
Ca' Foscari  
Venezia

# *CinemApp*

Progetto di Basi di Dati

Anno Accademico 2019/2020

## **Gruppo**

Francesco Camporese, 875287

Thomas Rossetto, 874312

Angela Pezzato, 875696

# Indice

Pag. 3: Introduzione

Pag. 4: Cos'è CinemApp? Quali sono le sue funzionalità?

Pag. 9: La base di dati

Pag. 13: Query interessanti

Pag. 15: Scelte progettuali effettuate

Pag. 17: Eventuali altre informazioni utili

# Introduzione

CinemApp è un'applicazione, principalmente scritta in Python, che utilizza il framework Flask ed SQLAlchemy per interfacciarsi con il DBMS, in particolare la scelta è ricaduta su MySQL.

La parte di front-end è responsive, chiara e funzionale. È stata creata con HTML, Javascript e Bootstrap: il famoso framework per lo sviluppo di interfacce. Inoltre, è stato utilizzato il motore di 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 CinemApp parlando delle query effettuate, della struttura dell'applicazione web, della base di dati progettata e di ciò che CinemApp stessa permette di fare sia ai gestori che ai clienti.

Il tema selezionato tra quelli proposti è quindi il seguente:

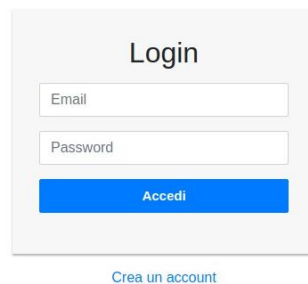
“Cinema: gli utenti possono avere un ruolo fra Cliente e Gestore. I Clienti possono cercare film in base ai loro gusti personali, comprare i biglietti e selezionare un posto a sedere, mentre i Gestori hanno il compito di amministrare l'elenco dei film disponibili. I Gestori hanno anche interesse a collezionare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

# 1: Cos'è CinemApp? Quali sono le sue funzionalità?

CinemApp 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.

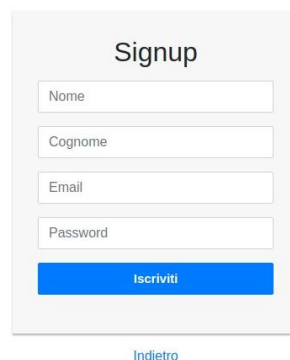
Gli utilizzatori possono essere dei gestori o dei clienti.

L'identificazione avviene attraverso un form di login nel quale è necessario inserire il proprio indirizzo mail e la propria password per accedere all'area personale.



The login form is titled "Login" and is contained within a light gray box. It features two input fields: "Email" and "Password". Below these fields is a blue button labeled "Accedi". At the bottom of the box, there is a link that says "Crea un account".

Se non si è iscritti, è possibile creare un account in modo semplice attraverso il form signup: è necessario inserire nome, cognome, un indirizzo mail (univoco, infatti più utenti non possono avere lo stesso) ed una password dalla lunghezza discreta al fine di poter creare un nuovo account di tipo cliente. Gli utenti di tipo gestore non possono essere creati attraverso il form signup per motivi di sicurezza e gli utenti di tipo cliente non possono essere promossi ad utente di tipo gestore.



The signup form is titled "Signup" and is contained within a light gray box. It features four input fields: "Nome", "Cognome", "Email", and "Password". Below these fields is a blue button labeled "Iscriviti". At the bottom of the box, there is a link that says "Indietro".

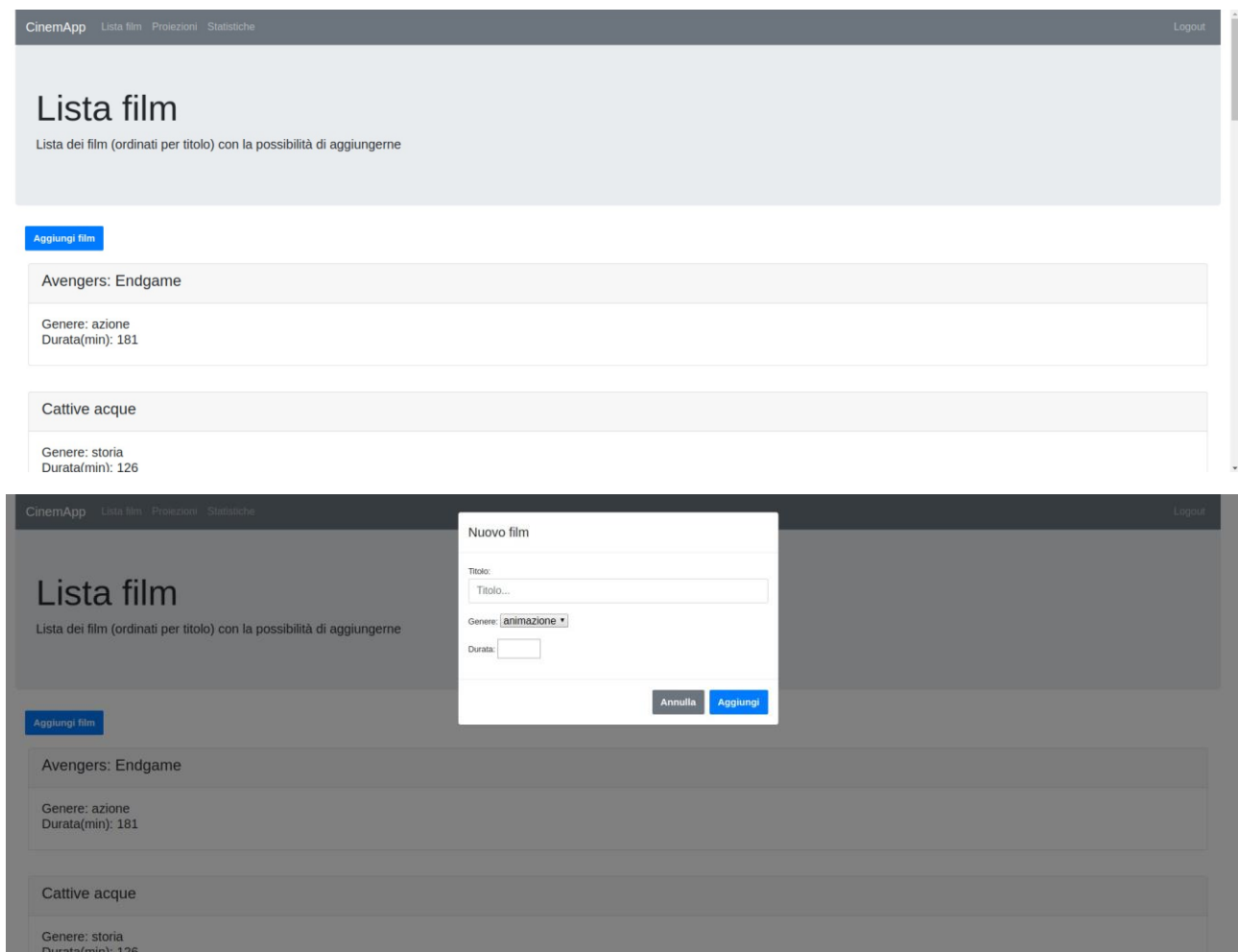
Una volta superata la pagina di login, in base al tipo di utente le features offerte sono differenti:

- **Gestore**
  - Pagina di benvenuto: accessibile appena si supera la pagina di login, oppure cliccando sul logo CinemApp, la pagina di benvenuto saluterà il gestore ed indicherà quali sale sono attualmente occupate da una proiezione. Per ciascuna delle sale occupate verrà indicato il numero che identifica la sala, la data/orario di inizio della proiezione ed il film proiettato



The welcome page for a manager has a dark gray header with the text "CinemApp" on the left and "Logout" on the right. Below the header, there is a large light gray box with the text "Benvenuto Stefano Calzavara!". Below this box, there is a table with two rows. The first row is labeled "Sala 1". The second row contains the text "Proiezione in corso: Avengers: Endgame (iniziata in orario 2020-07-17 17:48:00)".

- Lista film: accessibile dalla navbar (come le seguenti due sezioni), questa pagina mostra una lista dei film presenti nella base di dati e permette di aggiungerne altri. Per ciascun film viene indicato il titolo, il genere e la sua durata in minuti. È possibile aggiungere un film, infatti cliccando sull'apposito bottone apparirà un modal contenente un form dove viene richiesto di indicare il titolo del film (univoco), la durata in minuti ed il genere (selezionabile da una lista di generi già esistenti, ordinati alfabeticamente) del film che verrà aggiunto cliccando sull'apposito bottone di conferma. Aggiunto un film, questo comparirà nella lista di film ordinati alfabeticamente per titolo



- Proiezioni: questa pagina mostra una lista delle proiezioni create e permette di aggiungerne altre. Per ciascuna proiezione viene indicato il titolo del film proiettato, la sala in cui avviene, la data/orario di inizio ed il prezzo di un biglietto della proiezione. È possibile aggiungere una proiezione, infatti cliccando sull'apposito bottone apparirà un modal contenente un form dove viene richiesto di indicare il titolo del film, la sala (selezionabile da una lista di sale già esistenti, ordinate per numero identificativo), la data/orario di inizio della proiezione (di default quella attuale) ed il prezzo in euro di un singolo biglietto della proiezione che verrà aggiunta cliccando sull'apposito bottone di conferma. Aggiunta una proiezione, questa comparirà nella lista di proiezioni ordinate per data/orario di inizio e titolo

CinemApp
Lista film
Proiezioni
Statistiche
Logout

## Proiezioni

Lista delle prossime proiezioni (dalla più vicina in poi) con la possibilità di aggiungerne

Aggiungi proiezione

ciao

Sala: 1  
Inizio: 2020-07-26 18:00:00  
Prezzo di un biglietto: €5

Avengers: Endgame

Sala: 3

Nuova proiezione

Titolo:

Scegli sala:

Inizio:

Prezzo in euro (solo parte intera):

Annulla
Aggiungi

- Statistiche: questa pagina mostra la quantità di incassi totale ed una lista delle 10 proiezioni più recenti (in ordine di data/orario di inizio decrescente) che hanno portato degli incassi. Per ciascuna proiezione viene indicata la data/orario di inizio, la sala, il titolo del film, il numero di biglietti venduti ed il totale dei ricavi che ha portato. Inserendo il titolo di un film, o una parte di titolo (anche contenuta in più film), sulla casella di ricerca, al posto della lista delle 10 proiezioni più recenti che hanno portato degli incassi comparirà una lista di card che indicano per ciascun film il totale degli incassi che ha portato. In questa lista compariranno solo i film che hanno fatto guadagnare più di €0

CinemApp
Lista film
Proiezioni
Statistiche
Logout

## Statistiche

Vedi la quantità di incassi totale (per tutti i film) ed il numero di biglietti venduti con le 10 proiezioni più recenti che hanno portato degli incassi e con le proiezioni dei film contenenti il testo cercato

Cerca

Totale incassi (per tutti i film): €39

2020-07-30 14:00:00

La proiezione in sala 3 con il film Avengers: Endgame ha venduto 3 biglietti con un ricavo totale di €21

2020-07-28 16:30:00

La proiezione in sala 3 con il film Avengers: Endgame ha venduto 1 biglietti con un ricavo totale di €8

CinemApp Lista film Proiezioni Statistiche Logout

## Statistiche

Vedi la quantità di incassi totale (per tutti i film) ed il numero di biglietti venduti con le 10 proiezioni più recenti che hanno portato degli incassi e con le proiezioni dei film contenenti il testo cercato

avengers

Cerca

Totale incassi (per tutti i film): €39

Avengers: Endgame
Ha incassato in totale €29

- Logout: selezionabile dalla navbar, scegliendo questa opzione si tornerà alla schermata di login in modo da permettere di accedere con un altro utente
- **Cliente**
  - Pagina di benvenuto: accessibile appena si supera la pagina di login, oppure cliccando sul logo CinemApp, la pagina di benvenuto saluterà il cliente ed indicherà i soldi presenti sul conto del cliente stesso (inizialmente ciascun cliente ha a disposizione €50 sul conto)

CinemApp Acquistati Biglietteria Logout

## Benvenuto Francesco Camporese!

Hai a disposizione €50

- Acquistati: accessibile dalla navbar (come l'altra sezione), questa pagina mostra una lista dei biglietti acquistati riguardanti le proiezioni che devono ancora iniziare ordinati per data/orario di inizio della proiezione, titolo del film, numero della sala, fila e numero della poltrona. Per ciascun biglietto viene indicato il titolo del film che verrà proiettato, la data/orario di inizio della proiezione, la sala, la fila ed il numero del posto relativi alla poltrona scelta ed il prezzo del biglietto

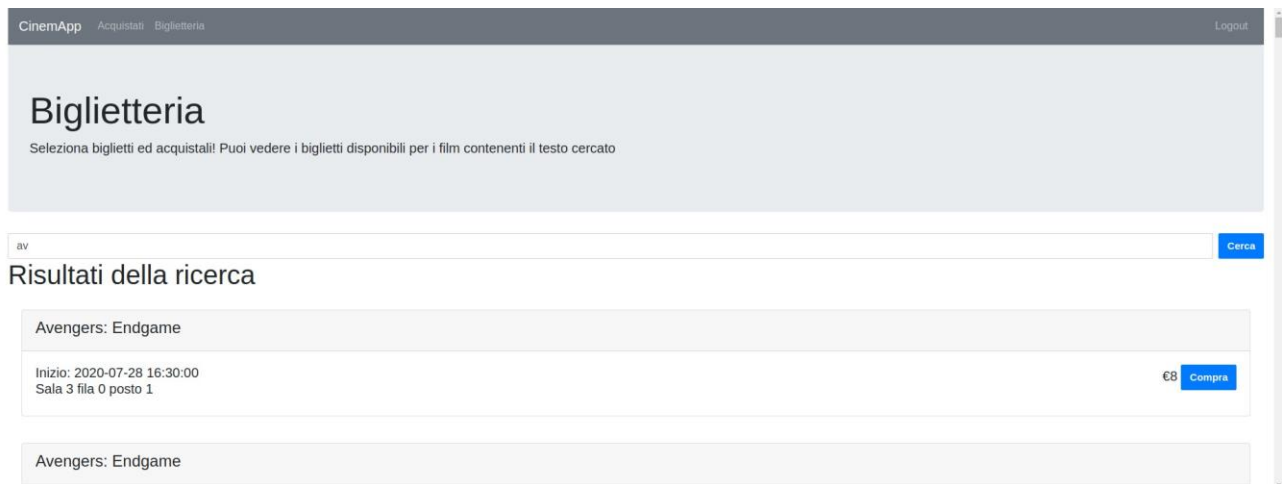
CinemApp Acquistati Biglietteria Logout

## Acquistati

Biglietti acquistati per future proiezioni (dalla più vicina in poi)

ciao	ciao	Avengers: Endgame	Avengers: Endgame	Avengers: Endgame	Avengers: Endgame
2020-07-26 18:00:00 Sala 1 fila 1 posto 6 €5	2020-07-26 18:00:00 Sala 1 fila 4 posto 7 €5	2020-07-28 16:30:00 Sala 3 fila 0 posto 0 €8	2020-07-30 14:00:00 Sala 3 fila 2 posto 3 €7	2020-07-30 14:00:00 Sala 3 fila 3 posto 4 €7	2020-07-30 14:00:00 Sala 3 fila 5 posto 13 €7

- Biglietteria: questa pagina, inserendo il titolo di un film, o una parte di titolo (anche contenuta in più film), sulla casella di ricerca, mostrerà una lista di card che indicano per ciascun biglietto acquistabile il titolo del film, la data/orario di inizio della proiezione, la sala, la fila, il numero del posto relativi alla poltrona relativa al biglietto ed il prezzo del biglietto stesso. La lista è ordinata per data/orario di inizio della proiezione, titolo del film, numero della sala, fila e numero della poltrona. Acquistato un biglietto cliccando sull'apposito bottone, il biglietto comparirà nella lista di biglietti acquistati

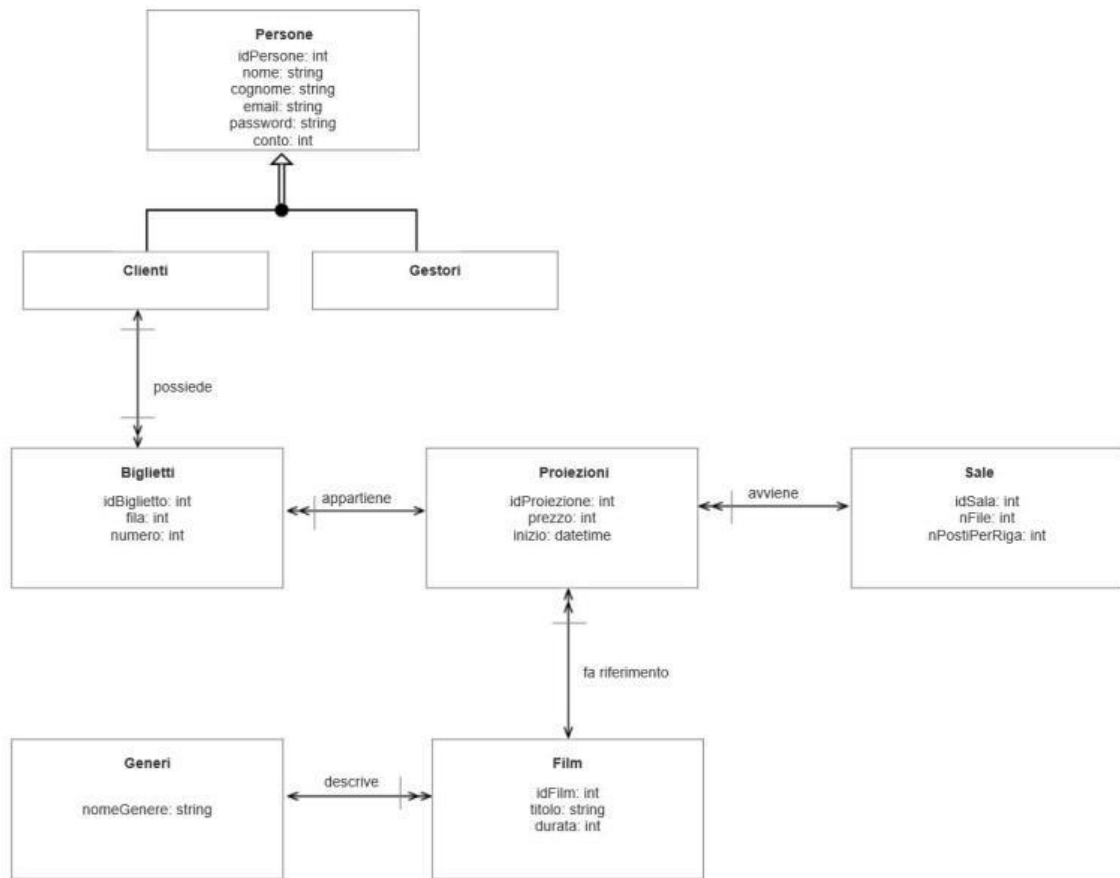


- Logout: selezionabile dalla navbar, scegliendo questa opzione si tornerà alla schermata di login in modo da permettere di accedere con un altro utente.

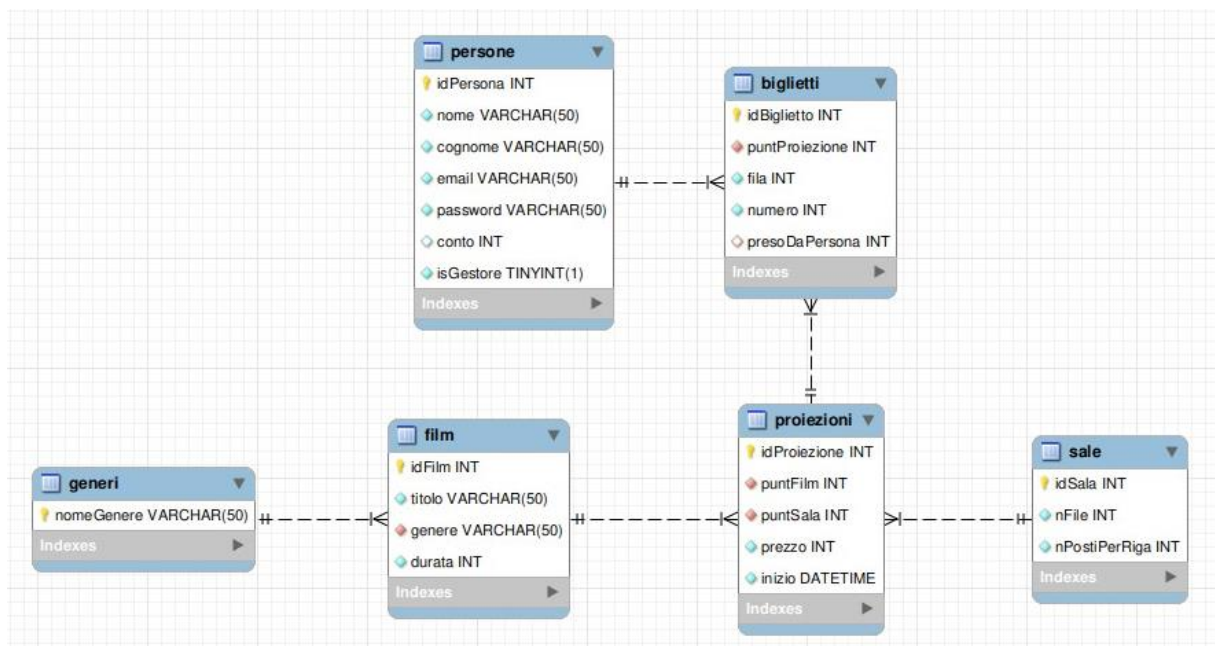


## 2: La base di dati

- A seguire, una rappresentazione dello schema concettuale:



- A seguire, una rappresentazione dello schema relazionale:



- **creaDB.py**

Richiamato da inserimento.py, creaDB.py principalmente sfrutta SQLAlchemy. La connessione dal DB avviene inizialmente da questo file, 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 MySQL, e dal driver specifico, nel nostro caso PyMySQL. La connessione avviene specificando l'username del DBMS che esegue il file e la password. CreaDB.py ha lo scopo di creare il database con nome "cinema" solamente se esso non esiste già (questo controllo ed eventuale creazione avvengono attraverso la sintassi "CREATE DATABASE IF NOT EXISTS cinema") e di utilizzarlo (attraverso la sintassi "USE CINEMA") per creare le sue relative tabelle che sono:

- Persone: una persona è identificata univocamente da un id, ha nome, cognome, un indirizzo mail (univoco), password, un conto (di default €50), un booleano che nel caso in cui sia True indica che l'utente è un gestore, se invece è a False indica che l'utente è un cliente. È stato scelto un booleano in quanto cliente e gestore si differenziano solo per il conto che in gestore non viene realmente utilizzato. Inoltre, quando una persona vuole iscriversi, il booleano viene settato sempre a False per non permettere di creare utenti gestori a piacere dall'applicazione
- Generi: un genere è identificato univocamente dal suo unico attributo, cioè una stringa che rappresenta il nome del genere. L'utilità di questa tabella consiste principalmente nel permettere all'utente di selezionare uno tra i generi predefiniti al momento dell'aggiunta di un film
- Film: un film è identificato univocamente da un id, ha un titolo univoco, punta ad un genere ed ha una durata che viene controllata, infatti deve essere maggiore di 0 minuti. Nonostante il titolo sia univoco, è più comodo usare un id per puntare ad un film
- Sale: una sala è identificata univocamente da un id, ha un numero di file ed un numero di righe che devono essere entrambi maggiori di 0. L'utilità di questi ultimi due consiste nel poter creare un biglietto per ciascuna poltrona al momento della creazione di una proiezione, biglietto che permetterà all'utente di scegliere la poltrona da prenotare
- Proiezioni: una proiezione è identificata univocamente da un id, punta ad un film e ad una sala, ha un prezzo che viene controllato, infatti deve essere maggiore di €0, ed una data/ora di inizio.
- Biglietti: un biglietto è identificato univocamente da un id, punta ad una proiezione, ha fila e numero che indicano la poltrona (e devono essere entrambi maggiori o uguali a 0) e punta eventualmente alla persona che ha acquistato il biglietto.

```

1 import sqlalchemy
2 from sqlalchemy import *
3
4 # creo l'engine passando il dialect+driver mysql+pymysql in quanto il DBMS utilizzato e' MySQL
5 engine = create_engine('mysql+pymysql://root@localhost', echo=True) # nella parte root e' necessario utilizzare il proprio username seguito da :password se e' presente una password
6 # il DB viene si chiama cinema e viene creato solamente se non e' gia' presente attraverso l'istruzione nella riga successiva
7 engine.execute("CREATE DATABASE IF NOT EXISTS cinema")
8 # viene specificato l'uso di cinema come DB
9 engine.execute("USE cinema")
10 # viene creato l'oggetto container Metadata
11 metadata = MetaData()
12
13 # creazione della tabella persone che rappresenta i clienti o gestori
14 persone = Table('persone', metadata,
15                 Column('idPersona', Integer, primary_key=True),
16                 Column('nome', VARCHAR(50), nullable=False),
17                 Column('cognome', VARCHAR(50), nullable=False),
18                 Column('email', VARCHAR(50), nullable=False),
19                 Column('password', VARCHAR(50), nullable=False),
20                 Column('conto', Integer, default=50),
21                 Column('isGestore', Boolean, nullable=False),
22                 UniqueConstraint('email'),
23                 )
24
25 # creazione della tabella generi che rappresenta i possibili generi usabili per i film
26 generi = Table('generi', metadata,
27               Column('nomeGenere', VARCHAR(50), primary_key=True)
28               )
29
30 # creazione della tabella film che contiene tutti i film del cinema
31 film = Table('film', metadata,
32             Column('idFilm', Integer, primary_key=True),
33             Column('titolo', VARCHAR(50), nullable=False),
34             Column('genere', VARCHAR(50), ForeignKey('generi.nomeGenere'), nullable=False),
35             Column('durata', Integer, nullable=False),
36             CheckConstraint('durata > 0', name='checkDurata'),
37             UniqueConstraint('titolo')
38             )
39
40 # creazione della tabella sale che contiene l'insieme delle sale del cinema
41 sale = Table('sale', metadata,
42             Column('idSala', Integer, primary_key=True),
43             Column('nFile', Integer, nullable=False),
44             Column('nPostiPerRiga', Integer, nullable=False),
45             CheckConstraint('nFile > 0', name='checkFile'),
46             CheckConstraint('nPostiPerRiga > 0', name='checkPostiPerRiga')
47             )
48
49 # creazione della tabella proiezioni per l'insieme delle proiezioni del cinema
50 proiezioni = Table('proiezioni', metadata,
51                   Column('idProiezione', Integer, primary_key=True),
52                   Column('puntFilm', Integer, ForeignKey('film.idFilm'), nullable=False),
53                   Column('puntSala', Integer, ForeignKey('sale.idSala'), nullable=False),
54                   Column('prezzo', Integer, nullable=False),
55                   Column('inizio', DateTime, nullable=False),
56                   CheckConstraint('prezzo > 0', name='checkPrezzo')
57                   )
58
59 # creazione della tabella biglietti per rappresentare i biglietti associati alle varie proiezioni
60 biglietti = Table('biglietti', metadata,
61                  Column('idBiglietto', Integer, primary_key=True),
62                  Column('puntProiezione', Integer, ForeignKey('proiezioni.idProiezione'), nullable=False),
63                  Column('fila', Integer, nullable=False),
64                  Column('numero', Integer, nullable=False), #numero del posto nella fila
65                  Column('presoDaPersona', Integer, ForeignKey('persone.idPersona'), nullable=True),
66                  CheckConstraint('fila >= 0', name='checkFila'),
67                  CheckConstraint('numero >= 0', name='checkNumero')
68                  )
69
70 # indice nei confronti dell'attributo puntFilm della tabella proiezioni
71 Index('puntFilm', proiezioni.c.puntFilm)
72 # indice nei confronti dell'attributo idProiezione della tabella proiezioni
73 Index('proiezione', proiezioni.c.idProiezione)
74
75 metadata.create_all(engine) # crea lo schema del DB

```

- **inserimento.py**

Richiamato da application.py, inserimento.py principalmente sfrutta SQLAlchemy per inserire un paio di utenti (uno gestore ed uno cliente), qualche genere, qualche film e delle sale. Nelle tabelle non vuote non verranno fatti gli inserimenti perché si suppone che le tuple che verrebbero inserite siano già esistenti in quanto eliminabili solo da riga di comando MySQL. Essendo questo file eseguito successivamente a creaDB.py, viene creato l'engine (che verrà poi passato ad application.py) in modo simile a come si fa su creaDB.py e viene subito specificato l'uso del DB cinema al fine di (eventualmente) fare gli inserimenti.

```

1 import sqlalchemy
2 from sqlalchemy import *
3 # eseguo il file creaDB
4 import creaDB
5 # importo le varie tabelle create
6 from creaDB import persone,generi,film,sale,proiezioni,biglietti,engine
7
8 engine = create_engine('mysql+pymysql://root@localhost/cinema', echo=True)
9 metadata = MetaData()
10
11 conn = engine.connect()#per erogare comandi tramite conn.
12
13 # conto il numero di persone presenti nel DB
14 s= select([func.count(persone.c.idPersona).label('conto')])
15 result=conn.execute(s)
16 result=result.fetchone()
17 # se non sono presenti persone eseguo degli inserimenti
18 if result['conto'] == 0:
19     personeIns = persone.insert()
20     conn.execute(personeIns,
21     [
22         {'nome': 'Stefano', 'cognome': 'Calzavara', 'email': 'stefano.calzavara@gmail.com', 'password': 'Admin_123', 'isGestore': True},
23         {'nome': 'Francesco', 'cognome': 'Camporese', 'email': 'francesco.camporese99@gmail.com', 'password': 'User_456', 'isGestore': False}
24     ])
25
26 # conto il numero di generi presenti nel DB
27 s= select([func.count(generi.c.nomeGenere).label('conto')])
28 result=conn.execute(s)
29 result=result.fetchone()
30 # se non sono presenti generi eseguo degli inserimenti
31 if result['conto'] == 0:
32     generiIns = generi.insert()
33     conn.execute(generiIns,
34     [
35         {'nomeGenere': 'animazione'},
36         {'nomeGenere': 'azione'},
37         {'nomeGenere': 'drammatico'},
38         {'nomeGenere': 'horror'},
39         {'nomeGenere': 'storia'}
40     ])
41
42 # conto il numero di film presenti nel DB
43 s= select([func.count(film.c.idFilm).label('conto')])
44 result=conn.execute(s)
45 result=result.fetchone()
46 # se non sono presenti film eseguo degli inserimenti
47 if result['conto'] == 0:
48     filmIns = film.insert()
49     conn.execute(filmIns,
50     [
51         {'titolo': 'Avengers: Endgame', 'genere': 'azione', 'durata': '181'},
52         {'titolo': 'Fast and Furious', 'genere': 'azione', 'durata': '106'},
53         {'titolo': 'Joker', 'genere': 'azione', 'durata': '123'},
54         {'titolo': 'Terminator', 'genere': 'azione', 'durata': '107'},
55         {'titolo': 'Inside Out', 'genere': 'animazione', 'durata': '94'},
56         {'titolo': 'Up', 'genere': 'animazione', 'durata': '96'},
57         {'titolo': 'Wall-E', 'genere': 'animazione', 'durata': '98'},
58         {'titolo': 'Philadelphia', 'genere': 'drammatico', 'durata': '121'},
59         {'titolo': 'The Irishman', 'genere': 'drammatico', 'durata': '209'},
60         {'titolo': 'Cattive acque', 'genere': 'storia', 'durata': '126'},
61         {'titolo': 'Il discorso del re', 'genere': 'storia', 'durata': '118'},
62         {'titolo': 'Via col vento', 'genere': 'storia', 'durata': '238'}
63     ])
64
65 # conto il numero di sale presenti nel DB
66 s= select([func.count(sale.c.idSala).label('conto')])
67 result=conn.execute(s)
68 result=result.fetchone()
69 # se non sono presenti sale eseguo degli inserimenti
70 if result['conto'] == 0:
71     saleIns = sale.insert()
72     conn.execute(saleIns,
73     [
74         {'nFile': '5', 'nPostiPerRiga': '10'},
75         {'nFile': '6', 'nPostiPerRiga': '12'},
76         {'nFile': '7', 'nPostiPerRiga': '14'}
77     ])
78
79 conn.close() # chiusura della connessione

```

### 3: Query interessanti

- Nella route listProiezioni, prima di aggiungere una nuova proiezione, bisogna verificare se durante l'orario richiesto la sala scelta è occupata o meno. Per fare questo, bisogna controllare se gli orari dell'eventuale proiezione precedente e dell'eventuale proiezione successiva a quella che si vuole aggiungere possano dare problemi. Infatti, bisogna controllare che la proiezione che si vuole aggiungere non si sovrapponga alle proiezioni precedente e successiva in termini di orario dipendente dall'inizio e dalla durata. In sostanza, la proiezione da aggiungere deve iniziare dopo della fine della proiezione precedente e finire prima dell'inizio della proiezione seguente.

```
s = select([proiezioni.c.inizio, film.c.durata]).select_from(proiezioni.join(film, proiezioni.c.puntFilm == film.c.idFilm)).where(
    and_(proiezioni.c.puntSala == sala, proiezioni.c.inizio <= inizio)).order_by(proiezioni.c.inizio.desc()).limit(1)
```

Per trovare la proiezione precedente seleziono l'inizio delle proiezioni e la durata dei loro rispettivi film, di tutte le proiezioni che hanno un inizio minore o uguale rispetto alla data inserita nel form e che vengono eseguiti sulla stessa sala. Tutte queste proiezioni trovate vengono ordinate in ordine decrescente sulla base della loro data di inizio e prendo solo il primo risultato, in modo che quell'unico risultato sarà la proiezione a livello temporale più prossima.

```
s = select([proiezioni.c.inizio]).where(and_(proiezioni.c.puntSala == sala, proiezioni.c.inizio >= inizio)).\
    order_by(proiezioni.c.inizio.asc()).limit(1)
```

Per trovare la proiezione successiva, seleziono l'inizio di tutte le proiezioni che vengono eseguite nella stessa sala passata nel form, che abbiano una data di inizio maggiore o uguale rispetto a quella inserita, ordinate in ordine crescente e prendo solo il primo risultato ossia la proiezione subito dopo rispetto a quella inserita nel form.

Dai risultati ottenuti riguardo la proiezione precedente e successiva si esegue un controllo in codice Python simile a quello descritto precedentemente per poter fare l'inserimento.

Sia nel caso di una GET che nel caso di una POST viene eseguita la seguente query...

```
proiezioneReg = conn.execute(select([film.c.titolo, proiezioni.c.puntSala, proiezioni.c.inizio, proiezioni.c.prezzo]).\
    select_from(proiezioni.join(film, proiezioni.c.puntFilm == film.c.idFilm)).where(proiezioni.c.inizio > select([func.now()])).\
    order_by(proiezioni.c.inizio, film.c.titolo.asc()))
```

... e vengono selezionate tutte le proiezioni che hanno una data di inizio successiva all'orario attuale e viene restituito il loro titolo, la sala, l'inizio e il costo ordinate in ordine alfanumerico per inizio e titolo.

- Nella route acquistati vengono restituiti tutti i biglietti acquistati dall'utente corrente e che hanno una data di inizio della proiezione a cui fanno riferimento che sia maggiore rispetto all'orario attuale, di questi biglietti vengono restituiti rispettivamente il titolo del film, l'orario della proiezione, la sala, la fila, il numero ed il prezzo del biglietto stesso e vengono ordinati sempre secondo questo ordine.

```
bigliettireg = conn.execute(select([film.c.titolo, proiezioni.c.inizio, proiezioni.c.puntSala, biglietti.c.fila, biglietti.c.numero, proiezioni.c.prezzo]).\
    select_from(proiezioni.join(biglietti, biglietti.c.puntProiezione == proiezioni.c.idProiezione).join(film, proiezioni.c.puntFilm == film.c.idFilm)).\
    where(and_(biglietti.c.presoDaPersona == current.user.idPersona, proiezioni.c.inizio > select([func.now()]))).\
    order_by(proiezioni.c.inizio, film.c.titolo, proiezioni.c.puntSala, biglietti.c.fila, biglietti.c.numero.asc()))
```

- Nella route biglietti\_cerca, attraverso la seguente query...

```
bigliettireg = conn.execute(select([film.c.titolo, proiezioni.c.inizio, proiezioni.c.puntSala, biglietti.c.fila, biglietti.c.numero, proiezioni.c.prezzo, biglietti.c.idBiglietto]).\
    select_from(proiezioni.join(film, film.c.idFilm == proiezioni.c.puntFilm).join(biglietti, biglietti.c.puntProiezione == proiezioni.c.idProiezione)).\
    where(and_(film.c.titolo.like("%" + single_film + "%"), biglietti.c.presoDaPersona == None, proiezioni.c.inizio > select([func.now()]))).\
    order_by(proiezioni.c.inizio, film.c.titolo, proiezioni.c.puntSala, biglietti.c.fila, biglietti.c.numero.asc())) # like non e' case sensitive
```

...vengono selezionati tutti i biglietti che fanno riferimento ad una proiezione non ancora scaduta (ossia il suo orario di inizio è maggiore dell'orario attuale), il cui titolo del film che esse proiettano abbia all'interno i caratteri inseriti nel form cerca. Di questi biglietti viene restituito il titolo del film, l'inizio della proiezione a cui fanno riferimento, la sala in cui viene eseguita, la fila del biglietto, il numero del biglietto stesso, il suo prezzo e l'id. Il risultato restituito viene anche ordinato sulla base

dell'inizio della proiezione, del titolo del film, del numero di sala, della fila e del numero del biglietto in ordine crescente.

- Nella route statistiche viene restituito, pronto per la stampa, il numero di incassi...

```
incassireg = conn.execute(select([func.sum(proiezioni.c.prezzo).label('somma')]).select_from(proiezioni.join(biglietti, biglietti.c.puntProiezione == proiezioni.c.idProiezione)).\
where(biglietti.c.presoDaPersona != None)).fetchone()['somma']
```

...che ritorna il totale del prezzo dei biglietti incassati dal cinema dall'inizio della sua attività.

Poi, nel momento in cui la richiesta nella route è una POST (ossia qualcuno ha cercato un film), si avrà che per ogni film che abbia nel suo titolo parte del testo inserito nel form verrà eseguita la seguente query...

```
result = conn.execute(select([func.sum(proiezioni.c.prezzo).label('sommaIncassi'), film.c.titolo]).\
select_from(proiezioni.join(biglietti, biglietti.c.puntProiezione == proiezioni.c.idProiezione).join(film, film.c.idFilm == proiezioni.c.puntFilm)).\
where(and (proiezioni.c.puntFilm == i['idFilm'], biglietti.c.presoDaPersona != None)).group_by(film.c.titolo))
```

...che restituisce per quel determinato film (in quanto prende l'id del film univoco) la somma di tutti i biglietti venduti che fanno riferimento a quel determinato film, e se il risultato è maggiore di 0 (ossia quel film ha incassato qualcosa allora) allora il film verrà aggiunto ad una lista che verrà stampata.

Nel caso in cui la richiesta consista in una GET, allora verrà eseguita la seguente query...

```
primi_dieci = conn.execute(select([func.count(biglietti.c.idBiglietto).label('nBiglietti'), func.sum(proiezioni.c.prezzo).label('ricavo'), film.c.titolo, proiezioni.c.inizio, proiezioni.c.puntSala]).\
select_from(proiezioni.join(biglietti, biglietti.c.puntProiezione == proiezioni.c.idProiezione).join(film, film.c.idFilm == proiezioni.c.puntFilm)).\
where(biglietti.c.presoDaPersona != None).group_by(proiezioni.c.idProiezione).order_by(proiezioni.c.inizio.desc()).limit(10))
```

...che restituisce il numero di biglietti venduti ed il ricavo totale portato dalle dieci proiezioni più recenti che hanno venduto almeno un biglietto.

## 4: Scelte progettuali effettuate

- **Bootstrap**

Utilizzato per rendere più gradevole la visione delle varie pagine dell'applicazione.

- **Indici**

Nel file creaDB.py vengono definiti due indici al fine di accelerare l'esecuzione delle query che utilizzano questi attributi (anche perché sul DB non si fanno operazioni di eliminazione e nemmeno di modifica se non per l'acquisto di biglietti con conseguente decremento del conto del cliente). I due indici riguardano due attributi della tabella proiezioni, rispettivamente il puntatore al film e l'id della proiezione stessa.

- **JavaScript**

Utilizzato, insieme a Jinja, per mostrare degli alert nel caso in cui l'utente abbia inserito valori non corrispondenti alle aspettative. Inoltre, JS è utile per settare attributi di elementi HTML in modo semplice, come la stampa della data attuale.

- **Jinja**

Jinja è il motore di 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 }}`, quindi è permesso l'uso di if ma anche di cicli come il for-in. In CinemApp, Jinja è molto utile ad esempio per utilizzare la stessa navbar su tutte le schermate di un certo tipo di utente, 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 anche per controllare i valori di certe condizioni che possono indicare tramite degli alert alcuni errori di inserimento.

- **Transazioni**

Nel file application.py sono presenti due transazioni, entrambe con livello di isolamento `SERIALIZABLE`.

- La prima si trova nel momento in cui si cerca di inserire una proiezione in quanto bisogna controllare la proiezione precedente e la proiezione successiva. Durante questi controlli, la tabella non deve essere modificata, in quanto un'aggiunta di un'ulteriore proiezione potrebbe sovrapporre delle proiezioni. Quindi per evitare questo problema bisogna includere il tutto, compreso anche l'inserimento dei biglietti che fanno comunque riferimento alla proiezione, in un'unica transazione
- La seconda transazione invece si trova nella route biglietteria\_acquista nel momento in cui un utente vuole acquistare un biglietto: questa operazione deve essere eseguita in mutua esclusione perché potrebbe portare ad interferenze come due utenti a cui viene assegnato lo stesso posto. Inoltre, se dovesse avvenire un'eccezione c'è la necessità che venga eseguita una rollback in quanto potrebbe essere che il biglietto venga assegnato all'utente ma non venga aggiornato il conto dell'utente stesso.

- **Vincoli**

Nel file creaDB.py vengono definiti dei check su attributo:

- `CheckConstraint('durata > 0', name='checkDurata')` controlla che la durata del film sia maggiore di 0 minuti
- `CheckConstraint('nFile > 0', name='checkFile')` controlla che il numero di file della sala sia maggiore di 0
- `CheckConstraint('nPostiPerRiga > 0', name='checkPostiPerRiga')` controlla che il numero di posti per ciascuna fila della sala sia maggiore di 0. Ogni fila della sala ha lo stesso numero di posti

- `CheckConstraint('prezzo > 0', name='checkPrezzo')` controlla che il prezzo dei biglietti, che verranno creati uno per poltrona della sala scelta per la proiezione, sia maggiore di €0
- `CheckConstraint('fila >= 0', name='checkFila')` controlla che il numero della fila della poltrona relativa al biglietto sia maggiore o uguale a 0. Infatti, il numero di fila varia tra 0 (compreso) e `nFile-1`
- `CheckConstraint('numero >= 0', name='checkNumero')` controlla che il numero della poltrona (nella sua fila di poltrone) relativa al biglietto sia maggiore o uguale a 0. Infatti, il numero di poltrona varia tra 0 (compreso) e `nPostiPerRiga-1`.



## 5: Eventuali altre informazioni utili

- **Come eseguire CinemApp su Linux**

Sia in creaDB.py (a riga 5) che in inserimento.py (a riga 8), sostituire root con il proprio username seguito da :password (con, al posto di password, la propria password) se è presente una password.

Infatti, la forma del database URL deve essere la seguente:

dialect+driver://username:password@host

In questo modo verrà creato l'engine.

Da terminale, dirigersi nella cartella CinemApp ed eseguire l'applicazione con: export

FLASK\_APP=application.py; export FLASK\_ENV=development; flask run

Aprire il browser recarsi su <http://localhost:5000/>

L'utente di tipo gestore ha come indirizzo mail [stefano.calzavara@gmail.com](mailto:stefano.calzavara@gmail.com) e come password

Admin\_123 mentre l'utente di tipo cliente ha come indirizzo mail

[francesco.camporese99@gmail.com](mailto:francesco.camporese99@gmail.com) e come password User\_456 ma è possibile creare altri clienti utilizzando la pagina signup.

Testato su macchina con Ubuntu 20.04 LTS a 64bit e browser Google Chrome.