BLG317E – DATABASE SYSTEMS TERM PROJECT REPORT

Project Name: MoviMovi

Name & Surname: Berkay Emre Keskin

ID: 150240721

1. About the Project

MoviMovi is a website that users can search for movies/genres, add movies to their watchlist, give reviews and ratings and add friends to see their watchlists and reviews. It is a movie recommendation / review website. I used python (flask) for the backend (urls, endpoints and database connection) and html,css,javascript for the frontend.

2. Implementation Details

a. Authentication

1. Implementation

I created an auth blueprint for the website. This auth blueprint has 3 routes and 1 load function for the user.

a. Login

b. Register

c. Logout

d. Load Logged User

In register function, I get the username, password and email of the user from the form in the frontend. Then I create a connection to my database and insert the new user if he/she is not in my user list.

```
cursor.execute("SELECT * FROM users WHERE username = %s", (us
ername,))
if cursor.fetchone():
    flash("Username already taken", "danger")
    return render_template('/auth/register.html')

cursor.execute("SELECT * FROM users WHERE email = %s", (emai
l,))

if cursor.fetchone():
    flash("Email already taken", "danger")
    return render_template('/auth/register.html')

cursor.execute("INSERT INTO users (username, email, user_pass
word) VALUES (%s, %s, %s)", (username, email, password))
```

In login function, I get the username and password and check my database for the correct match using a select query

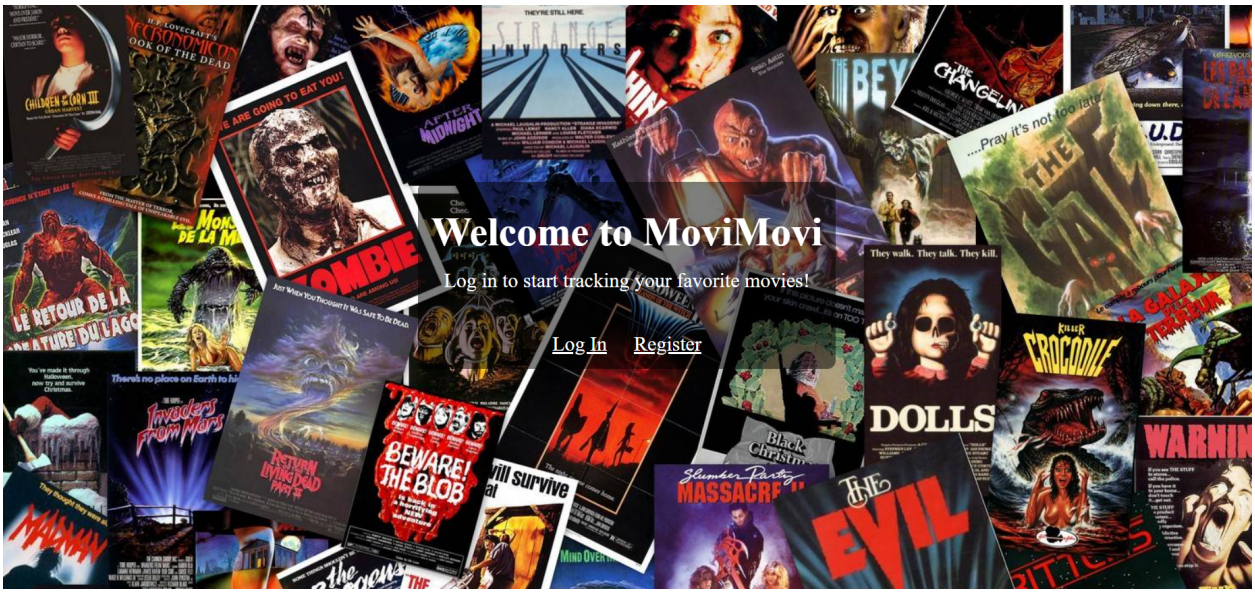
```
cursor.execute("SELECT * FROM users WHERE username = %s AND u
ser_password = %s", (username, password))
```

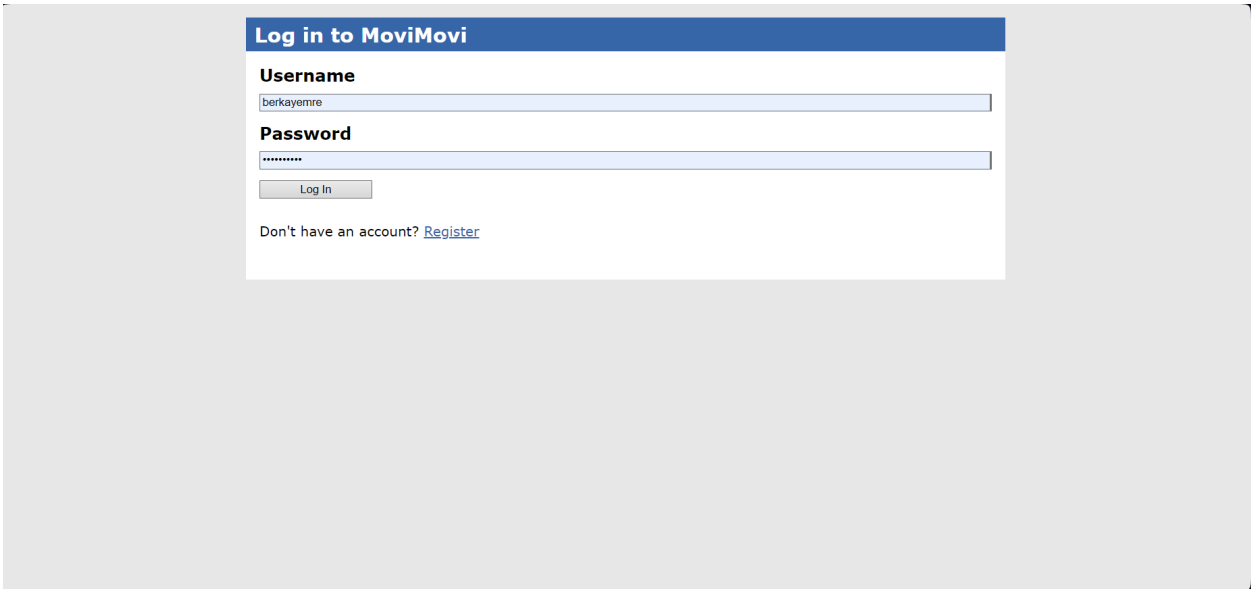
If there is a returned user, I update the session variables such as logged_in = True etc. Then I redirect the user to his own user page.

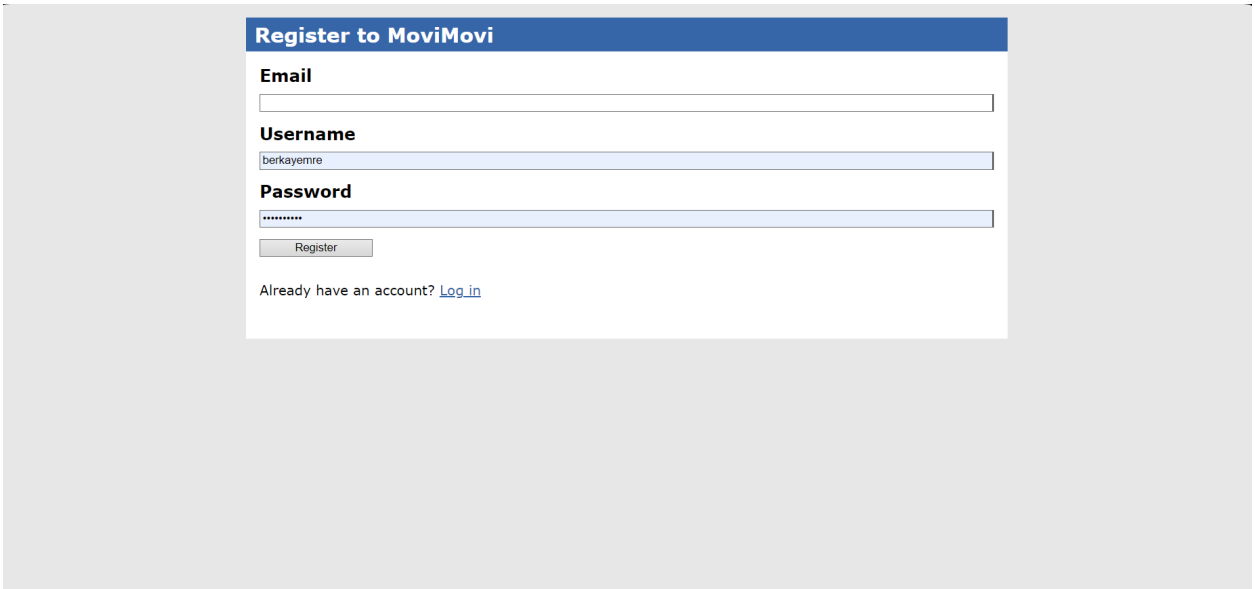
In logout function, I pop the user from the session and update the session variables.

In load logged user function, I check for the user if he/she is logged in or not. According to the return, I update the g.user.

2. Screenshots


Welcome to MoviMovi
Log in to start tracking your favorite movies!
Log In
Register


Log in to MoviMovi
Username
berkayemre
Password
Log In
Don't have an account? Register



b. Movies

I created an movies_bp blueprint for the whole movie related functions. There are also helper functions for the main functions.

a. Get all movies
b. Get a spesific movie
c. Add movie to a watchlist
d. Remove a movie from the watchlist
e. Add a review to a movie
f. Delete a review from a movie
g. Check the watchlist for a spesific movie
h. Search a movie using title or genre
i. Update a review given to the movie

a. Get all movies

   At first, I get the total number of movies. Then I make pagination using the count of the movies. I get my movie posters from a tmdb.org. My dataset has imdb and tmdb id's of the movies. I downloaded the tmdb posters using the api request and saved them in the posters folder. I get the posters from there.

```
def get_movie_poster(tmdb_id):
    poster_path = os.path.join('app', 'static', 'posters',
f'{tmdb_id}.jpg')
    if os.path.exists(poster_path):
        print(f'Poster for movie ID {tmdb_id} found')
        return url_for('static', filename=f'posters/{tmdb_
id}.jpg')
    else:
        print(f'Poster for movie ID {tmdb_id} not found')
    return None
```

I do my pagination using the count of the movies. For each page I send a request for 12 movies and I calculate the offset using the page I am currently in.

Then my other query is selecting the all movies and links with a join.

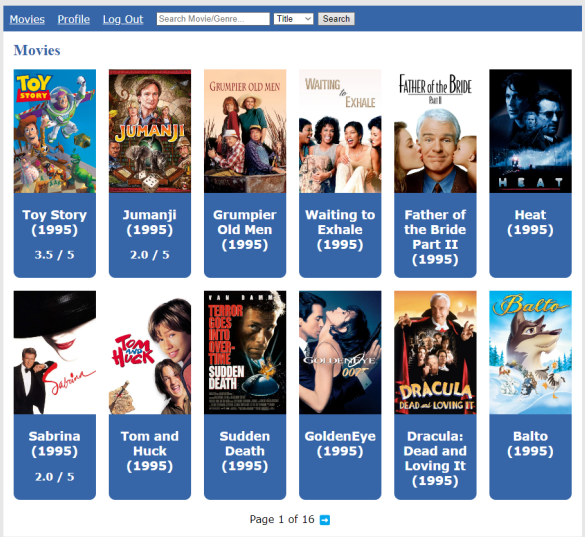
```
cursor.execute('''
            SELECT *
            FROM movies
            INNER JOIN links ON movies.movieId = links.mov
ieId
            LIMIT %s OFFSET %s
        ''', (movies_per_page, offset))
        movies = cursor.fetchall()
```

After that I get the average ratings of these movies using AVG()

```
cursor.execute('''
            SELECT ratings.movieId, AVG(ratings.rating) as
rating
            FROM ratings, movies
            WHERE ratings.movieId = movies.movieId
            GROUP BY ratings.movieId
```

```
        ''')
        ratings = cursor.fetchall()
```

Then for each movie, I add the movie poster url to the movie data for the frontend. I also add the round of the average rating of each movie to it's data. After all of these operations, I send the movies and other informations to my frontend and render my movies page.

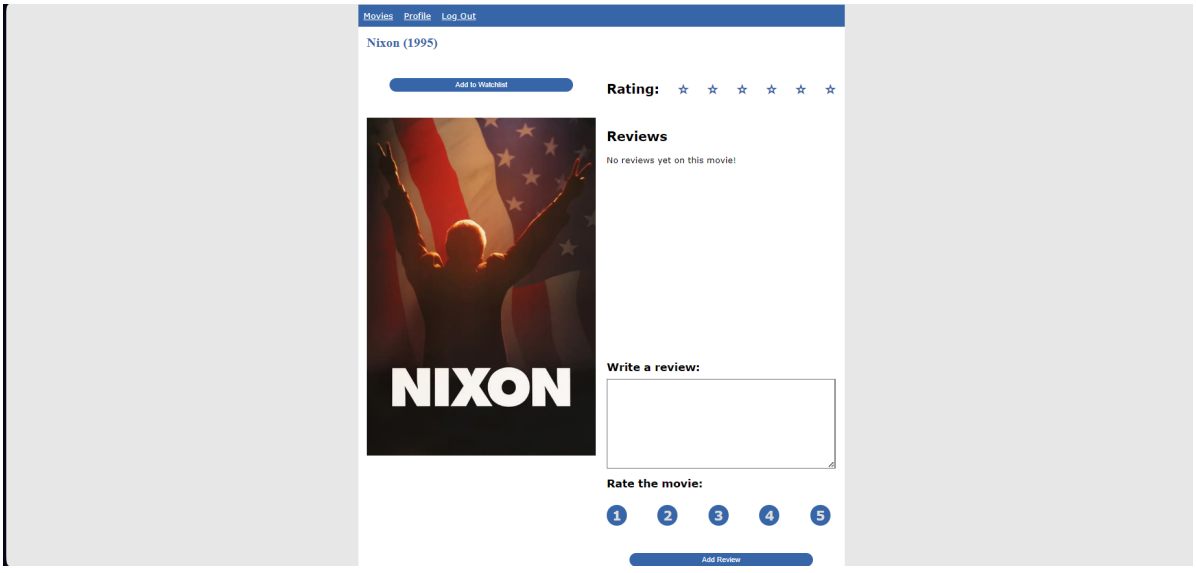


b. Get spesific movie

For getting a spesific movie, I use the movie's id for the all queries. I get the movie and link and then I get the reviews and ratings of that movie. I check if the user added this movie in it's watchlist and I also check if the user added a review or not. These checks are for the button style and forms to send.

```
cursor.execute("SELECT * FROM movies WHERE movieId = %s",
(movie_id,))
        movie = cursor.fetchone()
        cursor.execute("SELECT * FROM ratings WHERE movieI
d = %s", (movie_id,))
        ratings = cursor.fetchall()
        cursor.execute("SELECT * FROM links WHERE movieId
= %s", (movie_id,))
        link = cursor.fetchone()
```

```
        cursor.execute("SELECT * FROM reviews WHERE movieI
d = %s", (movie_id,))
        reviews = cursor.fetchall()
        cursor.execute("SELECT * FROM users")
        users = cursor.fetchall()
        cursor.execute("SELECT * FROM watchlists WHERE mov
ieId = %s AND userId = %s", (movie_id,session['user_id']))
        if cursor.fetchone():
            movie['in_watchlist'] = True
        else:
            movie['in_watchlist'] = False
        cursor.execute("SELECT * FROM ratings WHERE movieI
d = %s AND userId = %s", (movie_id,session['user_id']))
        if cursor.fetchone():
            movie['rated'] = True
        else:
            movie['rated'] = False
```

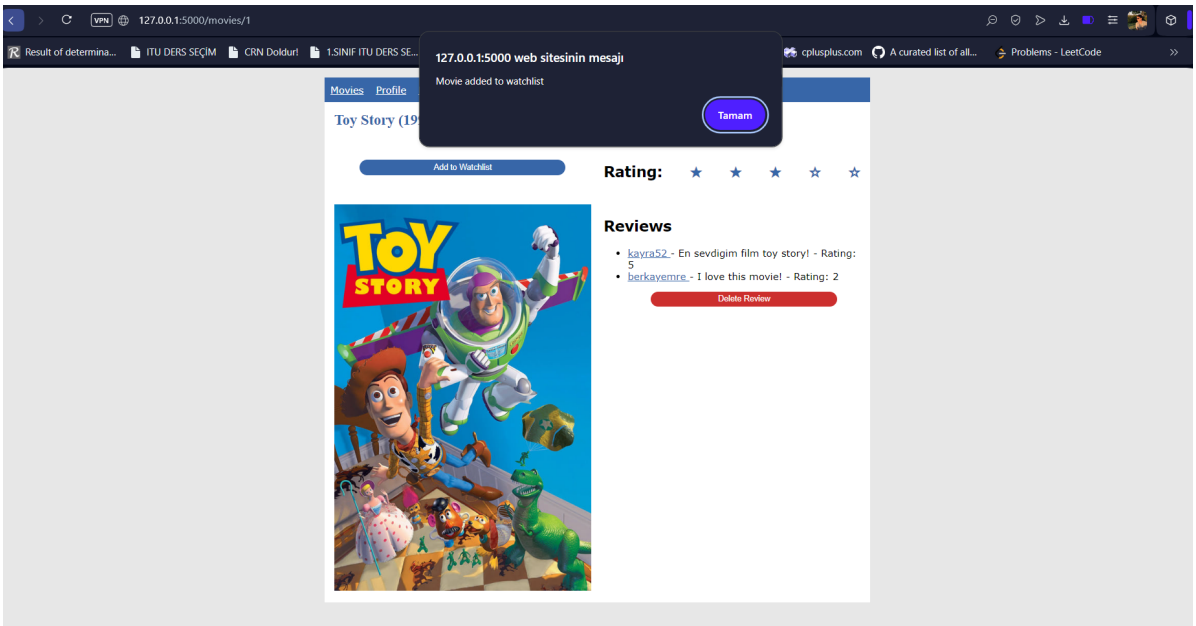


c. Add movie to watchlist

At first, I check for the movie in the wathclist table if it exists for the current user. Then if movie does not exist in the watchlist of the person, I insert the

movie to their watchlist. If it exists, then I return an error using a pop-up.

```
cursor.execute("SELECT * FROM watchlists WHERE userId = %s
AND movieId = %s", (user_id, movie_id))
if cursor.fetchone():
   return return_error("Movie already in watchlist", 400)
cursor.execute("INSERT INTO watchlists (userId, movieId) V
ALUES (%s, %s)", (user_id, movie_id))
```



d. Remove a movie from the watchlist

At first, I check for the movie if it is in the watchlist table for the current user. If it exists, then I delete the movie from the person's watchlist. If it does not exist, then I return an error using a pop-up.

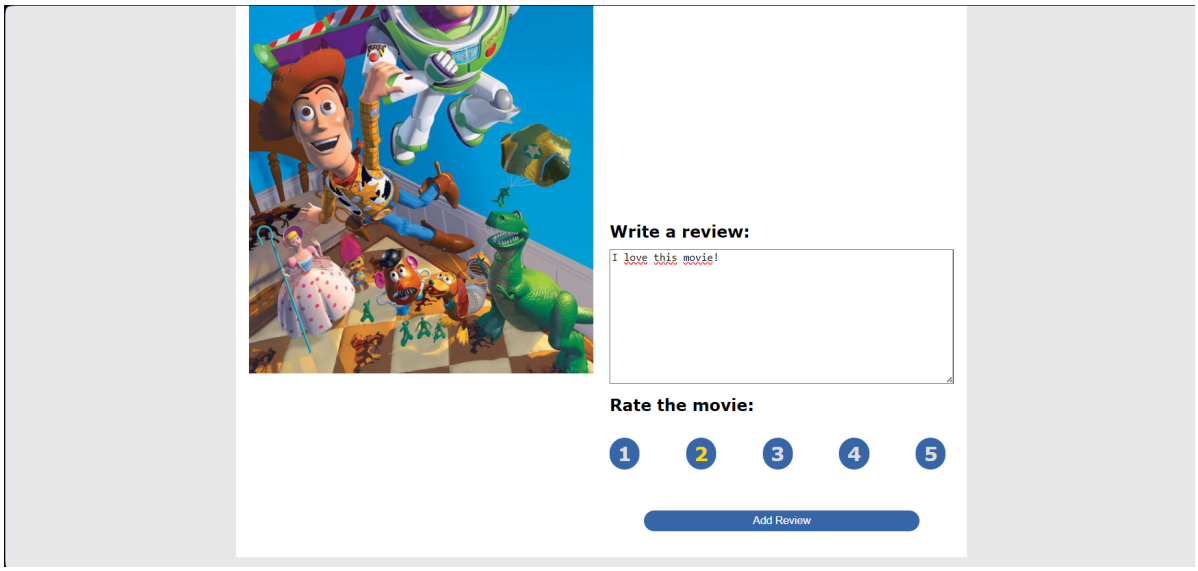
```
cursor.execute("SELECT * FROM watchlists WHERE userId = %s
AND movieId = %s", (user_id, movie_id))
        if not cursor.fetchone():
            return return_error("Movie not in watchlist",
400)
```

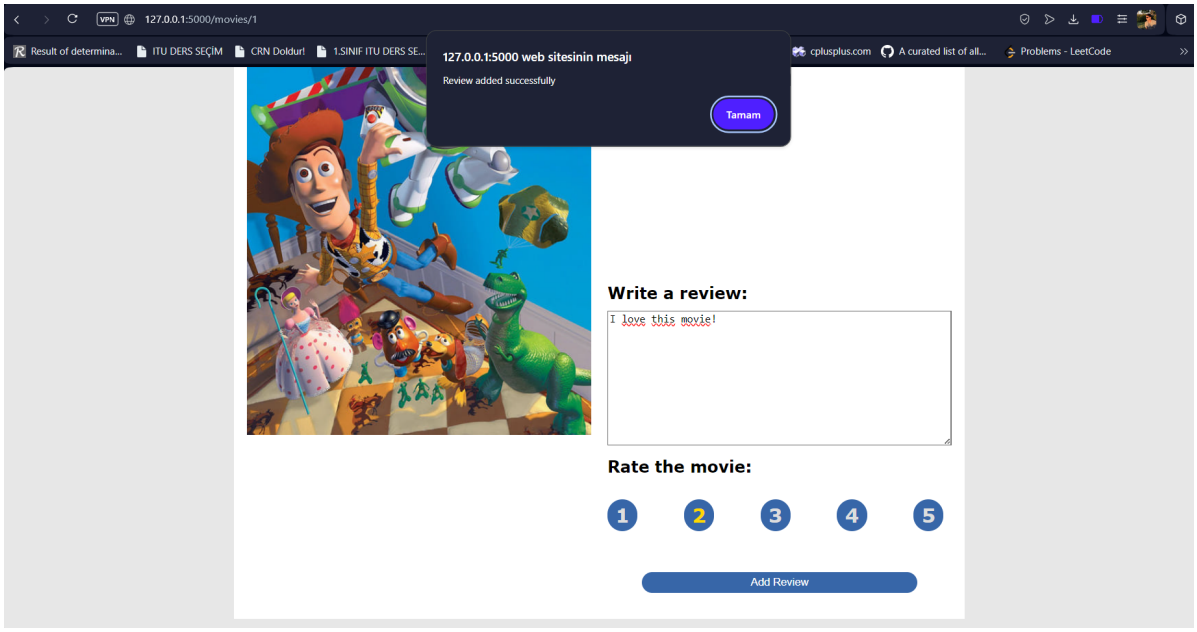
```
        cursor.execute("DELETE FROM watchlists WHERE userI
d = %s AND movieId = %s", (user_id, movie_id))
```

e. Add review to a movie

At first, I check for the given form data. If rating and review is not given together, I return an error. After that I check if the user given a review and rating to the movie. If he/she did, I return an error, if not I insert the review and rating to it's tables. I check for the rating between 0 and 5 in my sql query.

```
cursor.execute("SELECT * FROM reviews WHERE userId = %s AN
D movieId = %s", (user_id, movie_id))
        if cursor.fetchone():
            return return_error("You have already reviewed
this movie", 400)

cursor.execute("SELECT * FROM ratings WHERE userId = %s AN
D movieId = %s", (user_id, movie_id))
if cursor.fetchone():
    return return_error("You have already rated this movi
e", 400)

cursor.execute("INSERT INTO reviews (userId, movieId, revi
ew_text) VALUES (%s, %s, %s)", (user_id, movie_id, review_
text))
cursor.execute("""
    INSERT INTO ratings (userId, movieId, rating)
    SELECT %s, %s, %s
    WHERE %s BETWEEN 0 AND 5
""", (user_id, movie_id, rating, rating))
```



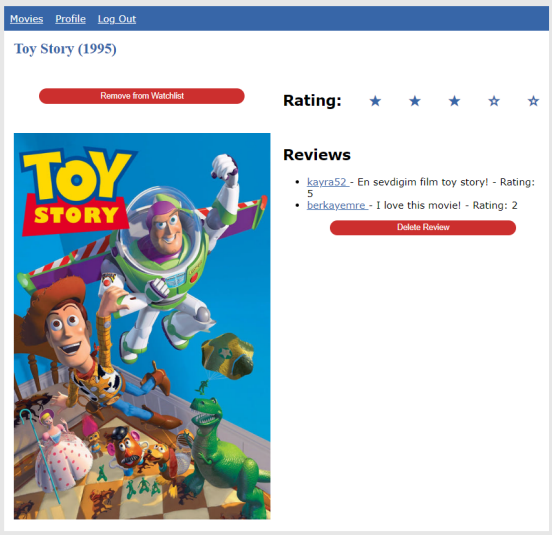


f. Delete a review from a movie

I check if the user has given a rating and review. If he/she has given a review and rating, then I simply delete them from their tables using the userId and movieId.

```
cursor.execute("SELECT * FROM reviews WHERE userId = %s AN
D movieId = %s", (user_id, movie_id))
        if not cursor.fetchone():
            return return_error("You have not reviewed thi
```

```
s movie", 400)

cursor.execute("DELETE FROM reviews WHERE userId = %s AND
movieId = %s", (user_id, movie_id))
cursor.execute("DELETE FROM ratings WHERE userId = %s AND
movieId = %s", (user_id, movie_id))
connection.commit()
```



g. Check the watchlist for a spesific movie

I created this function to return if the spesific movie is in the watchlist or not. I simply use a select query using userId and movieId for the watchlist table.

```
cursor.execute("SELECT * FROM watchlists WHERE userId = %s
AND movieId = %s", (user_id, movie_id))
        if cursor.fetchone():
            return "True", 200
        else:
            return "False", 200
```

h. Search a movie using title or genre

At first, I get the query from the searchbar and type of the search (title or genre). Then I check for the parameters and return errors accordingly. Like the get_movies function, I also state the movies per page and page numbers according to my movies which will be returned from my search query.

If the type of the search is title, I get the count of the movies which starts with the given query (For example, query = “Fath”, I check using the {Fath]%) in the title column of movies table. After that I get the movies and links together using a left join and limit them with movies_per_page and I also use the offset value. Then after getting the results, I simply add the poster url’s to the movies for the frontend. Then I render my movies page using the informations of the queries. If the type is genre, then I get my movies using the genre column of movies table.

```
cursor.execute("""
                SELECT COUNT(*) AS total
                FROM movies
                WHERE title LIKE %s
            """, (f"{query}%",))
            total_movies = cursor.fetchone()['total']

            # Fetch movies for the current page
            cursor.execute("""
                SELECT movies.*, links.*
                FROM movies
                LEFT JOIN links ON movies.movieId = links.
movieId
                WHERE movies.title LIKE %s
                LIMIT %s OFFSET %s
            """, (f"{query}%", movies_per_page, offset))
            result = cursor.fetchall()
```

```
cursor.execute("""
                SELECT COUNT(*) AS total
                FROM movies
                WHERE genres LIKE %s
```

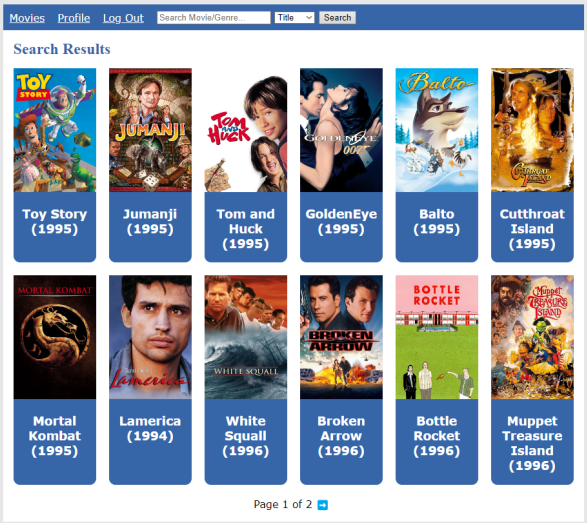
```
            """, (f"%{query}%",))
            total_movies = cursor.fetchone()['total']

            # Fetch movies for the current page
            cursor.execute("""
                SELECT movies.*, links.*
                FROM movies
                LEFT JOIN links ON movies.movieId = links.
movieId
                WHERE genres LIKE %s
                LIMIT %s OFFSET %s
            """, (f"%{query}%", movies_per_page, offset))
            result = cursor.fetchall()
```

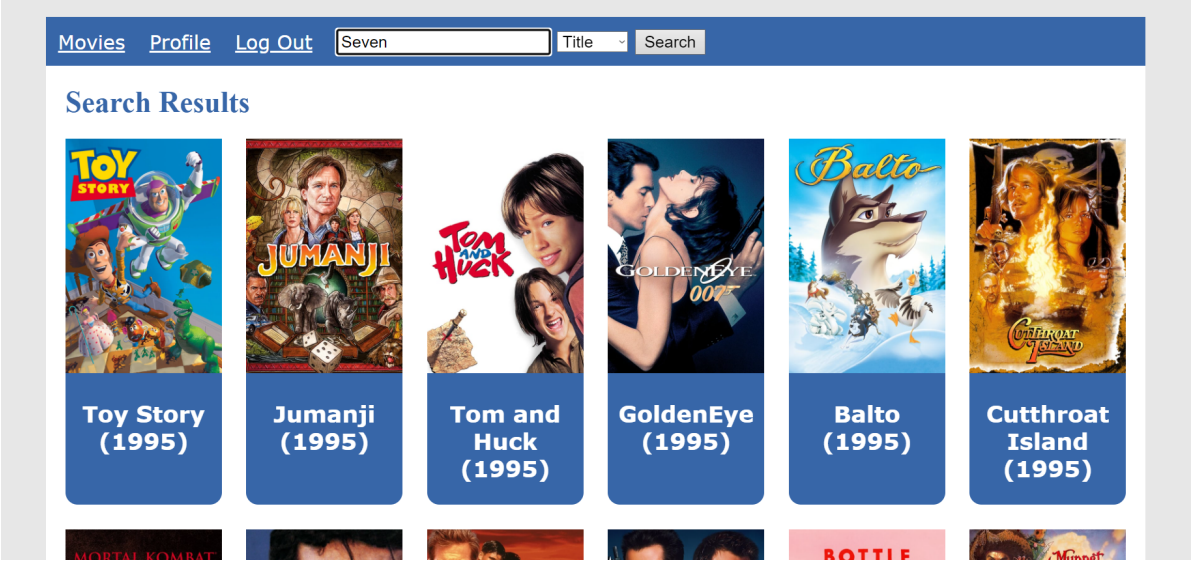
i. Update the review

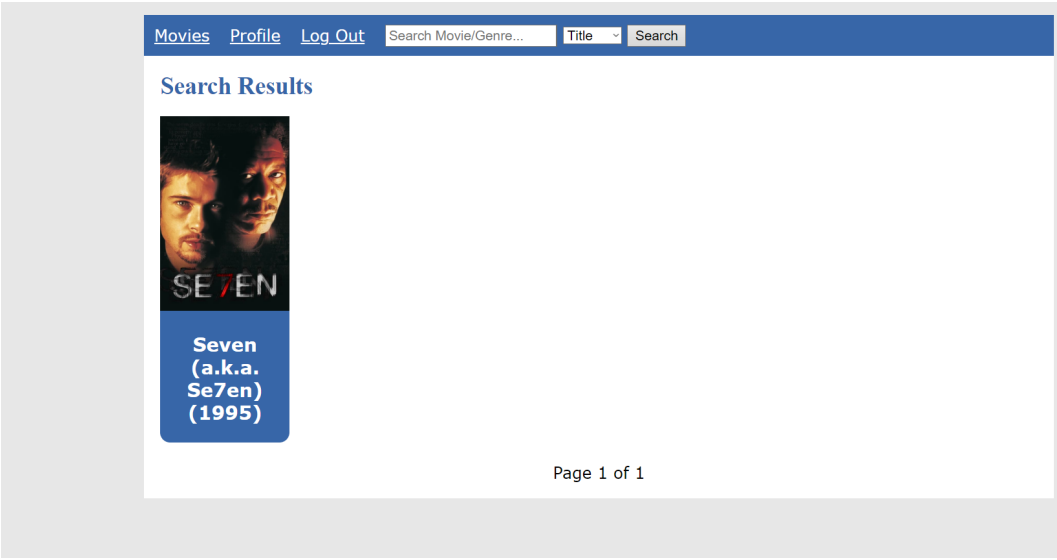
If user has given a review to a movie, I change the review part to update. User can update his/her review and rating at the same time.

```
cursor.execute("SELECT * FROM reviews WHERE userId = %s AND r
        if not cursor.fetchone():
            return return_error("You have not reviewed this r

        cursor.execute("UPDATE reviews SET review_text = %s W
        cursor.execute("UPDATE ratings SET rating = %s WHERE
```


Movies
Profile
Log Out
Adventure
Title
Search
Title
Genre
Movies
Toy Story (1995) 3.5 / 5
Jumanji (1995) 2.0 / 5
Grumpier Old Men (1995)
Waiting to Exhale (1995)
Father of the Bride Part II (1995)
Heat (1995)


Movies
Profile
Log Out
Search Movie/Genre...
Title
Search
Search Results
Toy Story (1995)
Jumanji (1995)
Tom and Huck (1995)
GoldenEye (1995)
Balto (1995)
Cutthroat Island (1995)
Mortal Kombat (1995)
Lamerica (1994)
White Squall (1996)
Broken Arrow (1996)
Bottle Rocket (1996)
Muppet Treasure Island (1996)
Page 1 of 2


Movies
Profile
Log Out
Seven
Title
Search
Search Results
Toy Story (1995)
Jumanji (1995)
Tom and Huck (1995)
GoldenEye (1995)
Balto (1995)
Cutthroat Island (1995)


Movies
Profile
Log Out
Search Movie/Genre...
Title
Search
Search Results
Seven (a.k.a. Se7en) (1995)
Page 1 of 1

Reviews

- kayraoner - A great masterpiece from Michael Mann! - Rating: 4

Delete Review

- metesirin - Nice! - Rating: 3

Update your review:

Update your rate to the movie:

1 2 3 4 5

Update Review

c. User

User blueprint is about the user info crud operations and friend crud operations.

a. Get User
b. Add friend
c. Remove friend
d. Get all friends
e. Get common friends

a. Get User

I used several queries for the user's information, friendships, watchlist movies and reviewed movies. At first, I get the watchlist using this query:

```
cursor.execute("SELECT * FROM watchlists WHERE userId = %s
LIMIT %s", (id, movies_per_page))
    watchlists = cursor.fetchall()

    watchlists_movies = []
    for watchlist in watchlists:
        cursor.execute('''
            SELECT * FROM movies, links
            WHERE movies.movieId = links.movieId
            AND movies.movieId = %s
        ''', (watchlist['movieId'],))
        movie = cursor.fetchone()
        if movie is None:
            continue
        movie['poster_url'] = get_movie_poster(movie['tmdb
Id'])
        watchlists_movies.append(movie)
```

I get the watchlist movies using the SELECT command and I also get the link of the movie with it because of the poster.

Then I get the reviews and ratings using this query:

```
cursor.execute('''
        SELECT * FROM reviews, ratings, movies, links
        WHERE movies.movieId = links.movieId
        AND reviews.movieId = ratings.movieId
        AND reviews.userId = ratings.userId
        AND movies.movieId = reviews.movieId
        AND reviews.userId = %s LIMIT %s'''
    , (id, movies_per_page))
    reviews_and_ratings = cursor.fetchall()
```

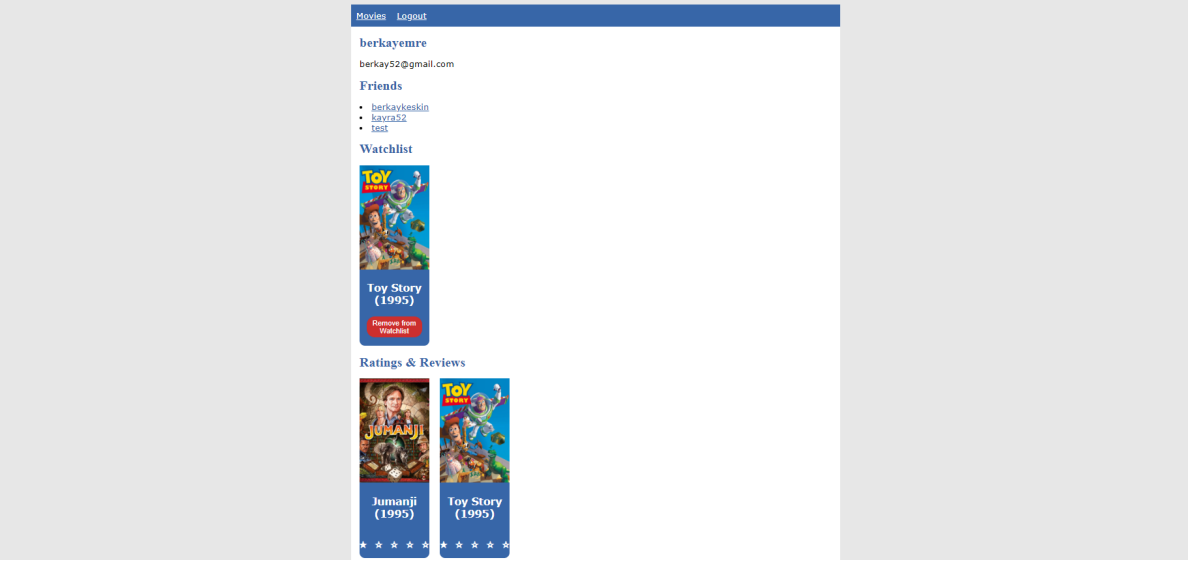
```
    for movie in reviews_and_ratings:
        movie['poster_url'] = get_movie_poster(movie['tmdb
Id'])
```

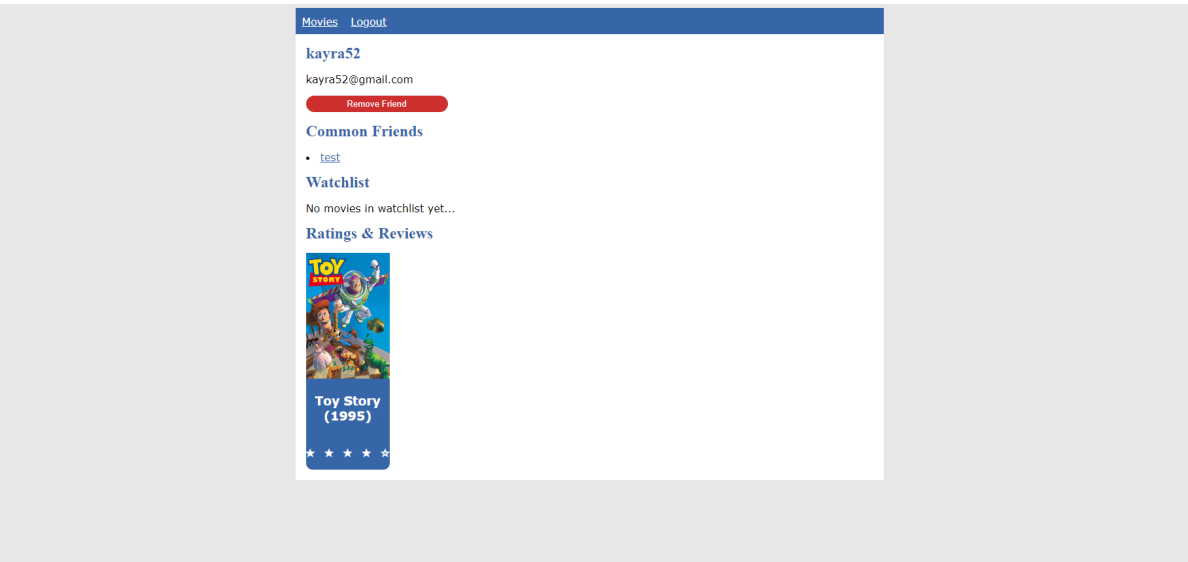
I get the reviewed and rated movies in this query using SELECT command and I join them with the movieIds and userIds.

At the end, I check for the friendships using this query:

```
user_id = session.get('user_id')
    cursor.execute('SELECT * FROM friendship WHERE userId
= %s AND friendId = %s', (user_id, id))
    friendship = cursor.fetchone()
    user['is_friend'] = friendship is not None

    friends = get_all_friends(user_id)
    user['friends'] = []
    for friend in friends:
        cursor.execute('SELECT * FROM users WHERE userId =
%s', (friend['friendId'],))
        user['friends'].append(cursor.fetchone())

    common_friends = get_common_friends(user_id, id)
    user['common_friends'] = []
    for friend_id in common_friends:
        cursor.execute('SELECT * FROM users WHERE userId =
%s', (friend_id,))
        friend = cursor.fetchone()
        user['common_friends'].append(friend)
```

If the page is user's own page, I show the user's friends. If user visits another user's page, I show the common friends with them.





b. Add friend

In this function, I create a friendship between the current user and the given userId. At first, I check if the friendship exists, then if there is not I insert the both users into the friendship.

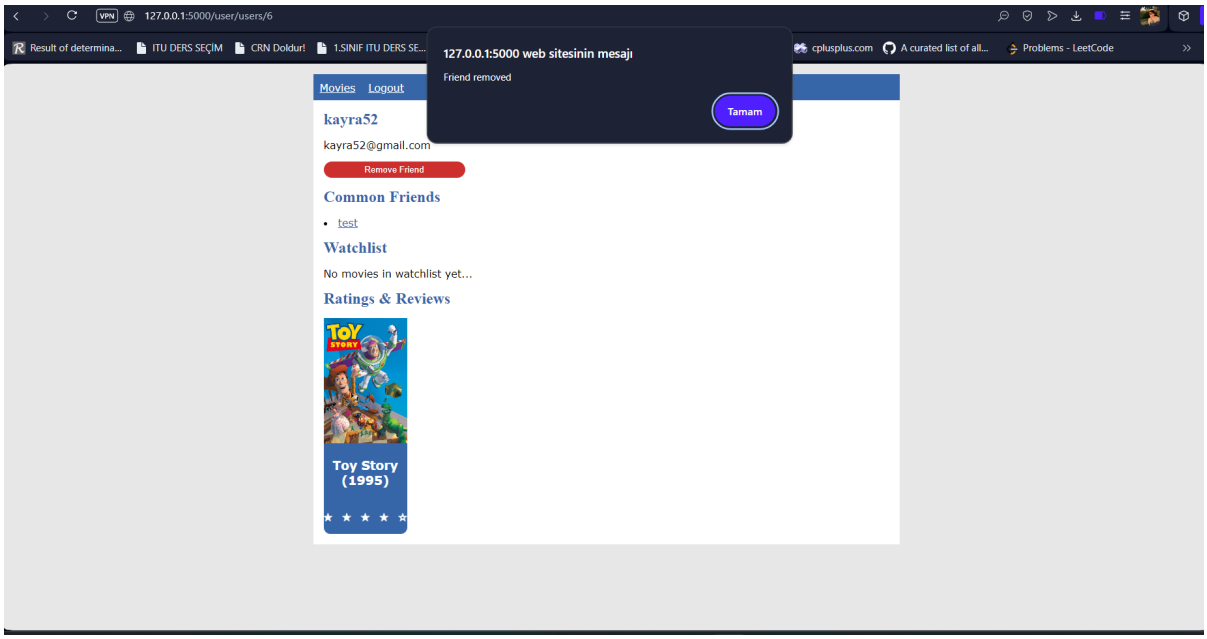
```
if friendship is not None:
        return return_error('Friend already exists', 409)

    cursor = connection.cursor(dictionary=True)
    cursor.execute('INSERT INTO friendship (userId, friendI
```

```
d) VALUES (%s, %s)', (session['user_id'], id))
    cursor.execute('INSERT INTO friendship (userId, friendI
d) VALUES (%s, %s)', (id, session['user_id']))
```

c. Remove friend

With the same logic in the add friend, I check if there is a friendship between the current user and given userId. If there is, I remove the friendship between the users.

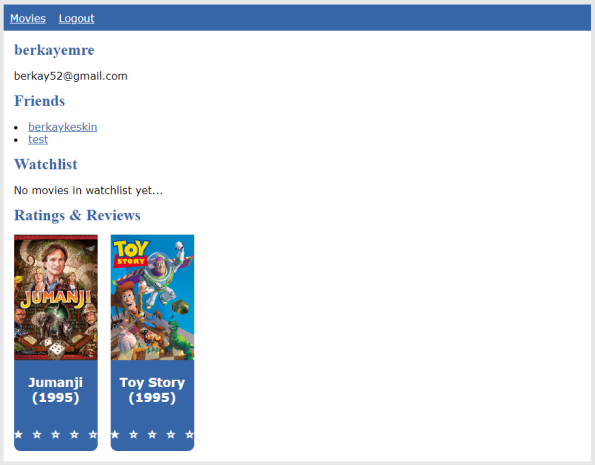
```
friendship = get_friendship(session['user_id'], id)
    if friendship is None:
        return return_error('Friend not found', 404)

    cursor = connection.cursor(dictionary=True)
    cursor.execute('DELETE FROM friendship WHERE userId = %
s AND friendId = %s', (session['user_id'], id))
```



d. Get all friends

It is just simply a SELECT query for the userId in the friendship table.

```
cursor.execute('SELECT * FROM friendship WHERE userId = %
s', (user_id,))
```



e. Get common friends

I use a SELECT query with inner join using the current userId and friend userId. I get the common friends from the query and return all the friends of given userId and current userId.

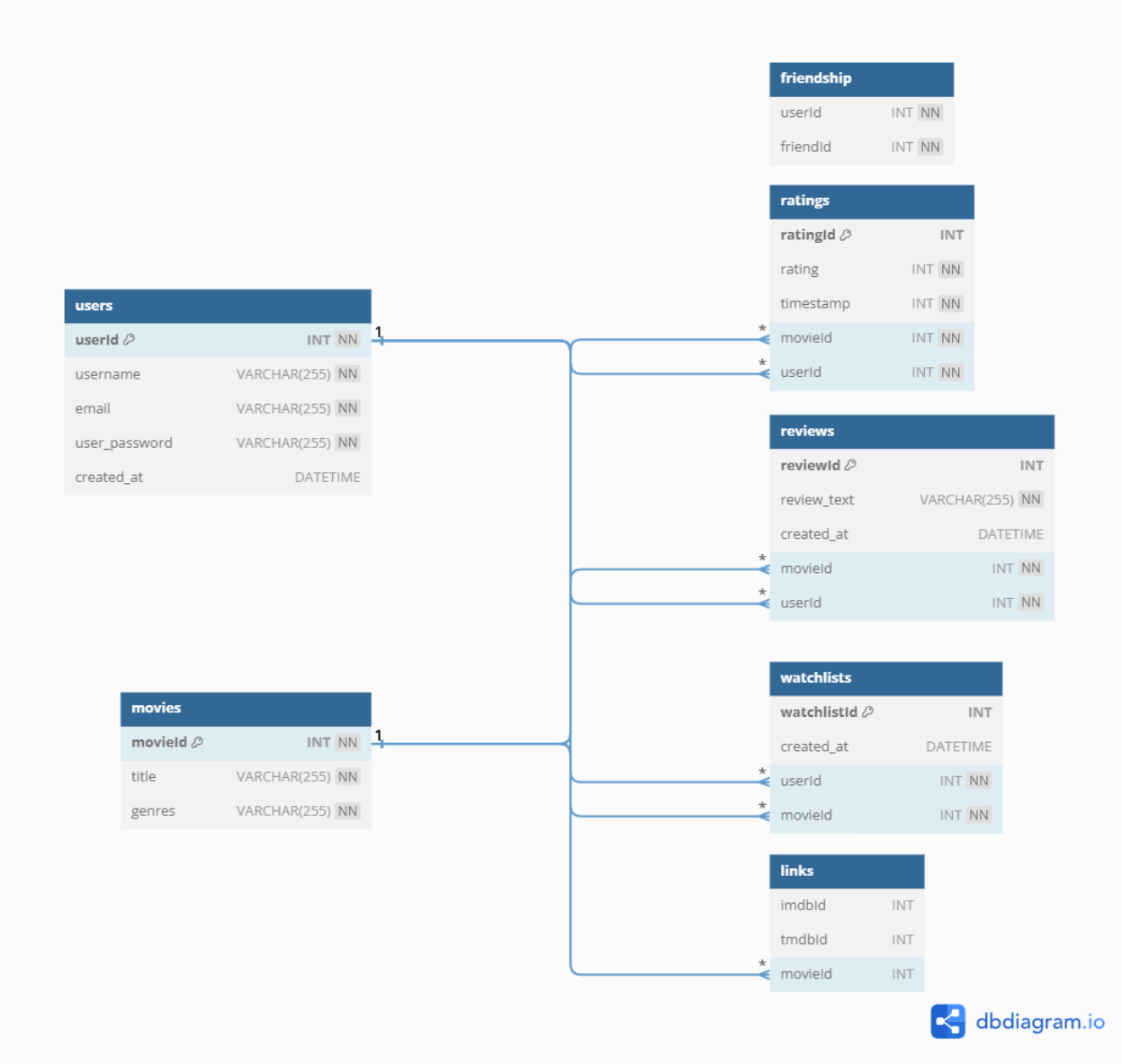
```
cursor.execute('''
        SELECT DISTINCT f1.friendId
        FROM friendship f1
        INNER JOIN friendship f2 ON f1.friendId = f2.friend
Id
        WHERE f1.userId = %s AND f2.userId = %s
    ''', (user_id, friend_id))
    common_friends = cursor.fetchall()
    cursor.close()
    connection.close()

    return [friend['friendId'] for friend in common_friend
s]
```

f. Get friendship

This is a simple helper function to use in another functions. I simply get the friendship if there is any.

```
cursor.execute('SELECT * FROM friendship WHERE userId = %s
AND friendId = %s', (user_id, friend_id))
```

Database SQL Diagram



There are 7 tables in my database. users and movies are the main tables and ratings, reviews, watchlists and links have foreign keys of these main tables. Friendship table does not have any connection with these tables. Users table consists userId (PRIMARY), username, email, password and created_at. Movies table consists movieId(PRIMARY), title, genres because of the data table I used in this project. The other tables are using the movieId and userId as their foreign keys.