Project Milestone-3 Report

TeamName - 3 Amigos
Project Name : CineBase

https://github.com/div652/3-Amigos

Front-end Design

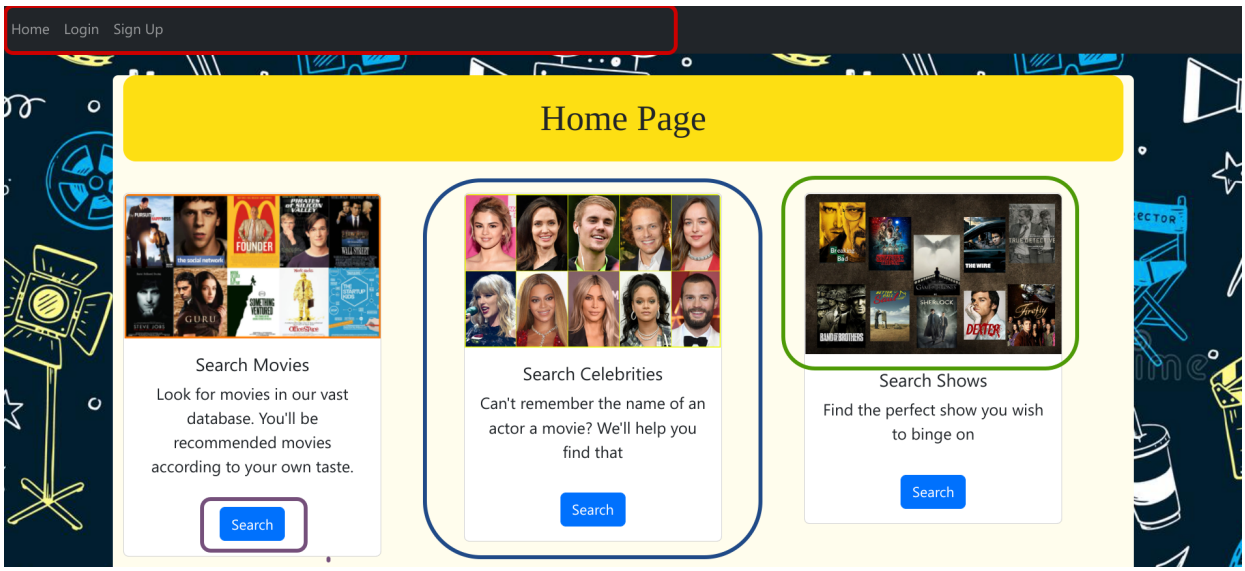
We designed our application using **Template inheritance in HTML**. We have a base template that includes the folowing features :

1. Interactive navigation bars
2. Page background
3. Content container

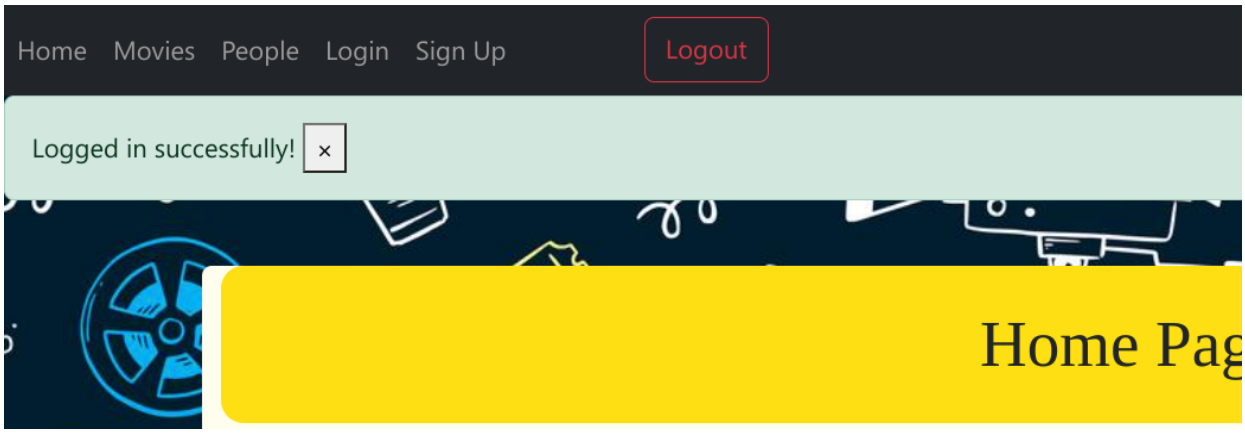
   and other html properties

This is how our main page looks like :

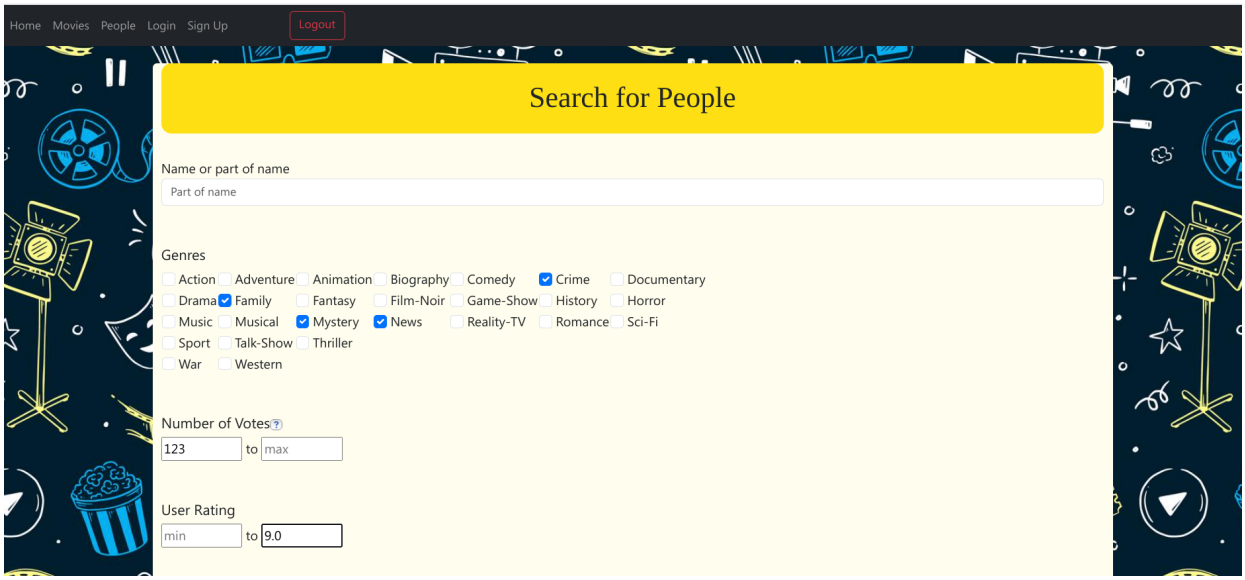
Notice the nav bar in red, the search buttons in purple boxes, floating cards etc.



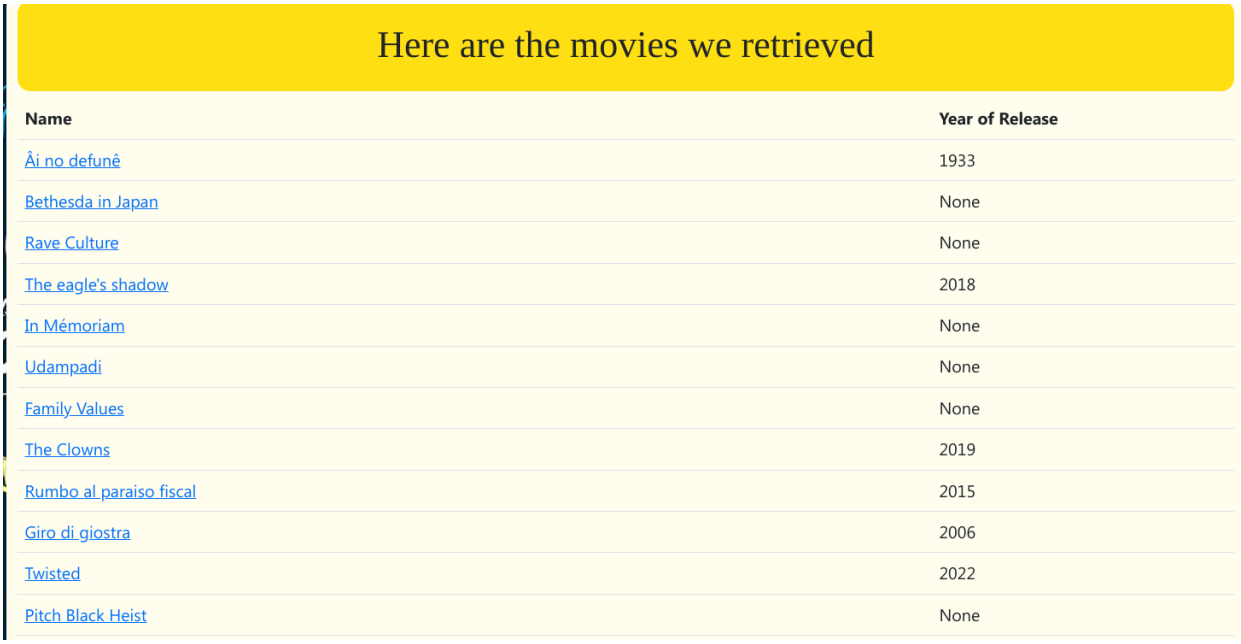
This is how we show when the login is successful using **Mesaage flashing**, notice how the navbar has been updated and include logout options:



This is how the page to search for movie looks like. We include buttons and prompts and instruction messages :



This is how the search results look like, each result is a hyper link on which if you click another search happens and we display all information and rating options for the movie.



Here are the movies we retrieved

Name	Year of Release
Âi no defunê	1933
Bethesda in Japan	None
Rave Culture	None
The eagle's shadow	2018
In Mémoriam	None
Udampadi	None
Family Values	None
The Clowns	2019
Rumbo al paraiso fiscal	2015
Giro di giostra	2006
Twisted	2022
Pitch Black Heist	None

Supported Transactions

The supported transaction by our database is as follows :

1. **Querying/searching** through the database : We support a wide range of queries , we pickup information wanted by users using "forms" like the ones above or using links that the user has clicked. Once we have this information we populate our sql queries with this information provided and query our database using our well optimised searches and implemented indices. We have elaborated more about our queries below.
2. **Updation**
    1. Whenever a user rates a movie, we store the user ratings in table "rated". We have implemented a **trigger** function that updates the movie's average rating and number of votes.
    2. Whenever a user creates a new account we add his/her details to the database and update our authentication tables.

       We have also implemented **foreign-key relations** in our database so any time a user makes a change to the data base all the required tables and data gets updated.
3. **Deletion**
    1. Users have the option to delete their account from Cinebase, in this case we have implemented deletion in our database, we delete all the user preference information that we have stored in our database, and update the authentication to remove the user from our list of registered users.

Our database supports the following transactions:

1. **Querying/searching** through the database : We support a wide range of queries, we pickup information wanted by users using "forms" like the ones above or using links that the user has clicked. Once we have this information we populate our sql queries with this information provided and query our database using our well optimised searches and implemented indices. We have elaborated more about our queries below.

We support the following major queries:

1. **Title**
   When the user provides a movie/series/miniseries title (or substring of the title), and other filters such as the average rating threshold of the movie, release year of the movie/ start year of the series etc., we display all titles from the title table that satisfy the user's constraints.
2. **Person**
   Given the name (or substring of the name) of an actor/actress, director, producer, cinematographer etc. we display the given person's top rated movies/shows etc. and the character (if any) they played in the corresponding movie/series.
3. **Movie info**
   The user provides the name of a movie/miniseries/short etc. and we display all the available information about this title, including the famous actors/directors/composers who worked in the title, the average rating of the title, the runtime of the title etc.
4. **TV series**
   The user may provide constraints such as the number of seasons in a series, min/max number of episodes in the series, its average rating, its start/end year etc. to filter through the series/miniseries in our database.

Queries supported

Our database supports a lot of transactions. We tried to incorporate almost all the possible search criteria a person would like to search a movie on .

Our transactions are broadly divided on two basis :

- Recommendation Searches : These are search for movies, people or shows , where we suggest results based on user's past choice and overall popularity of movies that we calculate using heuristic that we have implemented in sql.
- Quick Searches: These searches do not require the user to login. These are generic searches that can work without the user logging into our database.

  Here are the Recommendation searches that we support :

  By default a user is recommended movies on the basis of their previous searches and overall movie ratings. We have also implemented a genre bases recommendation system that calculates the "scores" for each genre for a given user on the basis of their previous ratings, and looks for movies of genres with high scores (for this user) to recommend.

  We also have the option to choose location based recommendation, in which we look for ratings of users in the same location as the current user, and provide recommendations to them based on the choices of the people around them. When this option is set our movie ordering schema changes and we order movies in a different way to present the final results to the user.

Overall Architecture

Database organisation

Our database contains the following relations (the primary key attributes are in boldface):

1. title(**titleid**, titleType, primaryTitle, genres, isAdult, runTime, startYear, endYear)
2. episode(**titleid**, parenttitleid, seasonnumber, episodenumber)
3. persons(**personid**, primaryname, birthYear, deathYear)
4. principals(**titleid**, **personid**, **category**, characterName)
5. personknownfortitle(**personid**, **titleid**)
6. ratings(**titleid**, averagerating, numvotes)
7. users(**emailid**, username, password, location)
8. rated(**emailid**, **titleid**, rating)
9. personategorytites(**personid**, **category**, numtitles)

Foreign-key relations
We've made created foreign key relations in our database according to the existing foreign key dependencies, some which are explained below.

1. episode(titleid) references title(titleid)
2. principals(personid) references persons(personid)
3. principals(titleid) references title(titleid)
4. rated(emailid) references users(emailid)
5. rated(emailid) references references title(titleid)
6. personknownfortitle(personid) references persons(personid)
7. personknownfortitle(personid) references title(titleid)

Indices
There are btree indices on each of the primary keys in each table. Some of the important indices we created are:

1. Since users search movies/series and actors/directors etc. by their names, we've created hash indices on the primaryTitle and primaryName columns in the corresponding tables.
2. Users may filter through the episodes table based on the range of their number of seaons, episodes etc., so we've added btree indices on these integer columns to support raneg queries effectively.
3. We've added indices on pairs of columns which are often queried together, such as (titleid, personid), (personid, category) etc.
4. We've added indices on average rating and numvotes as users give ranges for

Application Back-end Design - Using Flask

In the backend we have a main.py file that creates an application object. The application object is used to create blueprints for views.py and auth.py files,these are the two main files that handle the actions to be taken on specific routes and how to handle the HTTPS GET and POST requests for any webpage on our site.

Auth.py

This file handles all the authentication routes. Whenver a user logs in or signs ups, he/she fills up an html form and by submitting the form he/she generates a **POST** request. A **POST** request carries along with it information that we defined in the html page for the form in the form of key,value pairs. The python backend is supplied with these pairs and it can process them. On the basis of verification , if suppose a user verifies successfully we set `session['logggedin']` variable to true in our application. `session` is a FLASK feature to maintain user's login status within the application.
To verify if the username and password match we perform an **sql - query** from within the flask application by loading our queries from the `websire/pysql` directory. This directory stores all our sql queries and we supply the necessary values we obtain from the form and generate an sql query, that we execute on our connected database using the following commands :

```
conn = create_db_connection()
cur = conn.cursor()
.... generate formatted quer ....
cur.execute(formatted_query)
ret = cur.fetchall()
```

Our database then executed the formatted query, and returns the results. We get these results using the `ret` variable and store it in them.
`ret` variable stores the information returned by our sql query as **list of tuples** , where each tuple represents the record fetched by our query and we r
Then we extract information from this `ret` variable using python as follows :

```
    new_ret = [(x[0], x[2],x[5]) for x in ret]

    return render_template("loginverif.html",user=curr_user,name_of_user=Name_of_user,
    tuples=new_ret)
```

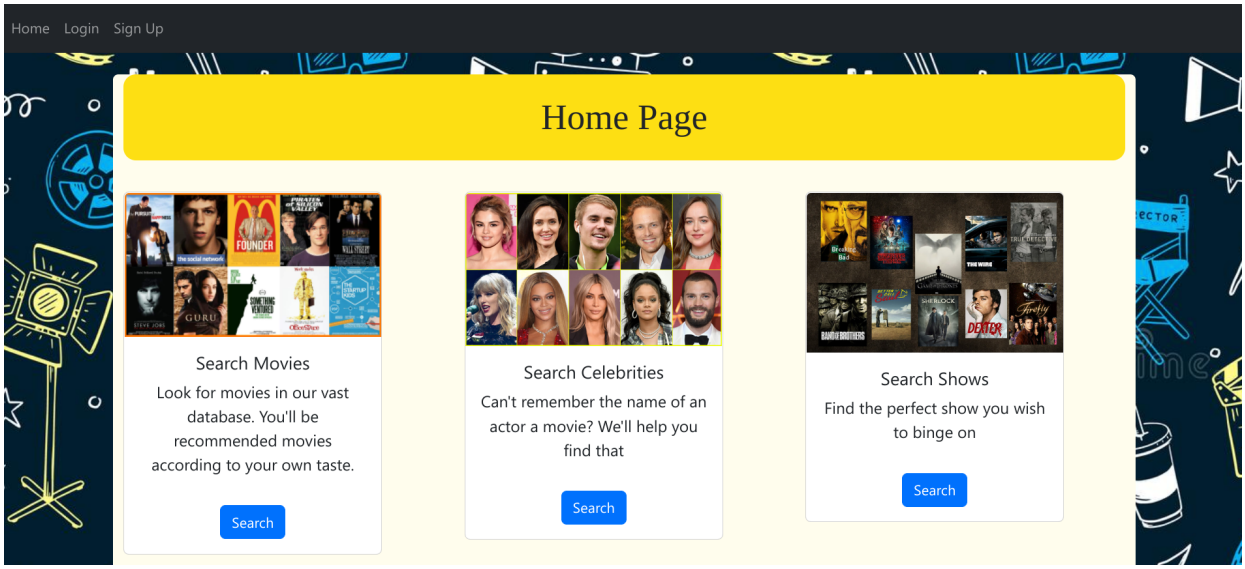
and then call `render_template`.

`render_template` function calls to render an html page and supplies it with necessary details, then we render an html page and it receives input in the from the arguments we supply using render_template function.
This is an example of how our movie search result page utilises the arguments passed to it :

```
            </tr>
          </thead>
          <tbody>
            {% for (id, name,year) in tuples %}
              <tr>
                <td><a href="/movie_info?titleid={{id}}">{{ name }}</a></td>
                <td>{{year}}</td>
              </tr>
            {% endfor %}
          </tbody>
        </table>
    {% endblock %}
```

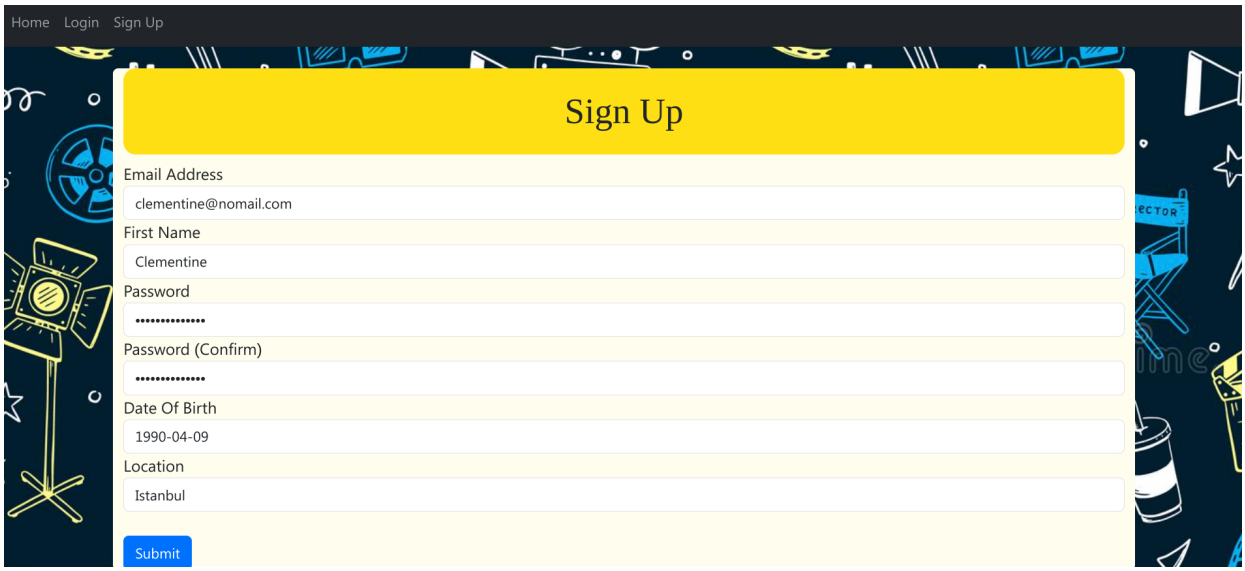
Demonstration Scenario

To demonstrate the front end design let us take you through a tour for a new user 'Clementine' .
This is how our webpage looks like when Clem logs in :



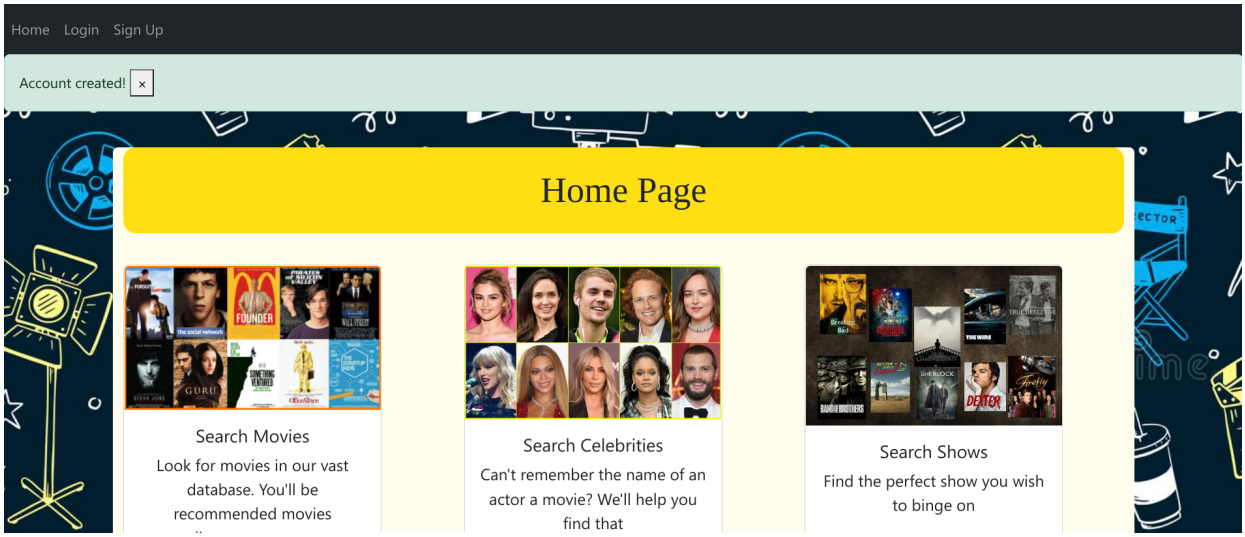
Since most of our queries are for recommendation based seach , we suggest users to login.

Clem goes to the sign up page, since this her first time here . She fills up the form and logs in :

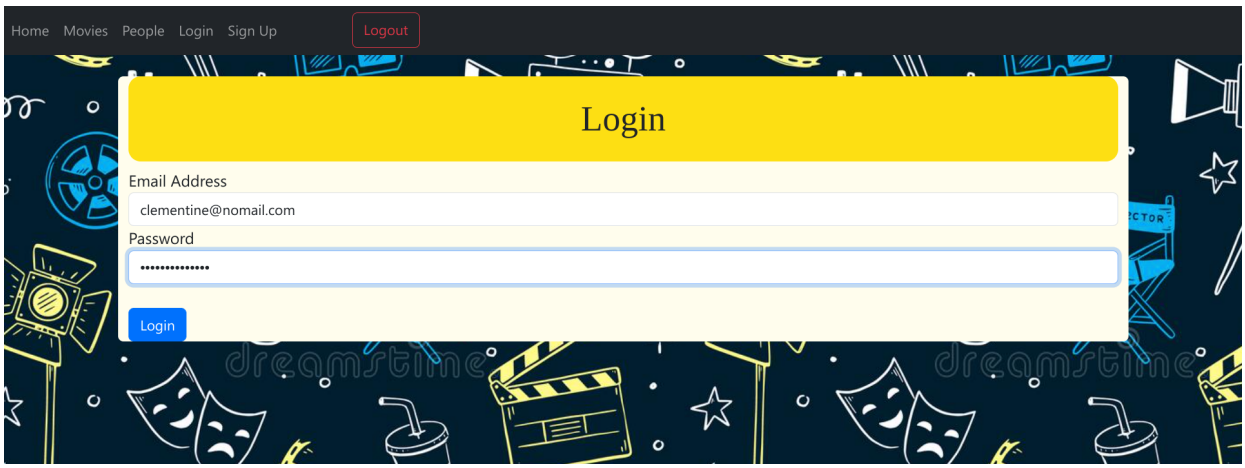


We match here passwords, if they match we store the **SHA256** hash of her password in our database along with other details . (*We care about Clementine's Privacy :)*)

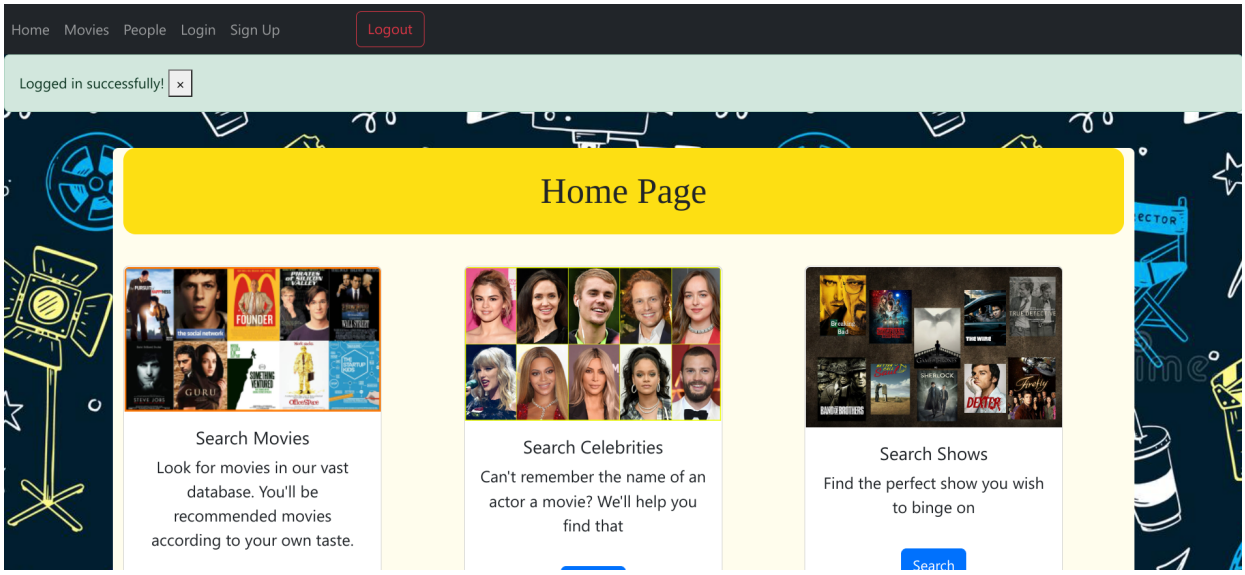
Once the account is created she is taken back to the home page :



Now she can login :

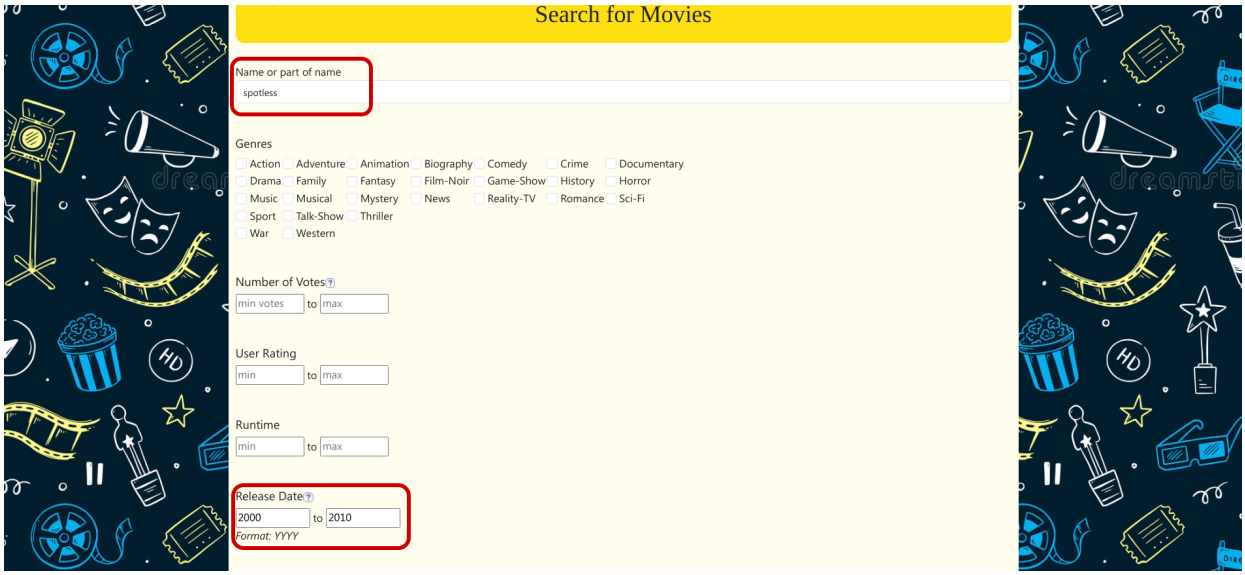


and if her password matches she gets loggedin :



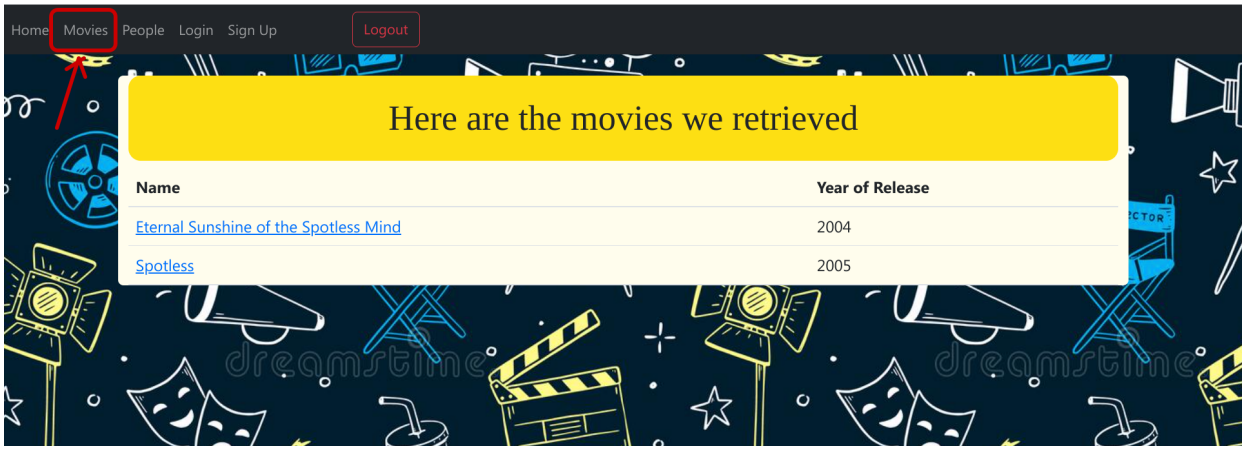
Then she can go on to search for movies/celebrities or shows in our database, let's say she wants to search for a movie . She clicks on the search button and we take her to the movie search page :

Clementine is looking for a movie , she does not remember the full name, but knows that the name contains : "Spotless" and was release some where in first decade of the century . She puts in all she knows, leaving other options blank.



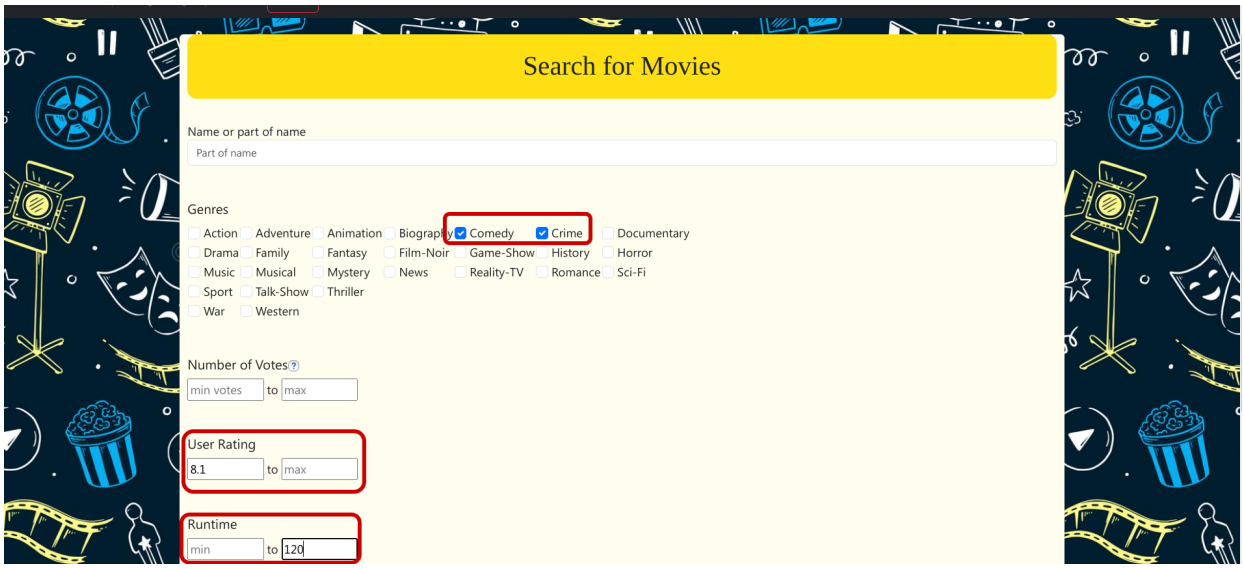
and clicks "Search"

Cinebase searches for movies set by option provided by her and displays her all the results in order sorted by popularity as well as taking into account user preferences from her past ratings :

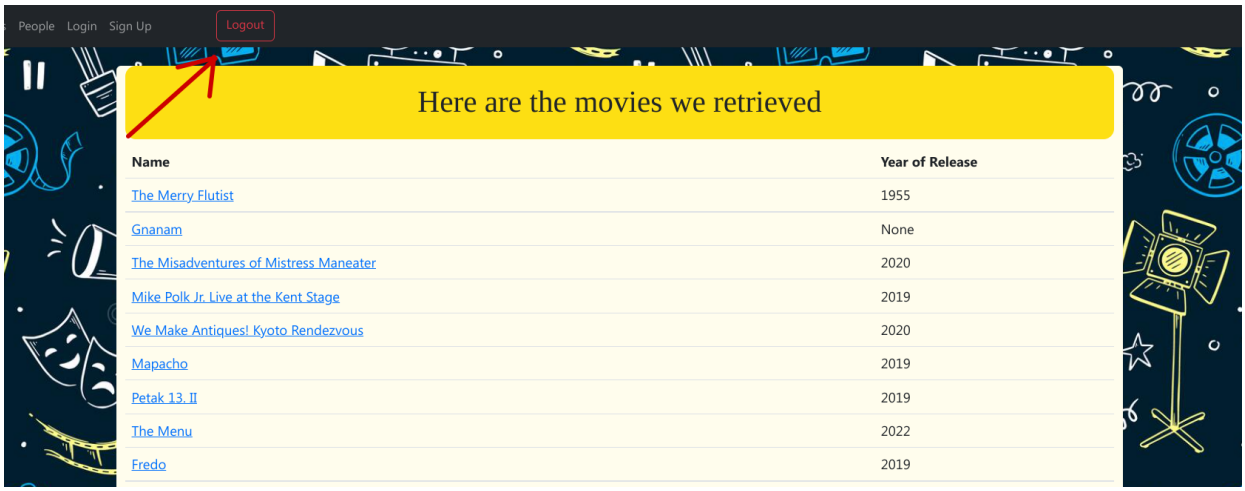


Clementine found the movie she was looking for ! She clicks on the name that takes her to a page showing here all the relevant details fro the movie :

She wants to make another search, this time she wants to know all the crime or thriller movies with runtime less than 120 mins. She clicks on the Movies button on the navigation bar at top and goes back to the search page and types her choice again :



and gets her results.
She is satisfied and done with cinebase and clicks on Logout button to logout



She is taken back to the home page. Similarly we have pages for People and shows search as well. :

If Clementine wants to do some generic searches like seeing the top 50 movies of all times, or the upcoming movies , she can access our **Quick Links** , on the homepage, these searched do not require any user information and don't need users to login :