Colin Zejda (worked alone)
Professor Ahmed Ezzat
CIS 008 Advanced Python
16 May 2022

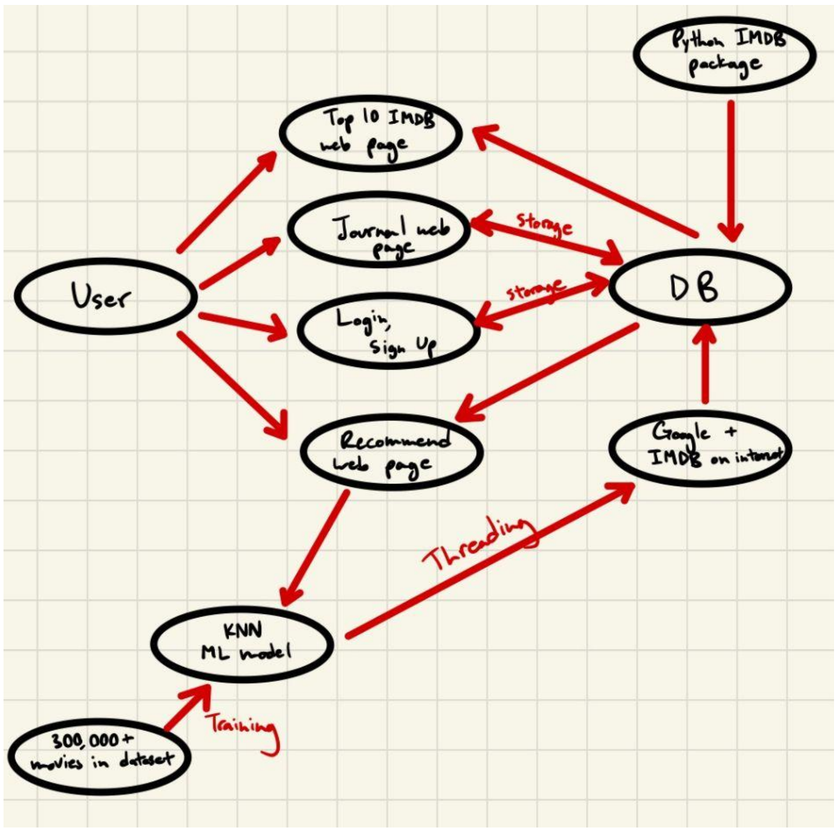
Final Project Report

Describe the problem the project is solving.

- People love watching movies! But sometimes they don't know what they want to watch next. My website, called MovieREC, allows users that have signed in to get movie recommendations for movies they already know and love.
- In addition, if they haven't seen any movies yet, the website has a page that displays the top 10 IMDB movies today, to give them a starting point.
- Lastly, the site includes a page that gives the user the ability to take notes and keep a movie journal (saved between sessions when logging on/off).

Project architecture

- My project is made up of three main things: the HTML pages, the database, and the python files (machine learning model, web-scraper, blueprints for the HTML pages). I go into more detail below.



-
- HTML pages

- For the login and sign_up pages
    - We accept an email and a password in a text box, and when the submit button is pressed, a POST request is sent to the server (configured locally by a create_app function in the __init__.py file).
    - All entered info is check against the database (password length, email uniqueness, and email validity (must end in @gmail.com)).
- Once a user has successfully logged in
    - They'll go straight to a landing page called movie recommender. This is the home page, which the user can use to get new movie recommendations, preview the movie posters, and find links to the IMDB and Google pages of a particular recommended movie.
    - If they navigate over to the upper left corner, they'll find a menu with two other pages.
- User journal page
    - This is yet another HTML page, and it's quite simple. The user can type in a text box, and save this text box with some notes about movies they liked, movies they want to watch, etc. It can even contain gibberish for all we care– it's just used for journaling.
    - If they don't want a note or they've ticked off a movie on their watch list, they can delete one of their notes.
    - A bonus feature is that whenever the user enters a note, the exact date which the user enters their journal note will become part of the journal.
- IMDB top 5 page
    - If the user doesn't know where to start their movie journey or if they're fresh out of ideas, they can navigate over to this page and check out the hottest movies on IMDB right now. Like the recommendations page, there are links and movie posters provided for the user's benefit.
- When logging out
    - The user is simply redirected back to the Login page.
- I didn't code much CSS for these HTML files
    - Instead, I imported a bootstrap template and a bootstrap theme in order to give the site some professionalism. I did this because CSS wasn't the focus of this project, and I didn't want to waste very much time on this.

- Database
    - Purpose: stores all information efficiently. This includes the user's name, password, email, journal notes, dates for each journal note, and all recommended movies.
    - We communicate with the DB using methods such as query.first(), add(), delete() and commit().

        - We can easily access the information inside our database without querying it when we use Jinja commands inside our HTML files, which allows us to loop through the tables in our DB and display their contents.
- Other python files
    - KNN machine learning model– we use this file to calculate distances between the user's movie keyword entry and all the movies in the database and return a list of the 10 closest movies.
        - However, on our site, we only display 5 of these movies.
    - We have a file that uses BS4, beautiful soup, to scrape the google search page for links and posters that I can display on my website.
        - The code I use closely matches that of what we learned in class, which I am quite pleased about.
    - Blueprints for our HTML files. I made this project using Flask, so Python was inevitably used in the creation/management of my web pages. In my __init__.py, auth.py, and views.py files, I have blueprints for all the routes in my site, the login/logout logic, the processing of POST and GET requests, and communication with a JS file for the deletion of journal entries.
- This concludes the architecture of my project.
    - Now, I will go into depth into each piece of architecture and explain what the pieces of code do.

Design and Implementation.

- Most of the code was written in Python, but there was some HTML, Jinja, and Javascript as well.
- All the code is stored in a folder called 'website,' inside which we have all our HTML templates and our python files.
    - All of it is united using Flask, which is a lightweight python framework that makes the creation of websites incredibly easy.
- All web pages
    - Were made with HTML.
    - They are decorated with a free bootstrap template that I found online.
    - Then, to display elements from the database, I used a language called Jinja. Jinja supports Python-like commands inside an HTML document, allowing me to loop through and display the movies. I use for loops inside the {% %} delimiters and used the {{ }} delimiter to display data from the tables.
    - For the login and sign-up pages, I would accept the POST request with the user's entered info. Then i would query the database table with the user info and check all users to make sure that the email+correct password was in the DB. If it was, I'd flash the user a green message and log them in. If some part of their login/sign-up attempt was unsatisfactory, I'd flash a small red error message

explaining what happened (too short a password, duplicate email, or wrong password), and bar them from having access to the site.

- How were the web pages organized and designed?
    - The HTML pages:
        - Login, Sign up, Recommend-movies, Journal and imdb_top_10 are the five pages I have on this site.
        - They were initialized using a base.html template, which I reuse as the base for all the other pages.
    - The login, signup, and logout logic are handled by the auth.py and views.py files, using decorated functions called recommend(), journal() and top_10_imdb. These functions handle the GET and POST requests, update/commit changes to the database, and will render changes in real time for instantaneous updates (no redirects, no lags). This is the beauty of using Flask.
    - Additonally, these two files contained blueprints for our HTML web pages, outlining what was to be done with each route as the user interacted with the site.
        - Views.py is the largest file amongst my python files, containing the blueprints and DB interaction-logic for the three main web pages on the MovieREC site.
- How about the creation of the DB?
    - This is handled in our __init__.py and models.py files.
    - The __init__.py file was responsible for creating our DB using SQL-alchemy, as well as creating our application. This part was made super easy (only 30 lines of code) because I used a lightweight Python framework called Flask for building the website.
    - models.py, on the other hand, was where I declared all my tables. I had 1 class per table (Recommended, Notes, IMDB_top_10, and USER).
        - I established relationships between these tables using the .relationship() method.
        - Because each user has their own journal entries, recommended movies, and login account information, my USER table was at the top of the database table hierarchy (it was a one-to-many relationship).
        - The entire database is stored in a file called database.db.
    - What did I store in my tables?
        - User_id, email, password, first name, all journal entries, journal entry dates, and all recommended movies
        - I didn’t need to define the primary, logical, or foreign keys because SQL-alchemy handles all of this when querying and updating the DB.
- What about the rest of the implementation?
    - Most of the back-end was handled in my python files.

        - These were created to provide functions that I could import and use, such as the get_movie_recommendation() and get_top_10_imdb() functions.
        - The KNN machine learning model that was used to find the closest movies is stored in movieREC_get_data+train_Model.py, which is kept separate from the function that can use the model (stored in another file). This was done on purpose, so we didn't end up having to redundantly train the model every time we used it.
            - This KNN machine learning model was adapted from a tutorial you gave us at the beginning of the year. It accepts a keyword (a movie name), compiles a list of matching movies in the 300,000+ movie dataset, retrieves the first one, feeds that movie to the KNN model, and we receive a data frame with 10 movie titles and how close they were (on a scale from 0 to 1) to the keyword entered.
        - Then I turned around and used these movie titles with beautiful soup and urllib to scrape the Google search page. The goal was to get the movie posters and clickable https links from the google search page.
            - The code begins by opening the regular google search page, then looking for the first <a> tag and retrieving the href from it. Then i went further and scraped the first <img> tag for its src parameter and got a link for the image of the movie poster. Then I stored them into the database until I needed them. When I did need them again, I simply tacked the links onto my HTML page hrefs and image src's, allowing the user to have easy access to the movie on the web.

<u>Any technical issues you encountered during the project</u>

- I had a hard time displaying info from python file to HTML pages.
    - Using a tutorial from TechWithTim on YouTube, I found a convenient way to display text (the recommended movies from my KNN model) onto the website.
    - I ended up importing my function into my views.py file
        - Then adding an entry into my database from my views.py file
        - And in my HTML files, i used Jinja to loop through entries in the database and display them onto the page in a neat manner.
        - I'm sure there was a more optimal way to display info onto the web page, but I could only make the text display work if it was through the database.
- Making the KNN machine learning model work was tough.
    - Professor, I attended your office hours to consult you on why my KNN model kept having an index out-of-range error, and your advice helped me a little bit. I went on Stack Overflow and further found out that I was excluding too many data entries in the movies.csv file that I trained my model on, and if the trained KNN

    model didn't have access to all the possible movies in the dataset, the created dataframe would have an out-of-index error.
  - I won't lie, this drove me crazy for a long time, so I'm really glad I got it figured out.
- Using the database was a challenge, since I had never seen anything SQL related before.
  - I had many issues using my SQL-alchemy database during the creation of this website. I constantly ran into OperationalError issues, and it took me a week to figure out that my database was corrupted because I cannot add new tables and change the structure of the database whilst the website was up and running.
  - Once I deleted the database file and re-ran my project (which created a new DB), everything started working again.
- Using beautiful soup made our site incredibly slow. Each time the user asked for movie recommendations, the site would take about 45-50 seconds in order to display all the information I wanted (google search link, imdb link, movie poster image). The same goes for the current top 10 IMDB movies.
  - I resolved this issue by using threads to speed everything up. Running several processes concurrently definitely helped bring user wait times down.
  - I simply imported the threading module, ran all the web scraping concurrently, stored it in an array, then passed that array into my views.py file so I could store this valuable information into my database.
- Not a technical issue, but a team-related one:
  - My team and I had a clear line of communication via Whatsapp, but they never got any work done. They would make promises to code a certain aspect of the site or watch some videos and share what they learned, but they never shared/did anything for the project.
  - After 2 months of this happening, In the end, I did the entire project myself.

Results/Conclusion of the project

- I want to say that this project was a success. I learned a great deal about coding websites in Python, researching what's necessary and what's not, and understanding how everything is structured.
- I'm pretty happy that I used concepts that we learned in class on this project, such as decorators, JSON, GET and POST requests, and REGEX (regular expressions).
- On top of all this, the website works!
- However, I was disappointed that my teammates wouldn't work on this project with me. If I had two other people actively working on this, my project could have been so much more than it is right now. Alas, I've learned a valuable lesson, and it is that I cannot always count on those that I am assigned to work with. I've also come to realize that doing all the work is HARD, but the solitude gave me the ability to take a different

direction from what was initially proposed by the group and I ended up adding a few bells and whistles that I’m quite proud of.

- My dad explained that he has also experienced similar bad-teammate situations at Synopsys and AWS, and that doing such a project by myself would only strengthen my abilities as a programmer and as a person. I very much agree.