CFG Nanodegree Project Report

Films For You: A Movie Recommendation App

CFG Nanodegree Autumn Cohort 2021

Saamiya Yousuf, Luciana Marini, Lizzie Wren, Lowena Hull, Annie Ding, Maebh Ni Ghuairim

Contents

1 Introduction

Our group project Film For You, is a movie recommendation app that takes in a user's input in order to recommend a movie of their choice. The construction of this app involves users creating a registration account with Film For You, a login to access their newly formed account as well as inputting specific details to output a movie based on their preferences . With this report, we will explore our group's decision-making process in choosing the type of web framework that was used to build the app, the movie API, called TMDB was used to call functions for the movies to watch as well as the database to store a user's details. As a group, we chose to pursue an idea of a movie recommendation app as our interests heavily lie with film and cinema.

2 Background

To begin, our app involves user's creating an account and having a watchlist based on the movies they have selected through their search. They can curate their list based on genre, age rating, minimum and maximum length of film in minutes, keywords associated with the genre which will output their results 3, 5 or 10 at a time. Our app would call upon the API when the appropriate input was given and then sorting the results so the relevant info could be used in the front end. Furthermore, the project was idealised with having a database for both authenticating users, but also storing user's data that could be useful and functional for the app, as movies already watched by the user (so it does not keep recommending it repeatedly) and a watchlist, for movies that the user can store within the app as a reminder of movies the user would like to watch in a future date.

3 Specifications and Design

Requirements

The project requirements were discussed and analysed by the team in order to determine what are the key technical and non-technical requirements for our app to run both successfully and efficiently.

Technical Requirements

- The app must have a registration page for users to create an account with Film For You
- The app must have a login page to access user's newly formed account
- The app must have a watchlist to store user's selected movies saved
- The app must store user's authentication details and watchlist in a database
- The app must allow users to input their preferences on what they want to watch and output their choices
- The app must call on the TMDB API using functions from the user, formatting correctly
- The app must call the functions to verify whether the input was valid and prompt the user to correct if not.

Non-technical Requirements

- **Documentation**
    - Well documented code within the python files for developers
    - Ensuring documented software engineer planning processes
- **Testing**
    - Extensive unit testing
    - Handle any unexpected errors early on
- **Security**
    - Data collected from users must be stored securely with only necessary data stored to the database

Architecture

Through careful analysis and consideration of the different options regarding the architectural choice our app, in relation to the key functionality and requirements, it was decided that the best type of architectural design for our project would be the user-server model. Our project would represent a web application that will be access by our users through the internet, but also will need access to a database to store their user authentication details and watchlist, as well as send requests to the API. The design is based on the three main components. Firstly, the user side that will be interacting with the Flask web app that would be responsible for presenting the front end using Jinja2 as HTML templates as well as sending the requests to the API and managing calls to the database, and finally the database component itself that will communicate with the database manager in the app, all shown in figure 1.

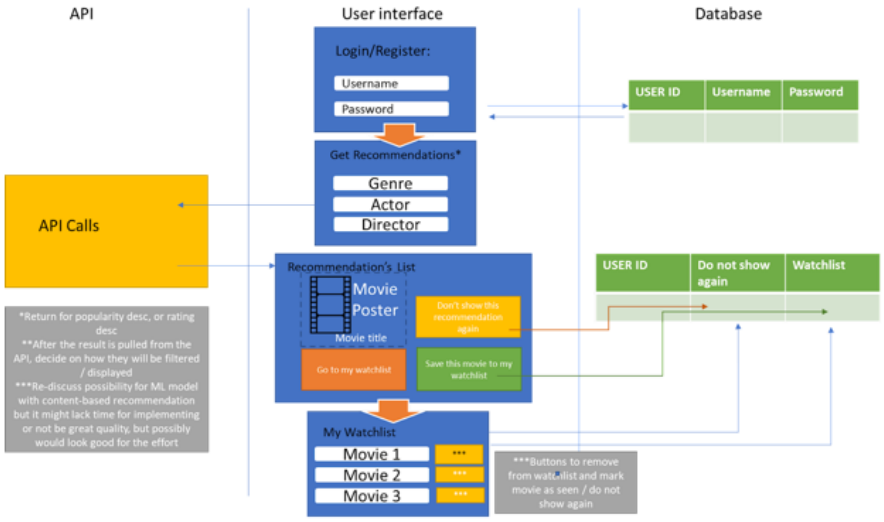

Figure 1. Shows relationship between front-end, API and database

User Interface Design

The aim of our project in terms of the front end was to design a simple yet sophisticated user interface that was easy to use, and obtain a recommendation as fast as possible, in order to begin their movie viewing. We wanted to create an app that would make movie night easier for organisers, film night hosts, family night in, film buffs, and anyone who likes watching films. It was important that the app maximised the potential recommendations, whilst conforming to the user's requirements on limitations. Figure 2 illustrates our groups mood board for our project, revealing our apps look to be modern with dark-mode undertones, reflecting the cinema feel one would experience when the lights are dimmed. A brand book and mood board guided the look and feel for the app.

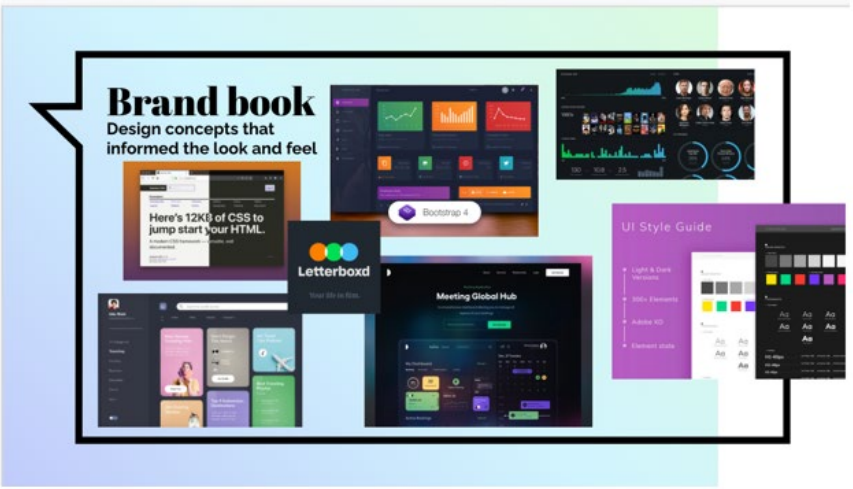

Figure 2. our groups brand book for our movie app

Database Design

The database was designed in MySQL workbench. The initial design consisted of a main table called “users”, where the username, email and password would be stored. A second table was created called “Movies”, where movies and associated details would be stored, in order to be referenced by the third table, “Movies_watched”, which would have a many-to-many relationship with tables “users” and “movies”. The table “users” is used for authentication purposes to store a user’s password safely, in doing so, it uses the AES_encryption which is inbuilt in MySQL database. Whenever a user logs in to the app, the users username is compared against the password stored on the database and defines if the authentication as either successful or unsuccessful.

Furthermore, the idea behind the tables “movies” and “movies_already_watched” was that when a user selects a movie already watched, it should not be displayed in future list of recommendations, so when the user logs in, all movies watched by the user can be fetched from the database and stores on that session’s info, so whenever a recommendation is requested, the list of movies to be displayed would be compared against the movies already watched information and decide which movies to be displayed.

Project Structure

Figure3. demonstrates our project structure, containing five folders within our Github repository. Outside of the file has our main FilmForYou.py file that is what we run, either in our command prompt on our computer or the terminal in the IDE. Furthermore, we also have our forms.py and our post_info.py that contains the post information of the team members in our group n. Our templates and static folder stores our HTML files and CSS file respectively, that provides the basic functionality of our app. These folders are required for Jinja2 to load the HTML files and for the Flask app to render the templates, therefore it is purposely structured in this way. All of database files and features are stored within our database folder that contains the configuration and connections, directly calling the SQL database created specifically for our app. Our Recommendation folder stores all of our groups relevant API functions that are in communication with our app, displaying the information extracted from the database and presenting the output to our user in the front-end. Finally, in our testing folder includes our API and database testing which we will explore in detail further down in this report. Our groups README file on Github displays a simple overview of our project idea and the relevant files needed to run our FilmForYou.py successfully.

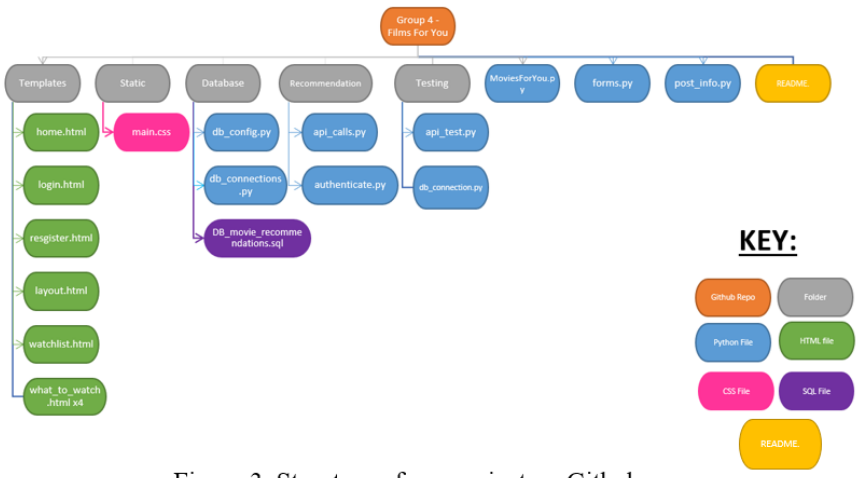

Figure 3. Structure of our project on Github

4 Implementation and Execution

Development Approach

Our project was run using the Agile methodology, where each member took on the Developer role as that was where our most time was spent on. However, during the meetings, everyone had a chance to discuss their portion of the work, the achievements and challenges and where other members provided feedback in order to improve.

Team Member Roles

Throughout the project, the members were divided into three main teams: Front-end, API, and Database. As we had equal number of members for our project, we split the group so two people could be a part of a team, taking on the workload equally and fairly. The teams was allocated by what part of the project they were most interested in working on, where it is creating and designing the front-end (Saamiya, Annie), choosing the best movie API that worked well with our app (Lizzie, Lowena) or creating a database with the requirements we had set (Luciana, Maebh). Once the teams were allocated, we had bi-weekly meetings in order to communicate our progress on what we had done, what help we needed and what were the next steps to take. Furthermore, we ended each meeting with the date and time for our next meeting and whether we needed to include our instructor to attend also. Moreover, our instructor was in constant contact with individual members of the group or as a collective, providing essential advice and direction throughout our project. Towards the end of the project, we added in a testing team that were involved with running functional tests against the API and database and seeing whether it passes or fails (Lizzie, Maebh).

Flask/Jinja2 Implementation

Flask was essential in building our app as through research, it was shown compared to other web frameworks suitable for python, such as Django and CherryPy, Flask stood out as best as it was simple to implement and use, flexible and included tools to customise its behaviour. It was used to create our routes for our app and hosted other functionality as well. In addition to importing Flask into our IDE, Jinja2 was automatically imported which was used to construct our HTML pages.

The Flask-WTF library was used to create specific forms called FlaskForm in order to create classes that inherit from this class to create our own custom forms. WTForms was used in conjunction with Flask-WTF to create form fields such as StringField, SelectField, and even a SubmitField that generated a submit link. It also works to easily collect submitted data and validate it on the frontend page all at once, requiring less code from our end and allowing us to use it to pass variables around in the flask app backend.

API Implementation

Regarding our API implementation, we researched what different film API existed, what data they had, whether you could filter results when making the call or had the information to do so afterwards, what they used to make recommendations on, what features they had in the free versions. Once the API was chosen, we had to figure out the correct syntax to make the call as the documentation wasn't always very clear. Requests was used to make the call to the API to get the data on films and TMDB API was used to get up to date information on films for the criteria needed.

An added bonus to our app as part of our "nice-to-have", was adding in keywords in order to refine the search process for our users. The API team researched the required input to the API (string of keyword IDs separated by a "|". Furthermore, we get input and change it into the required output,

as this requires searching up the associated keyword ID for each inputted keyword using another functionality of the API.

Database Implementation

For our group's database implementation, our code to connect the database to our app was written in PyCharm. After a successful connection, we wrote functions that would query the database to;

1) add new users
2) authenticate users
3) add movies/movies already watched to the respective tables
4) retrieve all movies already watched by user.

These functions were derived from two main functions query_db() and read_db(), which create a new connection and would query the database in order to accomplish its function. The main difference between these two functions is that read_db() does not have a method to commit() any changes to the database and it returns the result of reading the database, whereas query_db() is meant to CRUD other functions besides read (create, update and delete). The function read_db() also takes as parameter "fetchall" as a boolean, so it fetches all results in case this parameter is set to true (for example, to retrieve all lines of result of the query) or, case the parameter is set to false, would fetch only one (fetchone), the first result from the database. Both query_db() and read_db() would user the function db_connection() to stablish connection to the database when required.

Tools and Libraries

Front-end

- Flask/Jinja2 – flask was used as our web-framework for its durability for small-scale projects and uses Jinja2 as a templating engine that's fast and expressive in its implementation.
- Flask-WTF was used in order validate our login/registration and it was compatible with our web-framework. It was also easy to implement in our code, creating classes that could be used for testing.

API

- TMDB API and Requests - Requests was used to make the call to the API to get the data on films and TMDB API was used to get up to date information on films for the criteria needed

Database

- MySQL connector/MySQL connector python – was the preferred choice instead of Flask_MySQL connector, as the methods and use are more straightforward to use.

Implementation Achievements and Challenges

Achievements

Our project signified many achievements throughout its journey. The process of integrating the API and database with the front-end posed the most challenges yet was deemed our biggest success once our app was fully functioning. Similarly, creating the foundation of our app using Flask and Jinja2, which led to further contribution from other team members, developed our project in becoming a movie recommendation app for our users. Furthermore, we managed to add features and link them to the database and API to ensure functions e.g., buttons all worked as expected. Another success came from creating our watchlist which was part of our groups "nice-to-have" but ended being part of our finished product. In regard to our database, we were able to store the user's watchlist that can be added/removed also. On the front-end, we conducted successful handover as we were able to build the code developed in the last task. This led to improved collaboration and transparency between the team members.

Challenges

A challenge that we as a group came across was the unfamiliarity of using Github that required multiple contributions from others. Many branches were created that demonstrated the work in progress that we as a group contributed too yet faced the challenges of adding the completed version on the main branch. This coupled with the fear of overriding another team members work caused delays leading up to the final weeks This however was rectified by asking our instructor on how to upload our folders and python files without conflict and only uploading our finalised code to the main branch.

Secondly, in regard with working with the TMDB API, there were a few obscure errors discovered when executing a specific call, where the functions were mostly correct, yet it was not exactly running as it was supposed to. An example of this was the part that stops films on your watchlist being recommended again, worked if the films were spread out in the results but would only pick up the first one if they were consecutive. This was hard to spot before there were quite a few films in a user's watchlist.

Another challenge was it was difficult choosing the right syntax to use when descriptions contained both " and ' so were throwing errors, so we had to be consistent with our quote marks. The implementation of the database was straightforward, yet the main challenge was the encryption / decryption process. It was initially meant to be coded in python by using the cryptography library. However, once the password was converted into binary object and was passed into the query to be added to the database, it kept causing failures. Therefore, a decision was made by the database team to change the encryption process to MySQL's built-in AES encryption and decryption methods.

5 Testing and Evaluation

Testing Strategy

Our strategy was to test extensively and to test during development, and this was possible for standard unit tests that we could use on our API calls. However, it became evident that the team would need to spend more time mastering mock testing to apply it to the database connection.

Functional and User Testing

The aim of testing was twofold - firstly, to ensure that our code worked, and that there were no unnoticed bugs present. Secondly, the knowledge that our code was going to be tested made us focus on how to write code which is more easily tested and of high quality.

Unit testing was a must for our project. We knew that constant tests would allow us to work in a more Agile manner, as it allowed us to make changes more easily, if necessary. It ensured that we produced high quality code as we fixed any issues or bugs before we submitted our app. It also saved us precious time as we were made aware of issues sooner rather than later. Lastly, unit tests also serve as a form of documentation as they demonstrate how the functions are meant to work to each other, and to the markers of our app. We also used mocking in relation to database calls, as it was a safer way to test our code, as we were not relying on external databases.

The achievement for our project was when unit testing our application, we ensured that all our functions work as they're intended to, and as such we have a working movie recommendation engine, demonstrated by Figure 4. However, the hurdles we faced was mastering the ability to mock some of our database calls to test our functions. Whilst several days were spent trying to understand mocking with small success, we reached out to our instructor for guidance. Thankfully, with that help, we were able to complete the testing of our database calls and ensure we didn't have any bugs. Whilst this was certainly a challenge, we feel we have a far better grasp of mocking now and consider that to be an achievement.

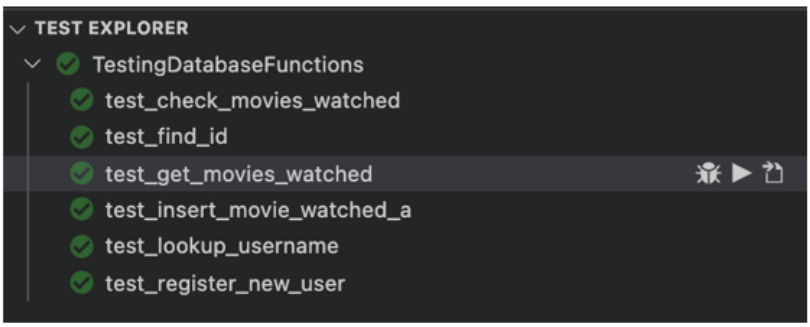

Figure 4. Successful testing of our database

In regard to testing our API, we undertook unit testing the API calling section, ensuring each function could be tested individually and produce the expected result. Furthermore, some exploratory testing's were also done once the application was integrated and working. This was to check the functions behaved as expected and produced the right response as well as checking all the different parts connected together correctly and check if any of the user interface added any

problems to the other parts of our project. An achievement of ours were all the API calling functions were fairly self-contained which made testing them individually fairly simple. Yet, the challenges were finding all the possible ways to combine things when doing exploratory testing.

System Limitations

As with most projects on a tight schedule, more testing is always possible, and our project is no exception. If we had more time to spend on testing, all our functions could be tested. Also, with more use of the app, we would have a better idea of what is useful, what's not needed and what could be improved. However, with the tests we have done, we have ensured our code is of a high standard and passes the parameters we have set.

6 Conclusion

This project resulted in the successful development of a movie recommendation app that allows users to register and login their account, input their choices with the questions asked as well as keyword choices to pull specific types of films. The project made good use of HTML syntax used in Python, as well as CSS that gave the overall stylistic look to our front end. The use of the TMDB API has allowed a variety of films to be explored by the users that could push a number of results at a time. The database is a minor but important part of the app development process, and as a group are pleased with the success of creating a fully functional database. Overall, our movie recommendation app was a success and enabled users to enjoy discovering new films for any occasion.

Appendix of our images

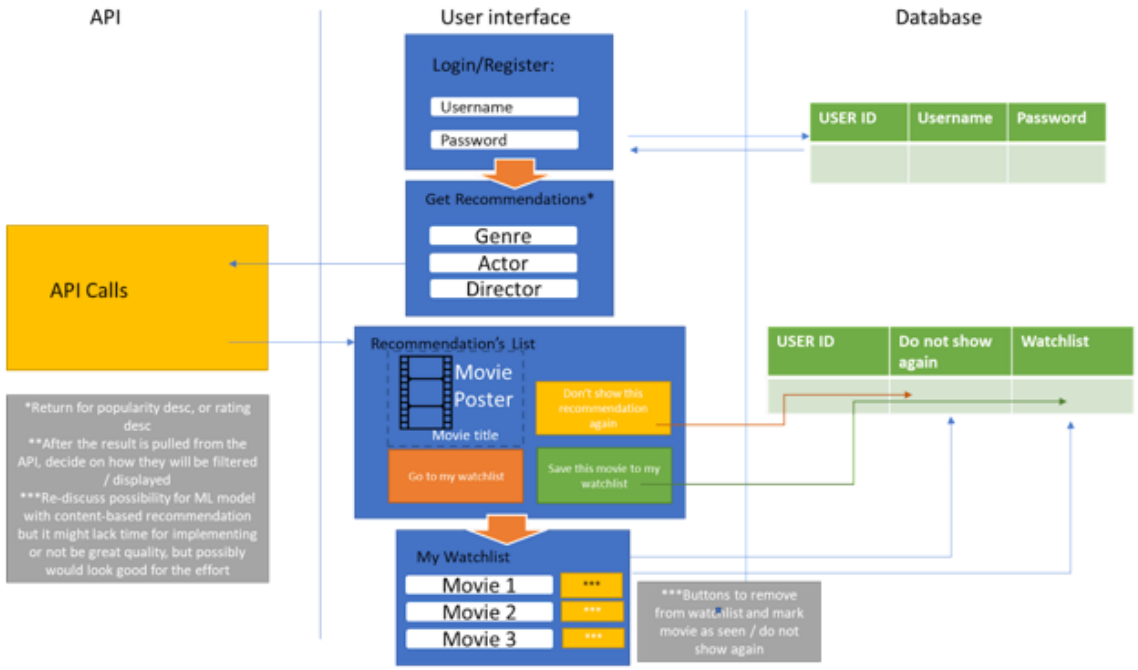

Figure 1. Shows relationship between front-end, API and database

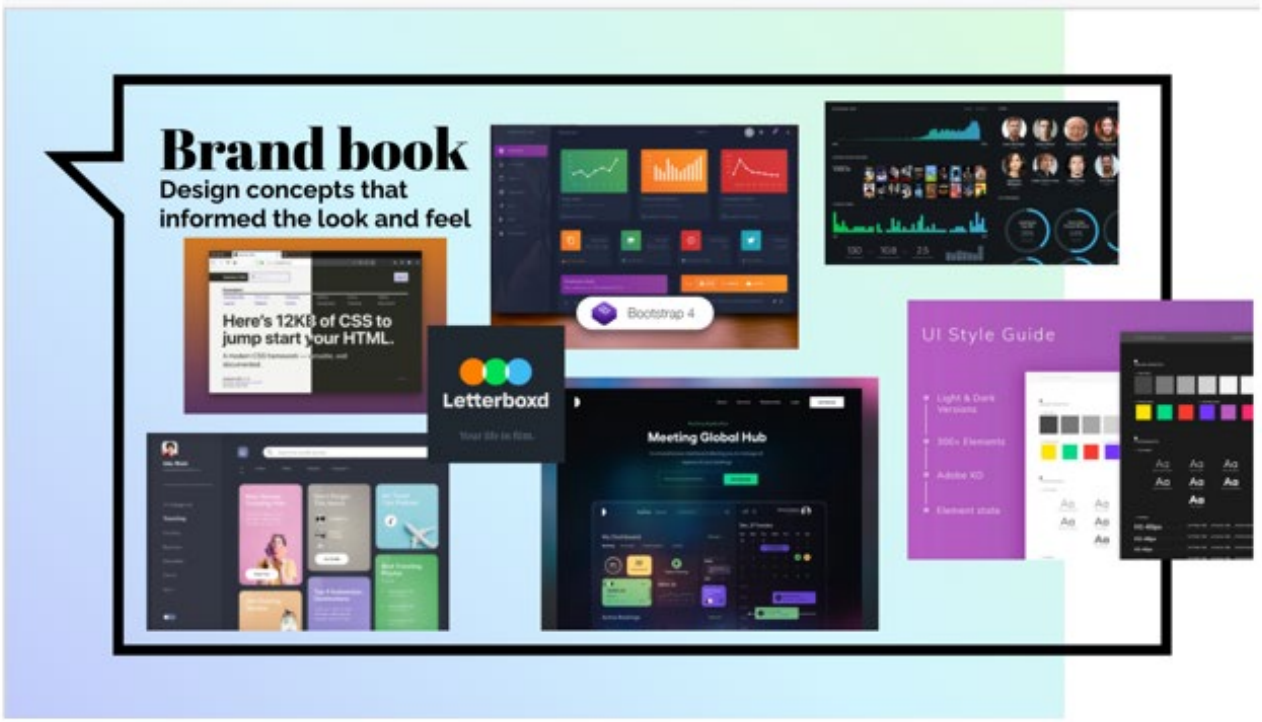

Figure 2. our groups brand book for our movie app

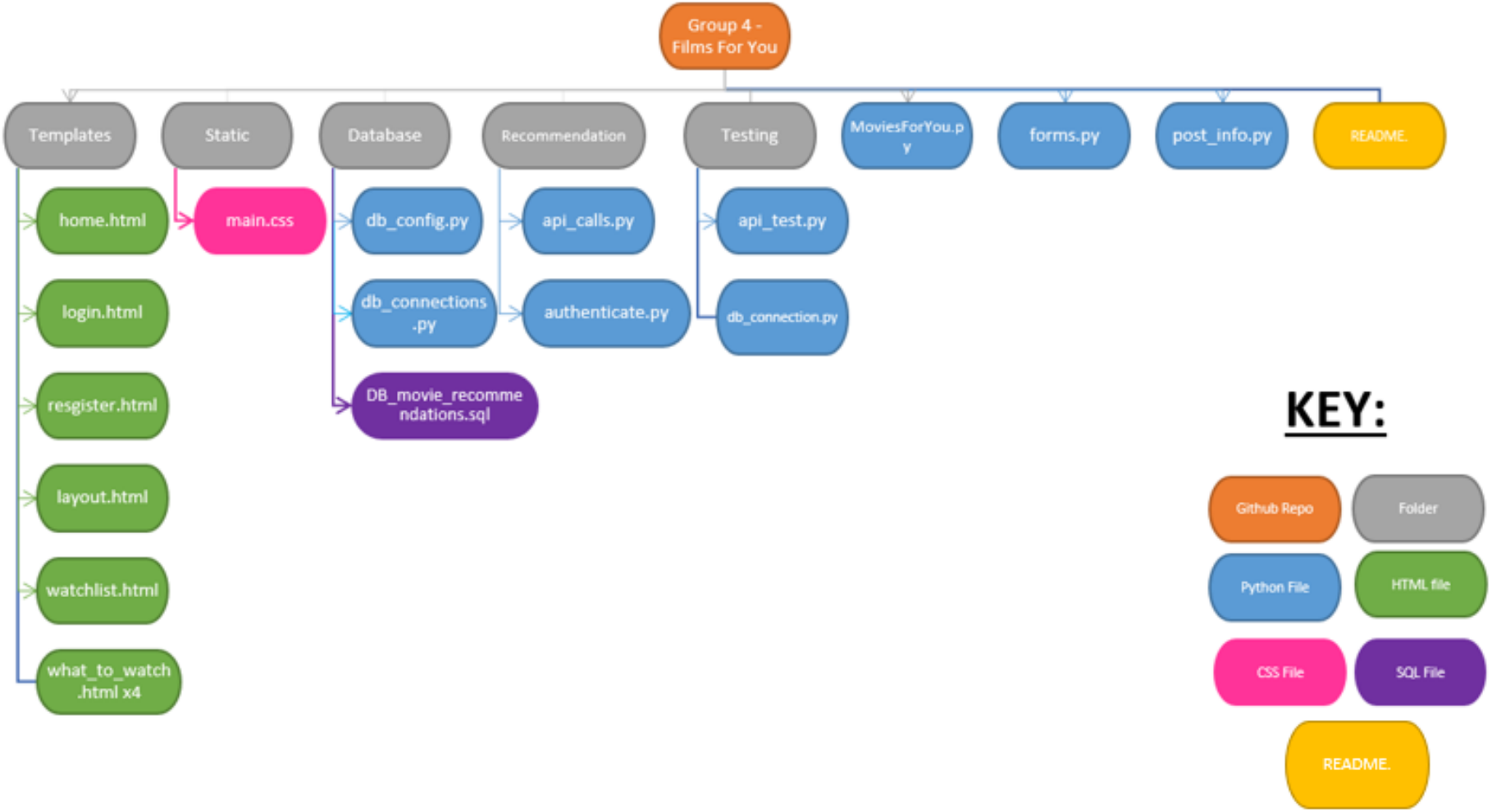

Figure 3. Structure of our project on Github

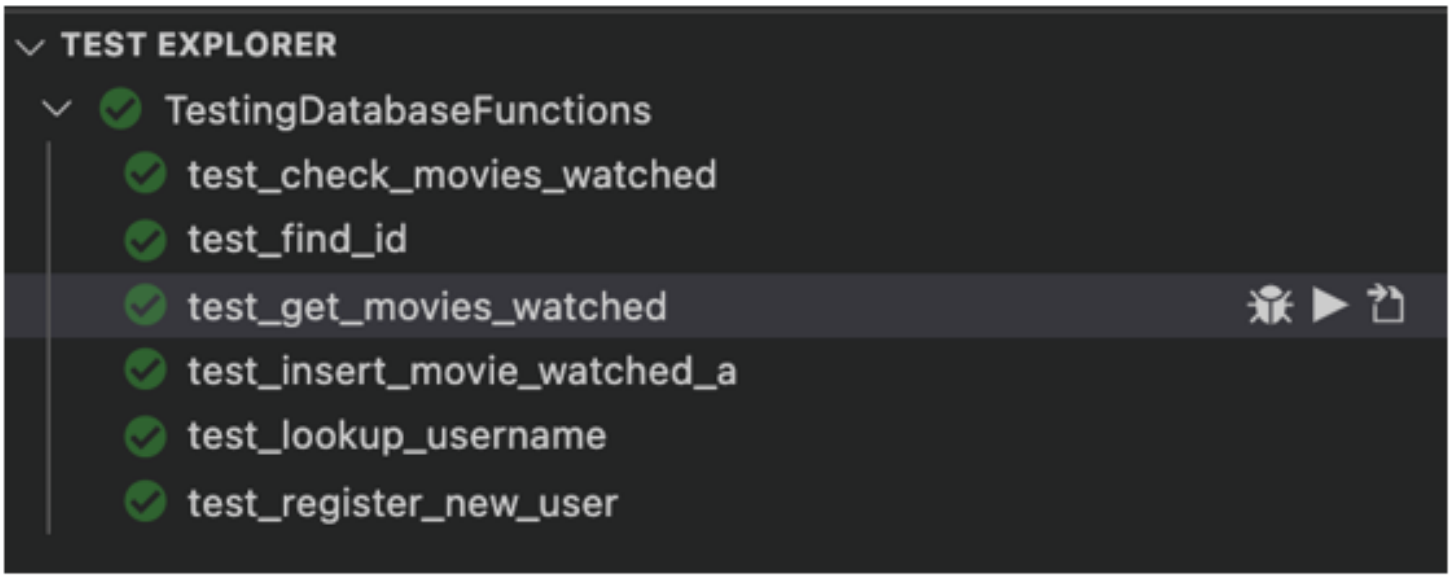

Figure 4. Successful testing of our database