FilmNow

INTRODUCTION:

Aims and Objectives:

The aim of our project was to create a full stack-application which helps users find movies that match their interests. The main idea of our application FilmNow is for the user to enter a movie title of their choice and the application will take that and return both the information on that movie and three recommendations based on it.

The main advantage of FilmNow is that it retrieves movie data from The Movie Database (TMDB) which is an open source, community built database for TV and film. This means that the database is updated daily and the users can enjoy up to date information on movies they have an interest in.

Our main objectives for FilmNow were:

- Be user friendly - consider accessibility, readability, and inclusivity.
- Use a database that provides a plethora of content - information on movies and recommendations.
- Seamless integration of information from back-end and front-end.

BACKGROUND:

When a user clicks on the film recommendation app, a login form will pop up and they will be able to login if they already have an account. The login form will ask for their email address and password. If they do not have an account, they can click our ‘Sign Up’ button on the top right corner of the page. Once they click it, a form will pop up prompting them to input their email and password in order to create an account. When the account is created, the user will be able to login and the app will direct them to the homepage.

The homepage contains a search engine that asks users to “Enter a movie title for recommendations.” If they search for a movie title that is in our movie API database (The MovieDB), it will take them to a webpage named movie where they will see a container with the film poster, title, overview and release data of the movie they searched for. Underneath this the user will find a container with multiple film recommendations based on the movie they searched for. The containers will include the film poster, title and overview for each of the recommendations.

If the user searches for a movie that is not in our API database, the user will be redirected back to the homepage and a Flask flash message will pop up prompting the user to search for another: "The movie you searched for is not in our database. Please search for a different movie title."

Users are also able to logout of the app when they click the Logout button on the top right of the page.

SPECIFICATIONS AND DESIGN:

Requirements technical and non-technical:

When we first started our project, we originally intended to have an account dashboard, a favourite movie list and a chatbot that will help users navigate the website, alongside what our app currently has.

Our initial requirements were:

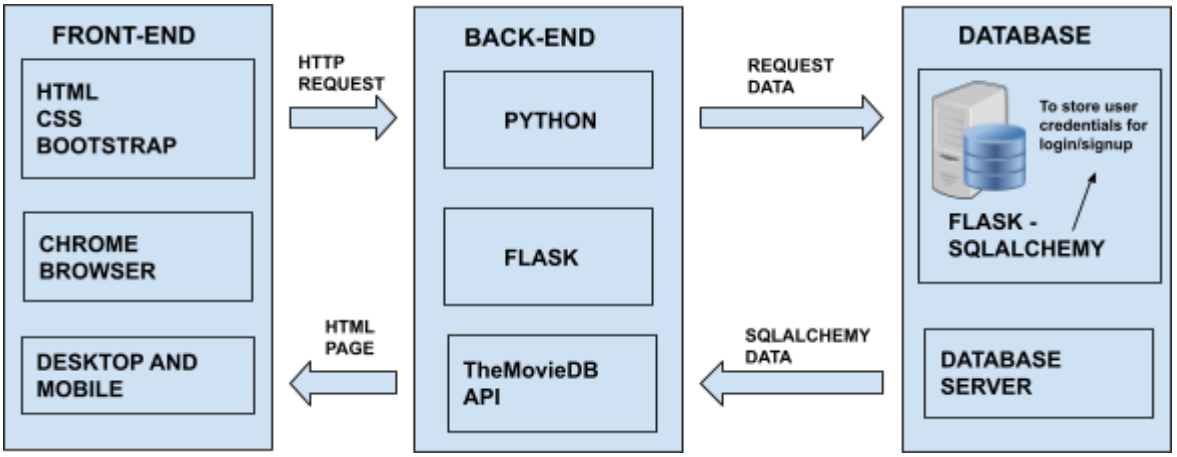
- A signup form for users to create an account with an email address and password.
- A login form for users who have accounts to log into the application.
- An account dashboard that will have the user’s information when they have logged in. The user should also have the option to reset their password and change their email address on the dashboard.
- Appropriate background image which will remain the same across all pages of the application.
- A search engine for the user to input a movie title that they would like to receive recommendations based on. The search engine should be visible on the homepage once a user logs in.
- Recommendations page where a user sees the movie they have searched (with title, overview, release date) and 3 movies recommended to them based on their search.
- Option for a user to create and add movies to their favourites list. This should be visible on their account dashboard.
- A chatbot which will act as a guide for users when they login. It will have a personalised message and will provide options to help the user navigate the application.

Design and architecture:

In order to showcase the structure and design for our system, we drew an architectural diagram. Due to the fact that our film recommendation system is a full stack web application, our diagram has been split into three components: front-end, back-end and database. FilmNow should be able to allow the user to interact with our web app via the front end and back end when they reach our main page. This page prompts the user to input their user

credentials (if they already have an account) or to sign up on the top right side of the page. Once the user inputs their details, you reach the database component where the credentials are automatically stored via Flask-SQLAlchemy. Once the user logs in and searches for which film they would like to view, the functions stored in the back end development will utilise the TMDB API that has been stored. The user will be able to view the film they searched for along with the film recommendations listed below by being redirected to the correct HTML page.

We initially wanted the credentials of our users to be stored in MySQL however we have found it a lot more convenient to store their details via Flask - SQLAlchemy. The advantages of using SQLAlchemy is that we don't need to create a user table on MySQLWorkbench and then connect it to Python. This is a much easier alternative as we create a model in Flask-SQLAlchemy which then translates to tables in a database.



IMPLEMENTATION AND EXECUTION:

Development approach and team member roles:

Each week, one of the team members is a scrum master. Everyday we had a 15-minute daily scrum to review our progress and plan our aims and objectives for the day.
The schedule for scrum master roles was as follows:

22/11/21 - Asia
29/11/21 - Yasmine
6/12/21 - Zodumo
13/12/21 - Arpineh
20/12/21 - Sulekha

This schedule ensured that each team member was given a week to develop and showcase their leadership skills through planning and delegating tasks.
In addition, we created a workspace on Trello to record our development process. The workspace allowed us to allocate tasks, plan, upload (attach) resources, and check on our progress.

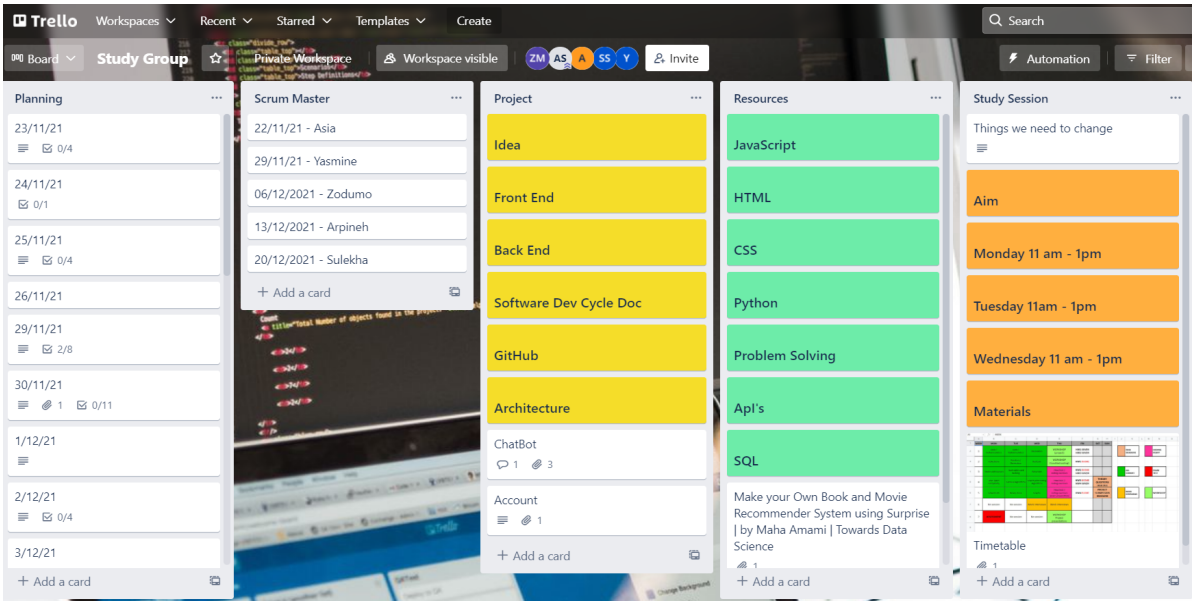


Diagram: Example of our Trello workspace that we used to record our development process. The workspace was organised into different lists that separated our Planning, Scrum Master, Project, Resources, and Study Sessions cards.

Tools and libraries:

- **API** - For our project, we used an API from The Movie Database (TMDB) to retrieve movie data.
- **Database** - We used SQLAlchemy for our database and storing user authentication.
- **IDE** - PyCharm was the IDE used for all code, this included all code to connect the backend to the frontend.
- **Backend Logic** - Python was the coding language used in our project and imported libraries and frameworks such as Mock, Pytest and Flask.
- **Frontend** - The frontend of our project was created using HTML and a CSS framework, Bootstrap.
- **Group communication** - Our main methods of communication and collaboration were Slack for messages/ relaying information and Google Meet for all our meetings/coding sessions.

Agile development:

For our project we adopted some agile methodology elements as we wanted to be in constant collaboration during the development stages. As well as continuously messaging on our Slack group chat, we held daily standup meetings everyday at 11am and sometimes these turned into scrum meetings to discuss our progress in much more detail. Due to people's different commitments and schedules, there were times when we could not have the whole team available in the meetings. To ensure everyone was kept in the loop in regards to our progress, we shared summarising notes in our Slack group chat.

Whilst we did our best to follow the planned schedule, we experienced challenges that meant some tasks had to be completed earlier or later than planned, so code reviews were done daily and we were sure to make note of all changes to our initial plans.

Week 1: Planning
- Consolidate project ideas, aims and objectives.
- Setting project timeline
- Finding and connect to API
- Set-up repository on GitHub
- Started the development of front-end
- Connected to Flask

Week 2: Planning + Development
- Complete the connection to API
- Completed and submitted group week 2 homework
- Completed wireframes for project interface
- Created Python files
- Web link connected to routes
- Connected to database

Week 3: Development
- Split tasks for login system and further connections to API
- Connect API to search engine
- Created login and signup functions
- Connected functions to SQLAlchemy

Week 4: Development + Testing
- Sorted out issues of connecting API to frontend routes
- App started to run
- Completed OOP
- Started unittest and Pytest
- Completed file organisation and cleaning code

Week 5: Testing + Documentation
- Completed mock patching in testing
- Testing functionality of full application
- Hid API key and Database secret key in .env file
- Completing documentation (Report/Readme)

Implementation process:

Challenges

- Git

Using Git to work collaboratively proved to be one of our biggest challenges. Within the first two weeks of the project we were finding it difficult to coordinate branches and work within them, this meant there was confusion with pull requests and it became hard to track branches and files. In addition, we were constantly finding ourselves creating new repositories which added further confusion. As a result, our collaboration was affected as not only could we not work on separate tasks independently, we were left with one person having the most recent files on their machine which left us only with the option of having that person do all the coding during Google Meet meetings.

- Implementing features

We also experienced issues when we tried to implement certain features we wanted for our application as it required us to constantly attempt to reformat already existing code which affected the connectivity between modules. For example, once a team member had coded the login system on a separate machine, we experienced issues with integrating that code to

the already existing code of the project. As a result we lost time due to how long it took to refactor our code in order to make it work cohesively.

As a big part of our project was having a front-end to present our application, we had to install Flask packages which would allow the user to interact with the functionalities of the application. This raised some issues as the changes made when trying to connect the backend (Python Flask) to the frontend meant there were more inputs that relied on each other, especially when trying to respect the rules of the Flask framework.

Changes along the way:
Initially we wanted our application to have multiple features that would allow for maximum user interaction just to keep high standards of being user-friendly, but due to the challenges we faced, we had to make a few changes.

- Chatbot - Initially we wanted to include a chatbot which would act as a guide for users when they first land on the application. The chatbot would give the user enough information for them to easily navigate around the app. However, due to issues we faced when trying to integrate the code of features with existing code, we decided to remove the chat bot as a feature as we did not want to refactor our code any further and were conscious of the time it would take to make it work together.

- User Account Dashboard - Initially we wanted to include the dashboard as a feature that would display the user's account information and present options to reset their password and change their email address. However, we found it challenging to create the functionality of features such as resetting the password and also integrate it with all our existing functionality. We felt as though having a signing up, logging in, and logout functionality was enough so we decided to remove the dashboard as a feature of our application.

TESTING AND EVALUATION:

Testing strategy:
Throughout each stage of our project we ensured that we were manually testing our app so that all of our functionality was working together. As we were making a lot of changes to accommodate all functionality, we were aware that at any point a functionality would be affected so we would have to tend to any regression caused by change. It was helpful to do manual testing parallel to the (coding) development as it helped us identify issues that affected how modules worked together, thus allowing the app to run as expected relative to each stage.

Functional and user testing:
User testing was carried out for all of our functions of the FilmNow application to check how change would affect the usability of it. This was key as we had to make sure that each function was rendering correctly and that the user would be redirected accordingly depending on their input. We adapted exception handling and flash messaging to help the user understand an error when it occurred for example, when a user searches a movie that is not in the our database they will see the message "The movie you've searched is not in our database" will flash above the search engine and the user will remain on the homepage instead of being redirected to a Jinja exceptions template. This allows for the app to remain

user friendly by not displaying information the user would not understand and helps direct their next move.

We aimed to write unit testing for our main functionality in the backend and we concentrated mainly on the business logic. As our functions used API endpoints, mocking was used as it allowed us to test the function without calling the actual API or the tests affecting any external dependencies. Patch was used to create a path to the MovieAPI class where we could then mock the json dictionary that returned from our Get method that retrieves the data for us. This meant we could inject the mocked json into the test and get results from that without affecting the real function. In addition, we used a Pytest fixture to set up our test_client so we could test multiple functions of our project. This was important as we successfully tested functionality that was built using Flask-login and whether these functions were taking in the input we expected the user to enter.

System limitations:
Whilst all the functions of our application work as we expect them to, there are limitations to the amount of testing we were able to complete. Due to challenges that come with the complexity of building a full-stack application with Flask and other packages, we found it difficult to test all of our functionality as our structure (lack of OOP) of the code meant we could not test a few of our functions independently. We would have liked to also be able to delve deeper into testing the json response we got from the API or how the API itself was being posted in the front-end. In addition, we struggled to find a way to test the functionality of SQLAlchemy and if we had more time, we could have been able to adopt more advanced testing methods to test our SQLAlchemy database connection to our application.

CONCLUSION:
Overall, we are very pleased with the final outcome of our FilmNow application. We were able to overcome many obstacles such as Git issues, differing schedules, and time limitations to produce a user-friendly full-stack application.

We were able to stay organised by using our Trello workspace and adopting agile methods that fostered an effective collaborative environment where we were able to bring together all of our ideas. We set out to build an application that solves the problem of not finding film content that matches your interest and our biggest achievement is we were able to achieve that through the FilmNow recommendation system. This is a scalable project with room for future developments that can be implemented such as including the chatbot and dashboard features or even extending to recommending TV shows. Moving forward we will look to improve on our understanding of Git/Github, unit testing and integrating code as we feel as those are the challenges that hindered our ability to include all of the requirements we stated in the planning stages.

To conclude, we have worked well as a team and have been able to apply everyone’s strengths to produce the application we had intended to develop. We are proud of the progress we made as we showed great dedication and were willing to learn new skills to ensure that FilmNow met and exceeded the standards we had set for our project.