2019

CAB432 Assignment 1

CLOUD COMPUTING

PATRIC MARCHANT – N9720316

Contents

Introduction

Movie based playlist creator/image viewer provides users with an easy to use, uncluttered web application to allow users to find images retrieved from Flickr, and create music playlists (Spotify), based on their choice of movie (The Movie DB), Both these actions are completed using the keywords or tags attached to the chosen movie, querying the Flickr and Spotify API's for matches.

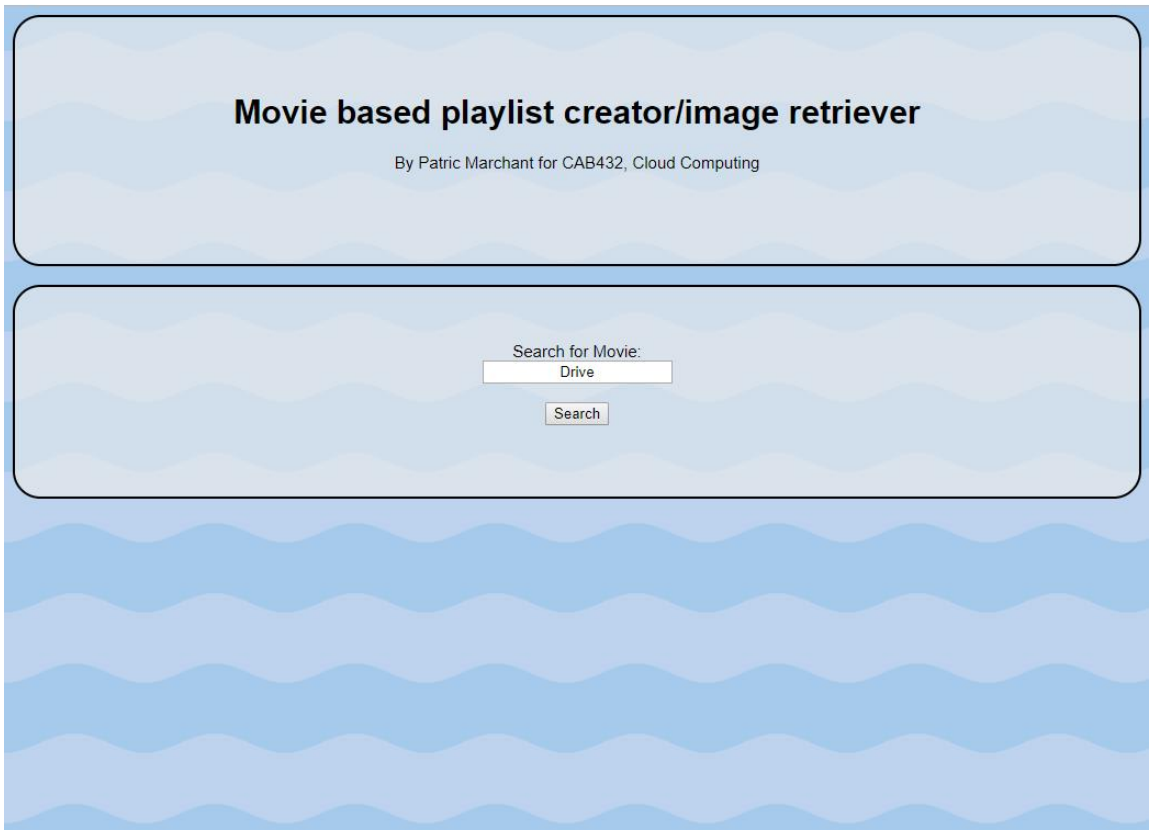
Users can search for movies by inputting a query to search for on the home page and clicking search, which returns a list of matching movies from The Movie DB sorted by popularity, including information such as title, rating, release date, as well as the poster image. From here users may click on a movie, which redirects to a page giving slightly more detail about the movie, including an overview, and genre list, the user then has the option of pressing one of two buttons, titled "Create playlist" and "View related images" respectively. If "Create playlist" is pressed, the user is asked to login to their Spotify account, if successful, the playlist is created, and the user may press the button to return to the index page. If "View related images" is pressed, the user is presented with a page sporting a gallery of images related to keywords from the selected movie.

Use cases/user guide

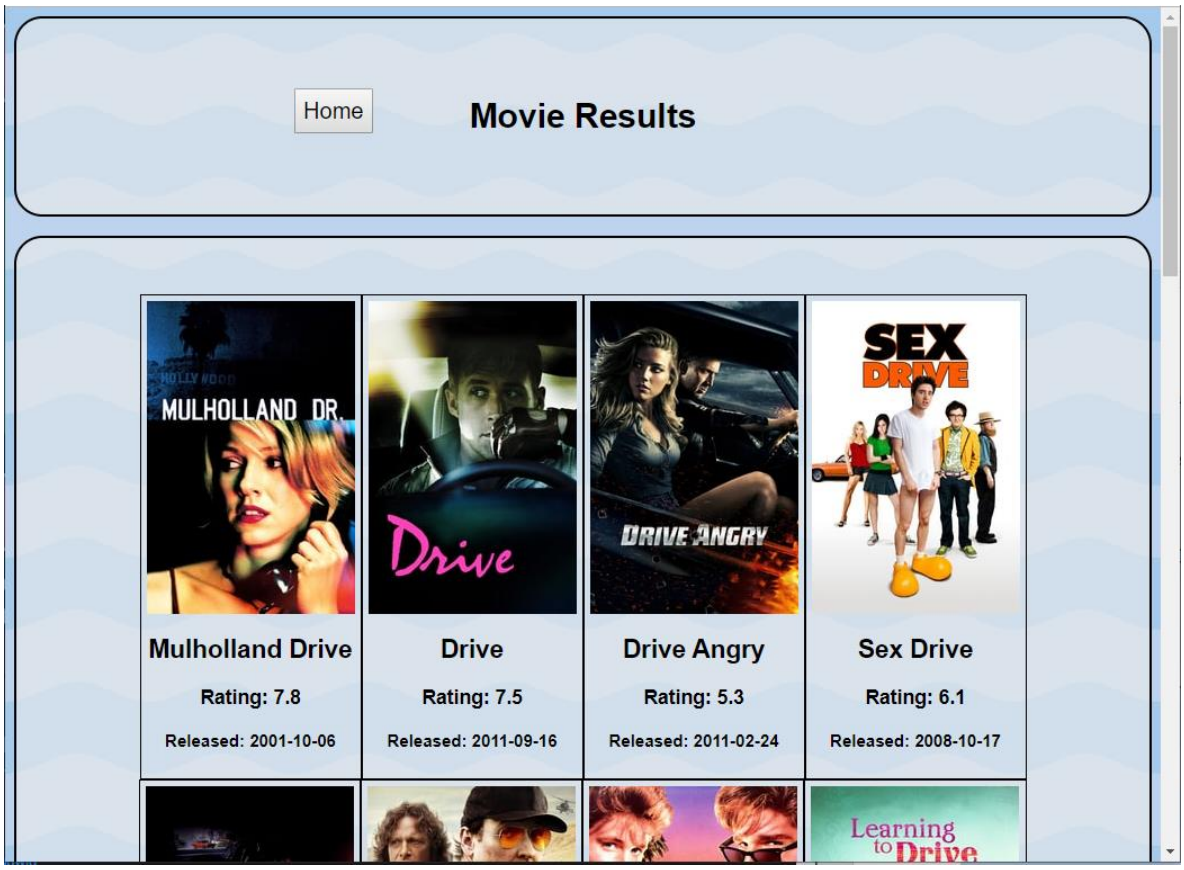
Use Case A

As a user, I want to create and listen to a playlist of music related to my favourite movie.

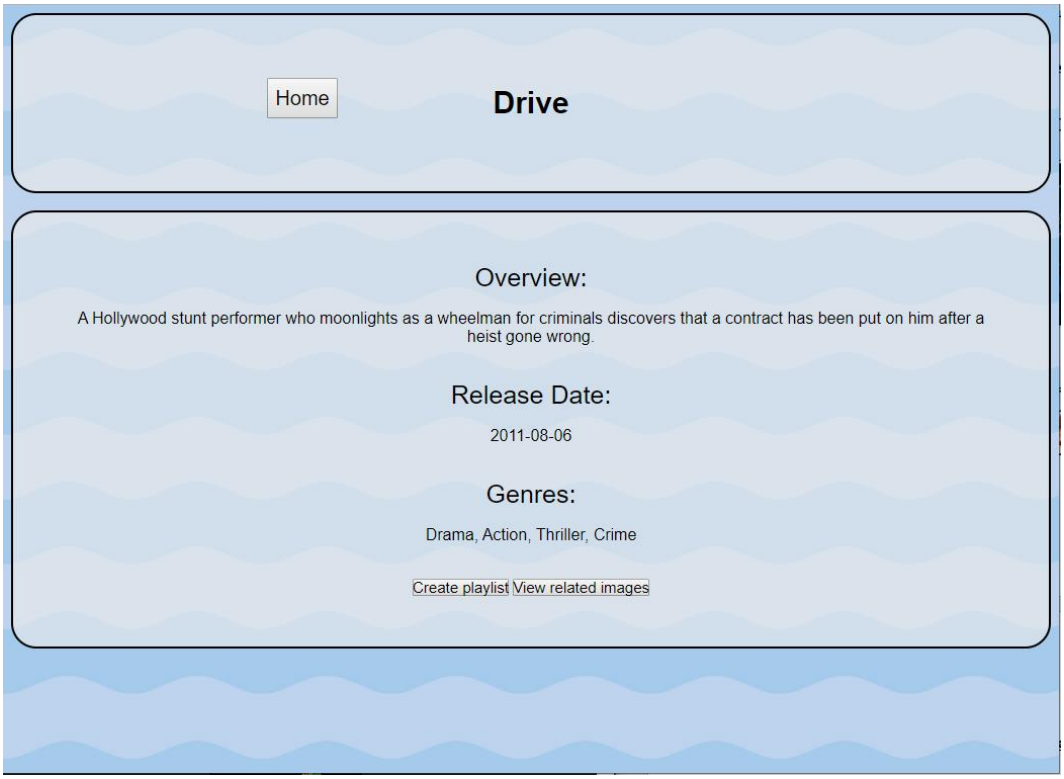
The user first navigates to the index page of the web page, enters their movie query into the search bar, and clicks on the "Search" button.



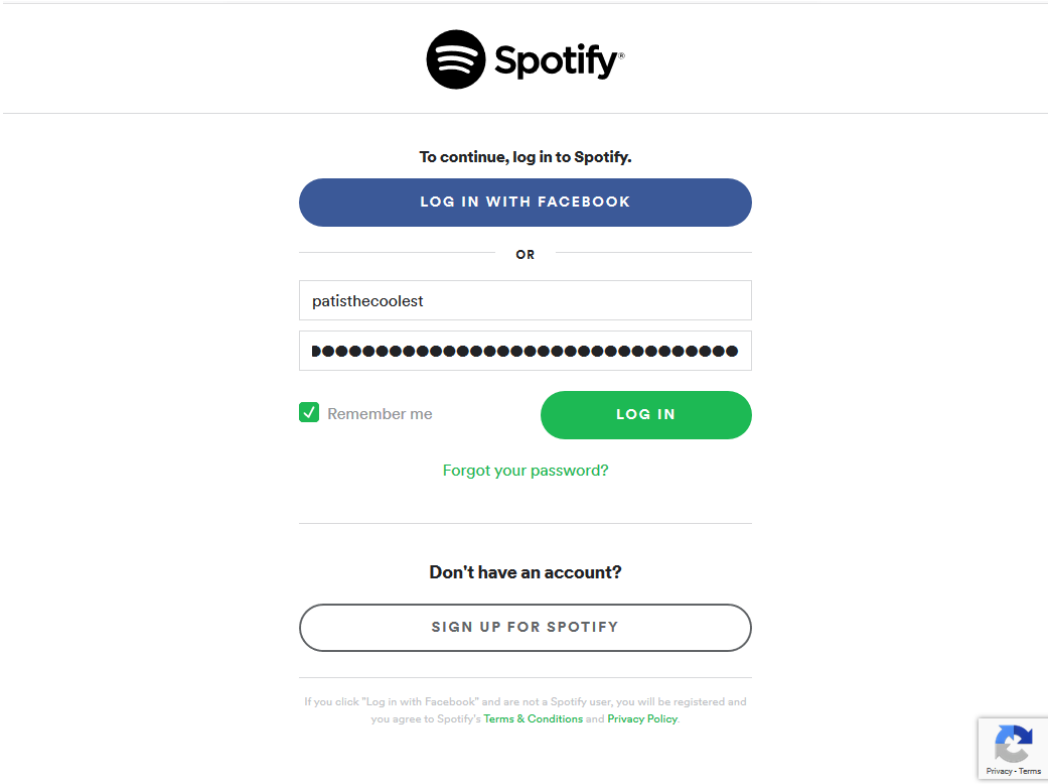
A list of movies based on the search is then presented to the user, allowing them to find and click on the movie they had in mind



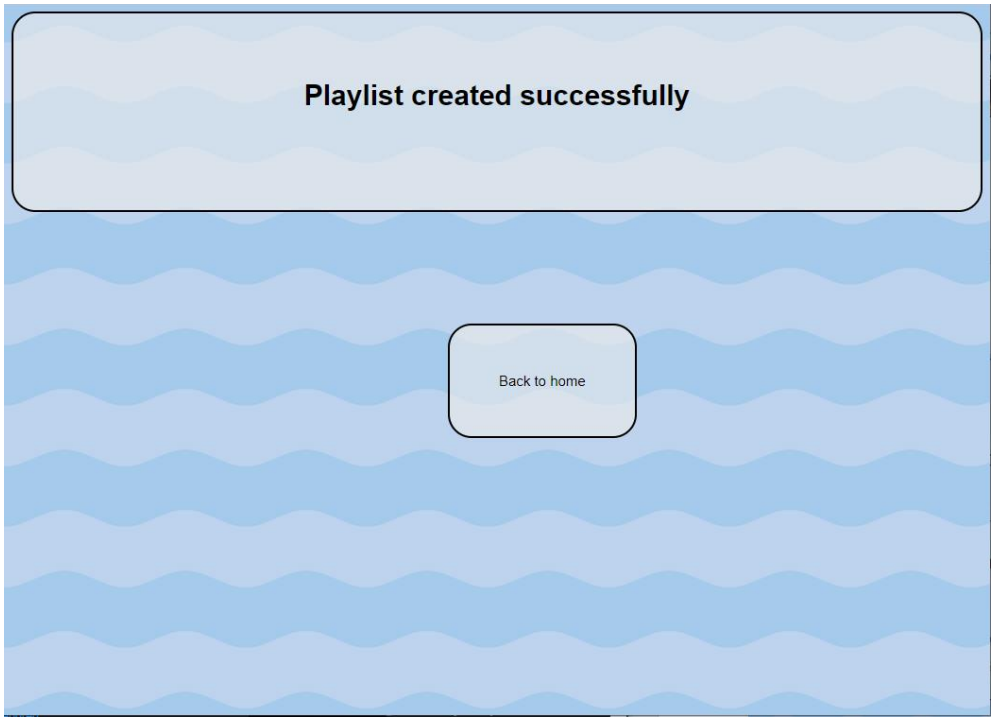
The user is then shown some details about the movie, and can then click on the “Create playlist” button



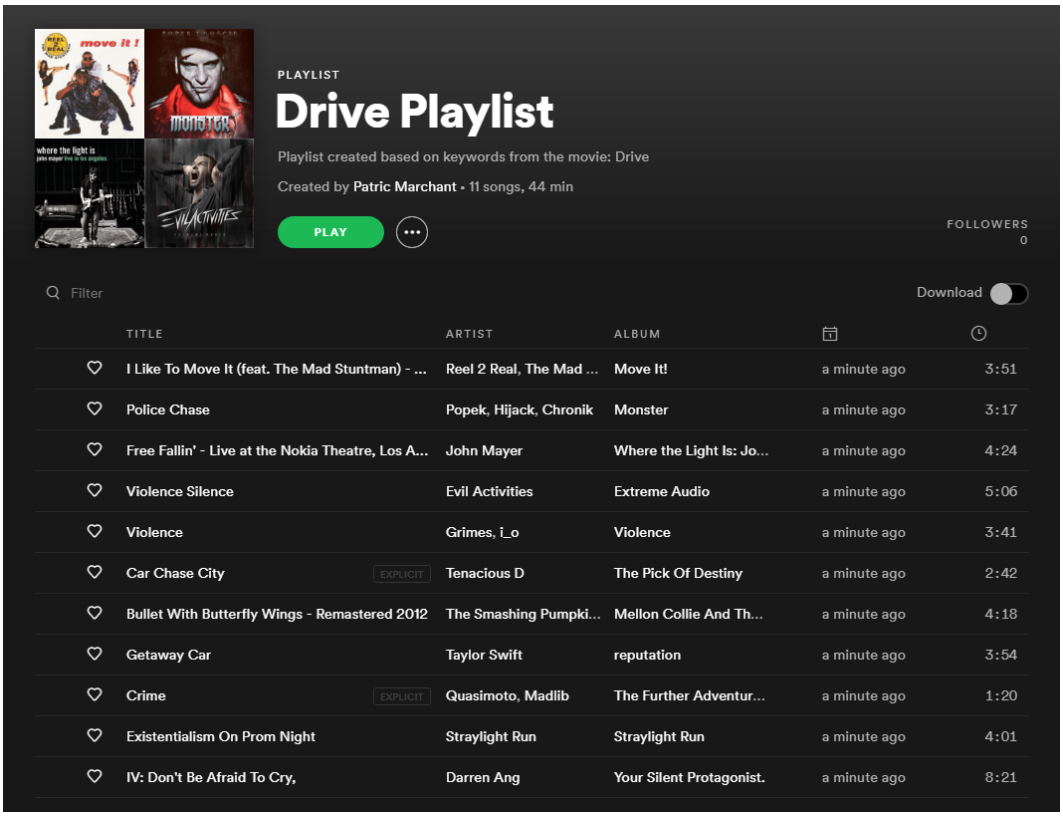
From here, the user is redirected to Spotify’s login page to authenticate themselves using their Spotify details



Once the user is authenticated, they are redirected to a page informing them that the playlist has been created successfully, and allowing them to press a button to redirect back to our index page.



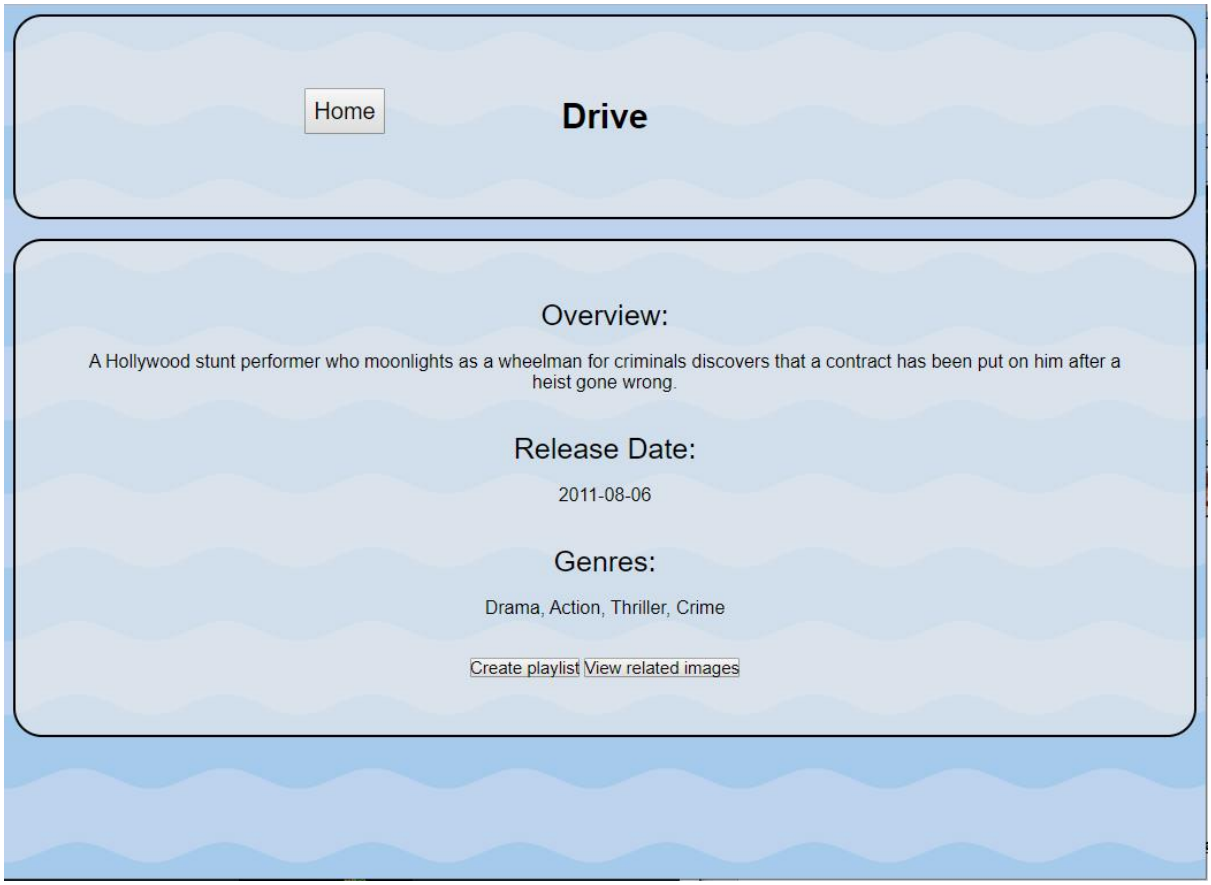
If the user checks their Spotify account, a new filled playlist will show up with the chosen movie's name in the title as well as description of the playlist.



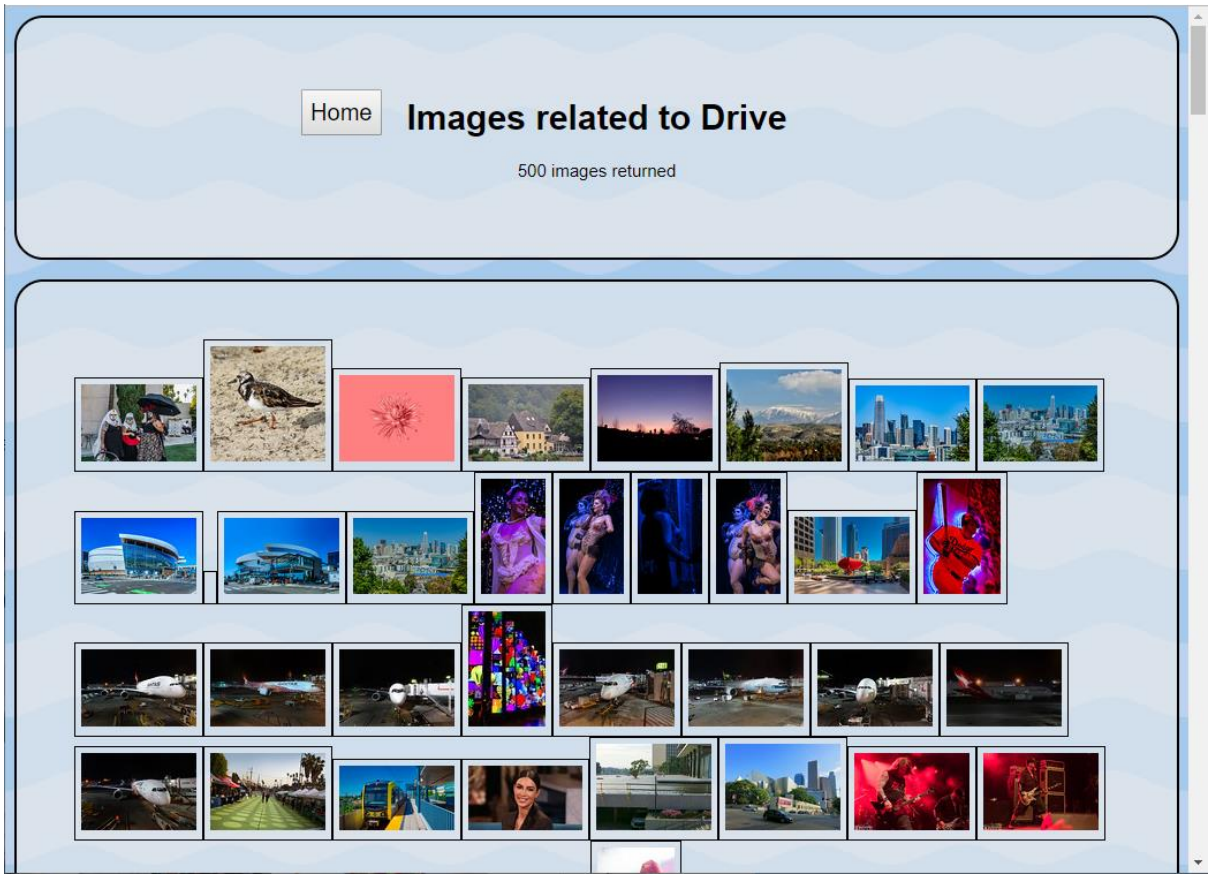
Use Case B

As a user, I want to search and view images related to my favourite movie.

As in the last example, the user will first use the index page to search for the movie, then select their choice of movie from the search page, I will not include screenshots for these actions to save space, so please see the first two screenshots from use case A. From here, the user can then press the “View related images” button.



The user will then be presented with a page showing images related to the selected movie presented in a gallery type format, links can be clicked to redirect the user to the original image on the Flickr website.



Service calls/technical description

APIs used

The Movie Database

https://developers.themoviedb.org/3/getting-started/introduction

The Movie Database is a movie/tv show website similar to IMDB which provides multiple API endpoints to GET, POST, and DELETE (depending on whether a user is logged in or not) various types of data relating to Movies, TV shows, and people (actors) through 3 main base endpoints: /search, /discover, and /find. For our application however, only GET requests through /search and more specific endpoints are used, as we only need to use the API as a database, pulling information about movies.

The “/search/movie” endpoint is used when the user enters a search query into the input box on the index page, returning a list of movies relating to the search query, the “/configuration” helper endpoint is also used on this page to build the URL for poster images for each movie. “/movie/{movie_id}” is also used once the user has selected their chosen movie, returning more information about the selected movie. Finally, “/movie/{movie_id}/keywords” is also used for the main function of the application, getting keywords based on the chosen movie, and passing them through to the Spotify and Flickr APIs for further use.

Use case A:

Spotify

https://developer.spotify.com/documentation/web-api/

Spotify is a music streaming service which also provides, multiple endpoints to GET, POST or DELETE data including playlists, tracks, album, artists, user info, and more. Out of these, our application uses GET requests to retrieve user info, and search for tracks, and POST requests to create the playlist, and add tracks.

"GET /v1/me" – to get user info

"GET /v1/search" – to search for tracks (passing movie keywords as query)

"POST /v1/users/{user_id}/playlists" - to create a playlist in the user's library

"POST /v1/playlist/{playlist_id}/tracks" – to add tracks to an existing playlist

Use case B:

Flickr

https://www.flickr.com/services/api/

Flickr is a photo hosting website which provides multiple endpoints for a range of services such as POST for uploading and replacing photos (if authenticated), GET for retrieving photos and information, DELETE for removing photos and information etc, however, the only one our application makes use of is the

"GET flickr.photos.search"

Endpoint, allowing us to retrieve photos matching our query (movie keywords in our case)

Technical description

The application is built using HTML5 for markup, CSS3 for styling, and Javascript for functionality served by a Node.js server in a Docker container, which is intended for use on an Ubuntu 18.04 LTS machine.

Server-side

On the server side, the Express JS web framework runs the page, which sits on top of NodeJS, which is all encapsulated in a Docker container. The application also uses several other node packages, which are listed below:

NodeJS

Asynchronous event-driven Javascript runtime for building scalable network applications. Used to serve pages, and send/receive requests and responses.

Express/ Express.json

Minimal and flexible node.js web application which provides a set of features for serving routes and views easily to web and mobile applications, with an added middleware function for parsing incoming requests with JSON payloads.

Axios

Promise based HTTP client for the browser and node.js, used for making multiple http requests (sometimes at once) mainly to API endpoints.

Querystring

Used to parse and stringify URL query strings for passing to APIs, used for interacting with the Spotify API when authenticating and redirecting the user.

Morgan

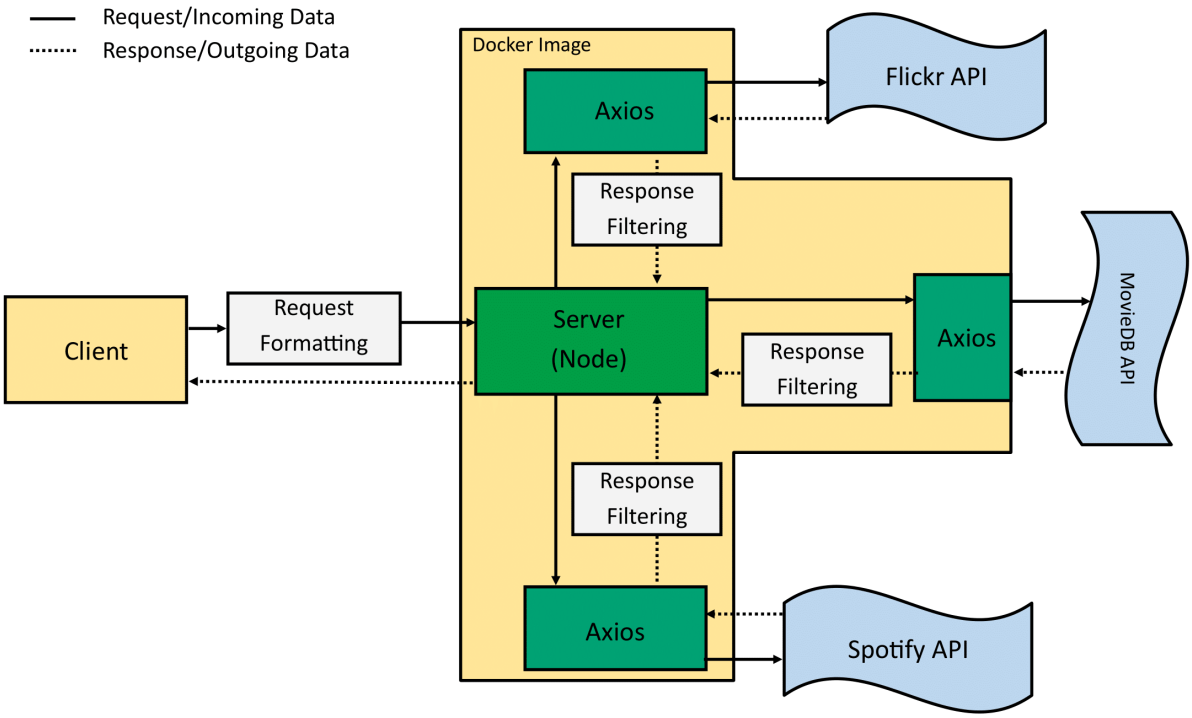
Middleware used to log HTTP requests in the command line for debugging purposes

Helmet

Used to help secure express apps by setting various HTTP headers in express.

Client-side

The client side of the application is quite minimal, using HTML5 and CSS3 to build and style web pages, with Node.js serving pages to the user.



Docker

Docker is a tool designed to make is easy to deploy and run applications within containers, which allow the application to run the same regardless of the environment and operating system, packaging libraries and dependencies and shipping it in one standardised container for easy use. The Dockerfile is key in building these applications, detailing a set of commands which help docker to build the image, and deploy it inside the container.

Our application is deployed using Docker, with the image built on Ubuntu 18.04 LTS. Our Dockerfile (see appendix) first installs Node, then copies our apps directory into the containers directory, and sets it as the “work directory”, we then run npm install to get our package dependencies, expose port 3000 (plus 80 for http), and runs “node app.js” to start the server.

Testing

Test	**User story addressed**	**Expected result**	**Actual Result**	**Pass/Fail**
Index page loads	A, B	HTML index webpage display	HTML webpage display	Pass
Search bar works and redirects user to appropriate	A, B	HTML movie search results display	HTML movie search results display	Pass

searched movies page				
Home button redirects user to index page	N/A	HTML index webpage display	HTML index webpage display	Pass
Clicking on movie in search result redirects user to individual movie page	A, B	HTML movie webpage display	HTML movie webpage display	Pass
Correct movie information is shown on movie page	A, B	HTML movie webpage display (correct title, overview, release date, and genres)	HTML movie webpage display (correct title, overview, release date, and genres)	Pass
Clicking "view related images" redirects user to photo gallery page	B	HTML photo gallery webpage with images using correct movie tags	HTML photo gallery webpage with images using correct movie tags	Pass
Clicking "Create playlist" redirects user to Spotify's authentication page	A	Redirected to Spotify's login page	Redirected to Spotify's login page	Pass
After logging into Spotify, user is redirected back	A	HTML page informing the user their playlist has been created	HTML page informing the user their playlist has been created	Pass
After logging into Spotify, playlist is created	A	New filled playlist in Spotify client	New filled playlist in Spotify client	Pass

*all screenshots for test cases can be found in Use cases/user guide section

Limitations

Tags with no search results on Spotify

When searching Spotify for tracks matching tags, most of the time tags are broad and well formatted and can be matched to a Spotify track, however, some movies exist with tags such as "aftercreditsstinger" and "duringcreditsstinger" (Avengers), and cannot be matched with any existing songs on Spotify. This is handled using a try, catch block which will skip over trying to add undefined tracks.

Possible extensions

While the webpage functions as intended, music tracks and images are returned to the user based off only keywords, which can end up being kind of trivial for actually capturing the mood/feel and giving songs/images based on the actual movie.

When creating the application, I had thoughts of using a 3rd party Spotify (and Flickr) algorithm which would allow the application to query not only by keywords, but also based

of genres from the movie. This however, ended up being too much work, as I could not find anything prebuilt that matched my needs. If we had more time, tweaking this would be my first improvement.

The Movie Database API also had multiple endpoints that also supported searching and retrieving information on TV shows as well as actors, which could both be implemented alongside movies to give users more freedom in what they want their playlist/images to be based off.

References

Foundation, N. (2019). About | Node.js. Retrieved 14 September 2019, from https://nodejs.org/en/about/

Express 4.x - API Reference. (2019). Retrieved 14 September 2019, from https://expressjs.com/en/api.html

axios/axios. (2019). Retrieved 14 September 2019, from https://github.com/axios/axios

query-string. (2019). Retrieved 14 September 2019, from https://www.npmjs.com/package/query-string

helmet. (2019). Retrieved 14 September 2019, from https://www.npmjs.com/package/helmet

API Docs. (2019). Retrieved 13 September 2019, from https://developers.themoviedb.org/3/getting-started/introduction

Flickr Services. (2019). Retrieved 13 September 2019, from https://www.flickr.com/services/api/

Web API | Spotify for Developers. (2019). Retrieved 13 September 2019, from https://developer.spotify.com/documentation/web-api/

Appendix

Appendix A: Dockerfile

```
# set base image
FROM node:10

# File Author / Maintainer
MAINTAINER Patric Marchant

#copying relevant files
COPY . /app
WORKDIR /app
COPY ./assign1/ /app

#filling dependencies and setting up
RUN npm install

#exposing ports
EXPOSE 80
EXPOSE 3000

#running server
CMD ["node", "app.js"]
```