Wiki-Flix

Through Wiki-Flix, users are able to search for a movie and view the movies rating, release date, and other relative information. Another feature is the ability to search for an actor or actress and view their date of birth, place of birth, and their known movies. Further, users have the ability to see what movies are currently playing in theaters. However, what is the use of knowing what’s playing in theaters and not knowing where to find one? To remedy that, users have the ability to locate movie theaters in their immediate area on the fly. From this point on, I will discuss all the technical details of the application.

All information obtained by Wiki-Flix is retrieved from a public API provided by *themoviedb.org*. To access the API, I created an account on *themoviedb* and was granted an API key for non-commercial use. The forefront of the Wiki-Flix is a simple yet elegant dynamic web page created using HTML and CSS. On the main page you will find a search bar and four tabs: Movies, Persons, Now Playing, Find a Theater. I will now go into further details of each tab/section.

Upon a user entering a search query, two queries occur in the background – a movie query and a person query. The process begins with the users search query being passed as a parameter to a javascript movie function. Within that function, an XML Http GET Request is made to the public API mentioned above. Attached to the URL are the API key and users search query. Once the request is sent, a JSON response is returned to the application. The response text is then parsed using JSON.parse. The API returns very general information about movies related to the search query such as movie title, release date, and movie ID. To retrieve more information relevant to the movies that are returned, I loop through the results that were returned and store the movie ID attribute of every movie in an array. I then loop through the movie ID array and create a new XML Http GET Request using the movie ID stored in the current index along with the API key. With this method, I am able to obtain all available information relevant to that movie. Through the results received for the specific movie ID, I obtain a movie poster path (/justanexample.jpg), movie title, subtitle, release date, vote average, vote count, and the movie genre. All results are printed dynamically in the ‘Movies’ tab using document.getElementByID().innerHTML. In the case that a movie poster’s link is dead, a stock movie poster that is stored in the application’s ‘WAR’ folder is displayed for a cleaner look. The results are then printed dynamically to the ‘Movie’ tab. Once the movie function is done printing all results a ‘person’ function is called where a person query is initiated.

Once the persons function is called, a request is sent to API and attached to the URL is the same search query used in the movie function along with the API key. A JSON response is received and then parsed with JSON.parse. The result list contains general information about actors/actress’s related to the search query. Similar to the movie function, to get more in-depth information about each person in the result list I loop through the result list and store the person ID attribute in an array. I then loop through each index and create a new request to API where I attach the person ID along with the API key. Through the results I obtain the person’s profile picture path, name, date of birth, date of death (if applicable), place of birth, and their known movies. If the person’s profile picture link is dead, a stock photo stored in the applications ‘WAR’ folder is displayed. The results are then printed dynamically to the ‘Persons’ tab.

The moment the Wiki-Flix web page is loaded, a function named ‘now\_playing’ is called. In this function, almost all of the code used in the movie function is reused. The only difference is when making the request to the API, I request their list of movies currently playing in theaters (which is updated daily). I then proceed to loop through the results received and store the movie ID for each movie in an array. I then loop through each index of the array performing GET requests to the API using the movie ID for more in-depth results.

The last available tab is the Find a Theater tab. This is the only tab that is not dynamic due to rendering issues with Google Maps. Once the user clicks on the tab, they are taken to a new page that looks almost exactly like the main Wiki-Flix page with the exception of a view of Google Maps. This page uses two API’s - Google Maps API v3 and Google Places. Through Google Map I am utilizing the user’s geolocation for ease of access and simplicity. In the case that the user denies the page access to their location, a default location is set in Pensacola. Once the location is determined (using geolocation or a default set location), a marker is created to indicate the user’s current position and a search for movie theaters in that area is initiated using Google Places. To create the request a request variable is created that contains the latitude and longitude of the location, the type of place to search for (movie theater in my case), and the radius of the search. Once the request is sent to the Places API, a response is received with a list of movie theaters. I then loop through the results and create markers on the map along with printing the name of the theater in a ‘Theater List’ positioned on the right side of the page.

To recap, once the main Wiki-Flix page is loaded a request is sent to *themoviedb* API to retrieve a list of movies now playing in theaters and then the results are printed in the ‘Now Playing’ tab. When a user enters a search query, two queries occur in the background – a movie query and a person query. As the web page is dynamic, when a user enters a second search term the previous query results are removed and the new query results are printed.