Movie Search is a website which allows users to search for titles based on title, keyword, and rating. The front-end had to be able to communicate with the back-end in order to apply the inputted filters, and the backend had to be able to respond with the appropriate filtered data. Therefore, I designed the system with a separate front and backend which communicated through the Django REST framework (DRF). On the front-end, we can make simple fetch calls with the appropriate parameters and url. In response, serializers convert Django models (which are connected to our MySQL database) into JSON data which our react front-end can read.

Django models connect directly to our database, so I didn’t have to directly deal with SQL statements unless I was making specific changes. I created a Movie model, with fields for the information that I would need to display on the website, such as tagline, reason for matching with the movie, etc. Then, I created a Keyword model, with a field for the word itself and a many-to-many relationship with the Movie model. A many-to-many relationship works well here because a keyword can be associated with different movies, and a movie has multiple keywords. Behind the scenes, django creates three tables for these two models. One table is associated with the Movie model, another with the Keyword model, and the third maps movie and keyword ids to each other so that we can store information about which keywords and movies go together. Besides the models, the backend was relatively simple to code. I created urls to accept parameters for the title, keyword, and rating filters, and appropriately applied them. In the front-end, I had three search bars for each filter and a search button. When the user presses the button, I create a fetch call to update the results.

In order to populate the database, I created two scripts. One reads in the excel file with movie metadata, and creates rows with the appropriate information in the movie table. The other script reads in the keywords and creates rows in the associated keyword table. Then, it reads in the associated movies and makes connections with each respective movie. These scripts were only run once in order to populate the database.

Running the server with a local host was easy. There is a detailed README file on github which shows how to recreate the project environment on a local host. In order to deploy the website to a production environment, I would have to find a host, and then export the project to that server. Using the system is very intuitive. The user simply has to enter their desired filters into the labeled search boxes and press search.

One functionality I think would be interesting to add to this system would be the ability to see related movies. For example, if a user likes a certain movie, they can ask to see movies related to that title. In the backend, we can find which movies have the most keywords in common with the title and show them as suggestions to the user. This wouldn’t require any changes to the database and would be a useful feature to have.