Movi: movie & tv recommendation service

Janae, Ryan, Dom

Update 2

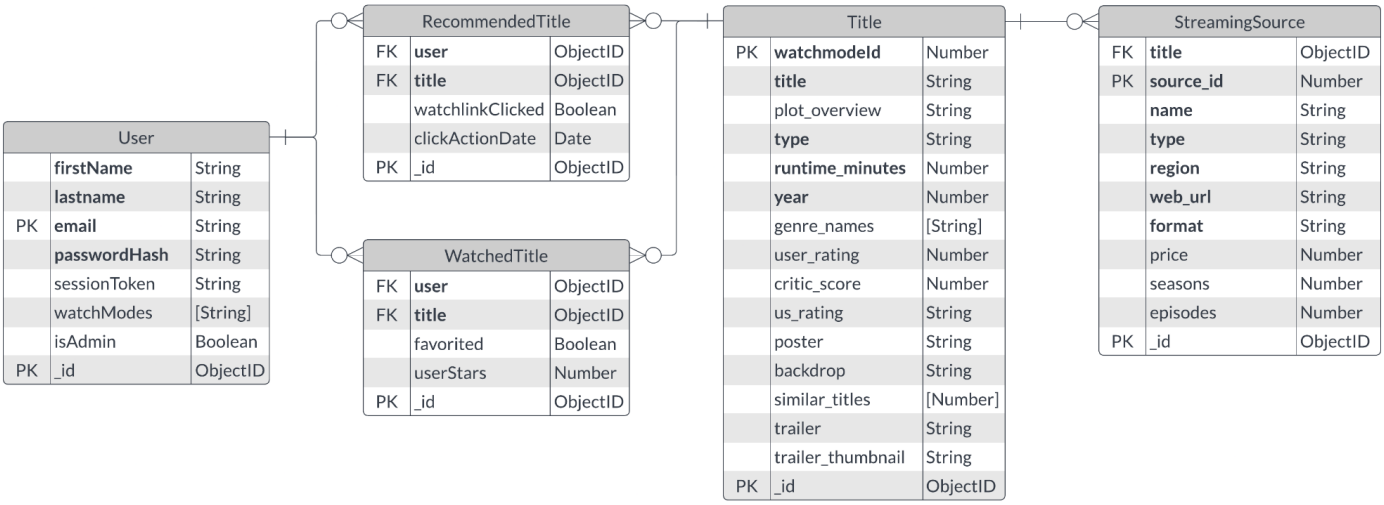
GitHub: https://github.com/jjm8759/MOVI

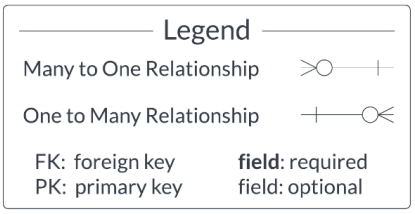
Data Modeling

Since update 1, we have grown a deeper understanding of the data needs of our project and attained a more comprehensive grasp of data modeling in general. We learned that the previous data model was not normalized in the database and would therefore require more maintenance to ensure consistency across duplications. In light of this, we have completely revamped our NoSQL data model to ensure data normalization via the MongoDB equivalent of foreign keys and to support flexible data access via data joining by foreign keys using the populate method in mongoose. We have produced an entity relationship diagram for our data model.

In summary, we have a Titles model in which each document holds user-agnostic title information, which we populate from WatchMode API calls. We then link documents in this model to documents in the WatchedTitles and RecommendedTitles models in a one-to-many relationship, normalizing fields across the database.

Movi
Entity Relationship Diagram





User - *changes from the last update*
To support data normalization, we have removed the arrays of WatchedTitles and RecommendedTitles on the user document. Instead, WatchedTitle and RecommendedTitle documents are linked to a particular User document via a reference objectId field, seen below.

We have added several fields related to user data, such as name, email, password, and web session authorization token

RecommendedTitle - *changes from the last update*
As seen in the image above, a RecommendedTitle is linked to a user on document creation via a *user* field whose value is a document ObjectId, with an added “ref” or reference to the User model. The same is done in the title field to normalize title data across the database

WatchMode API

In our endpoint logic, we have used the WatchMode API's AutocompleteSearch, TitleDetails, and TitleSources endpoints to populate our database with title information and generate results for partial string search queries over title names. The next steps include implementing extended title search functionality, which considers metadata such as source availability and genre. This will use WatchMode's ListTitles endpoint. We will use data from this endpoint heavily in our recommendation algorithms while simultaneously attempting to optimize API call cost.

Back End API

We have solidified our understanding of backend API architecture using expressJS to set up RESTful API endpoints. To support the next steps in implementing the REST API serving the React front end, we have stubbed out routing logic and CRUD methods for each resource route associated with our four database models.

Abstraction layer over external API integration (WatchMode)
A simple step to make switching or extending our external API usage easier in the future is hiding direct integration of those APIs behind an abstraction interface exposed to the rest of our backend. This practice is known as the facade pattern, although it is also related to the adapter pattern. It's beneficial because if in the future we need to switch our external API to a different provider or add in additional external APIs, we need only change the facade implementation rather than gut the entire project.

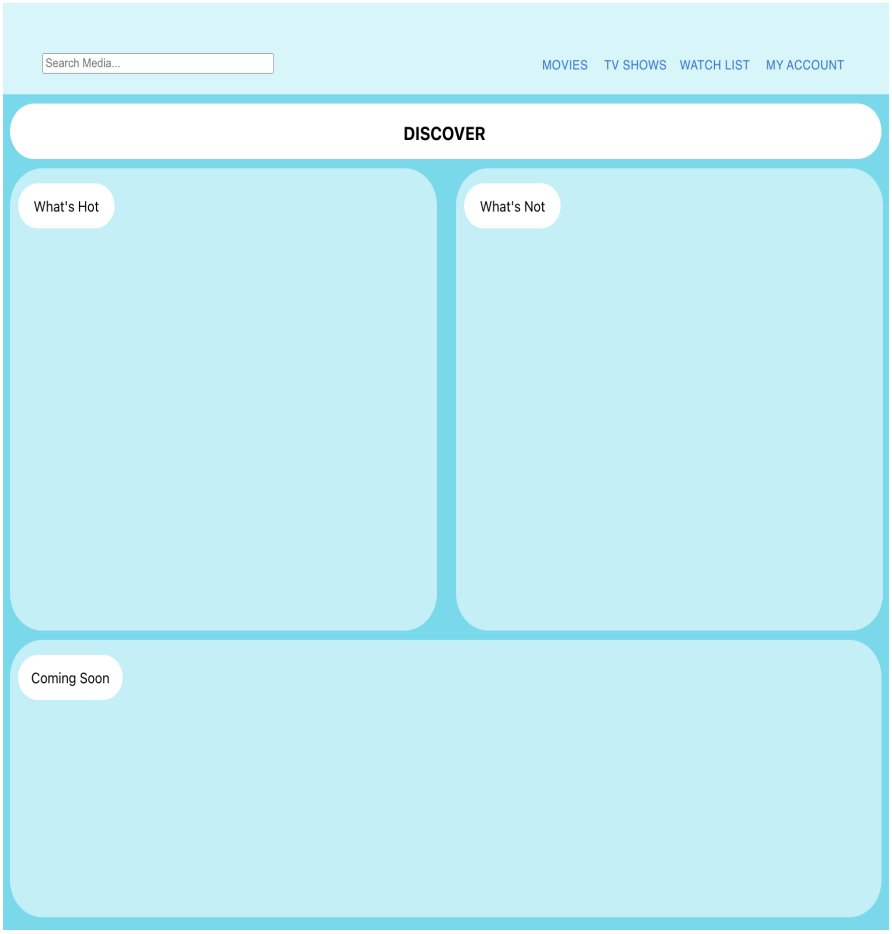
We have **completed two endpoints**: get title by id, and search by query partial string. These endpoints are all set up and ready to be used. Next, we will finish the user action endpoints, such as registering, logging in, and logging out. We will also begin working on CRUD endpoints for the watched titles and recommended titles resources.

The next steps include working on API action endpoints for the User, WatchedTitle, and RecommendedTitle resources.

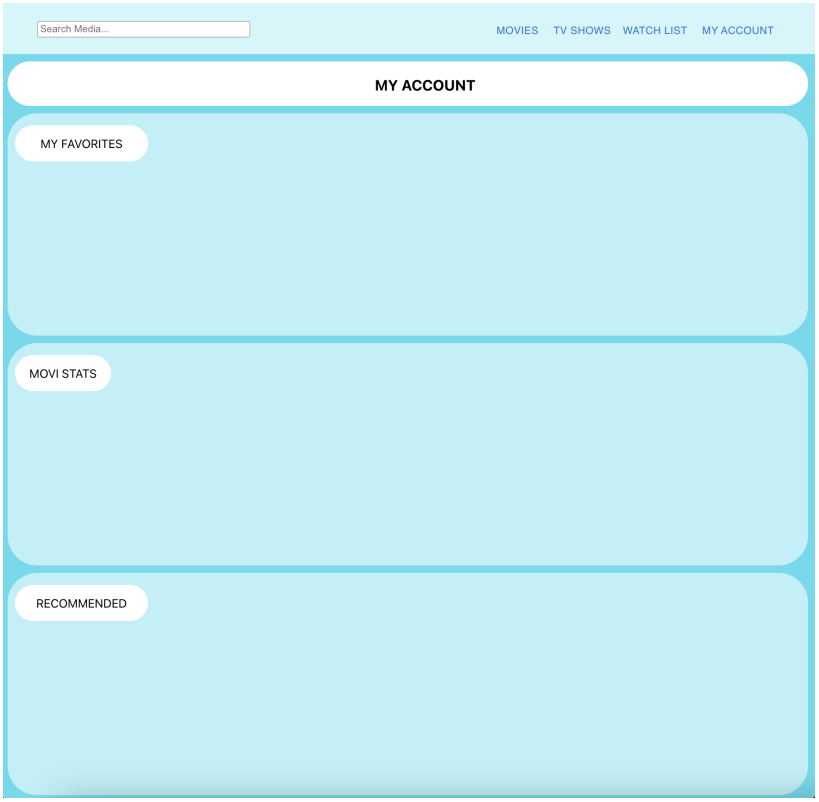
Front End

We have begun working on a basic implementation of the front-end wireframe, including a watched section and recommended section, as well as the start of a login system using Google OAuth. In addition to our authentication, we have also updated and built a functioning model of the Discover page seen in the wireframe guide using a combination of HTML and CSS, a short gif of which can be seen below, with the required buttons, search bar, and containers to hold the movies that we pull using WatchMode API. We have now begun researching and transferring this same design into a React JS model that will use the React component library to build out the front end instead of HTML. Although the learning process has been slow in terms of React, we are beginning to make headway in the process of the front end development in React.

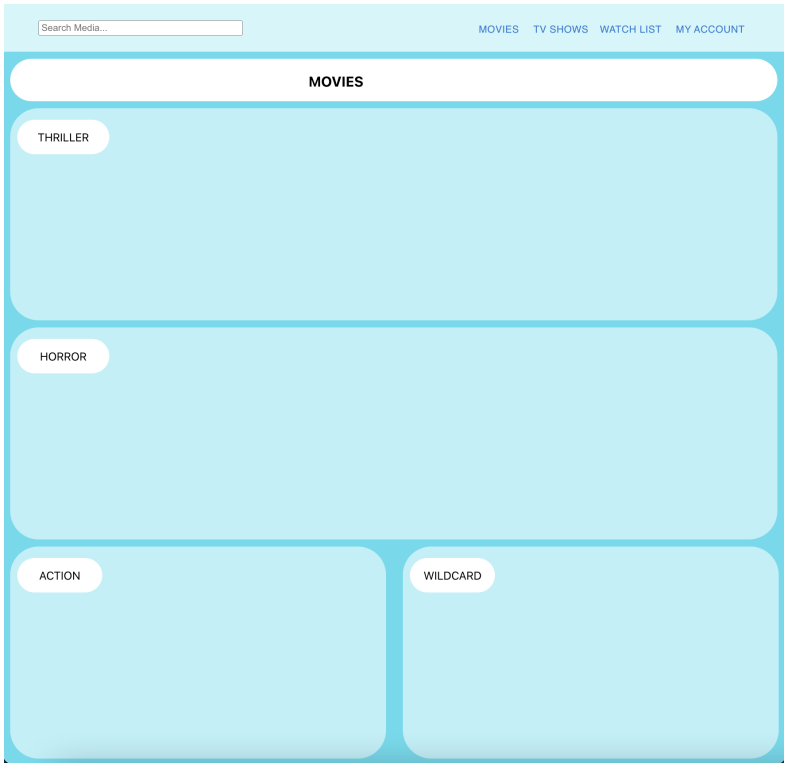
Current 'Discover' Page React



Current ‘My Account’ React Page



Current ‘Movies’ Page



Current ‘TV Shows’ React Page