(1) Project Description

Our movie organizer has a simplistic design with an advanced search bar on the left sidebar with relevant views updating in the main content area to the right of it. Our advanced search allows querying the database by minimum rating, year range, tags, age rating, media availability, genre, actors and directors which update in real time to the view when a change is made. Additionally, our basic movie search field checks string containment each time the field is changed and can update the corresponding view instantly due to the speed of core data. Having searches that can query very fast was a user goal because it eliminates the need for users to wait and constantly click search buttons. Our auto-updating when search fields change instantly reflecting changes to the corresponding view allows users to work much faster and see these changes instantly.

Our movie organizer allows querying based off of availability on Netflix, iTunes and Shomi, a feature that we found was very important to users with the high usage of streaming and online movie renting services. We implemented this in advanced search and then when a movie is selected, three simple icons of the media services denote whether the movie is available denoted by a coloured icon versus a gray-scaled icon.

We also implemented a foreign API to fetch movie posters if they’re available based on the movie title. We show these in the movie detail view to give a more visual experience to the user if the poster is available.

We implemented a watchlist which allows users to store movies in a separate view by adding it to their watchlist in the movie detail view. This feature allows users to keep track of movies they are interested in without having to go through the trouble of remembering them or searching for them again.

Our movie organizer allows users to view movies in both a table view where they can search quickly and sort columns to increase speed and a graph view, giving another representation a user can choose to view movies in.

Navigation in our movie organizer is very simple with a segmented control in the top right of the window for easy navigation, where users can click on different segments to see a table view of their search or a watchlist, with the option of viewing either of these in a graph view.

Our movie organizer allows users to add, edit, or delete movies from the database so they can customize the database, implemented with thorough input validation.

Our movie organizer supports tagging movies with strings that can then be subsequently queried for in the future for customization utility for the user

Our movie organizer allows users to rate and review movies that are stored in our database. These reviews can be viewed later if, for example, the user wanted to remember what they thought of a movie they have not seen in a long time.

(2) Interface description

Our interface allows users to either search for a movie title in a basic search field or by using the advanced search on the left sidebar. There exists a content viewer which changes views based off of what is chosen. Navigation between views is done through the segmented controller in the top right of the window, defaulted as a table view. Users may alternate between the tableview and watchlist with the segmented control in the top right with the option of viewing them as a graph view by toggling the graph segment. To view a movie's details, one must simply double click a movie to shoe the movie detail view and can return to the previous view on the click of the back button within the detail view.

Our interface is expected to behave such that whenever a change is made by advanced search or basic movie search, changes will be updated in the corresponding table or graph view instantly. Whenever a movie is selected in the table view, the content view will navigate to the related movie detail view which should contain the movie title, rating, availability on media and other pertinent information. If the selected movie can be found by the API which we implemented, a poster of that movie will update in the movie detail view. Selecting the back button in the movie detail view should return to the previous view that the movie was selected in. When a user adds, edits, or deletes a movie from the database, the system should update and reflect the changes in the corresponding views.

To add, edit, and delete movies from the database, the three corresponding buttons at the top right of the screen can be used. To edit and delete, a movie must currently be selected, but that is not a requirement to add. After pressing add or edit, a movie editor window will show where the user may modify information corresponding to that movie as they please. If a movie is being edited, the information will be filled in, but if a new movie is being added everything will be blank. There is extensive input validation in the movie editor to ensure that all information is properly formatted, including restricting the year and length parameters to appropriate numbers and names to only letters and some punctuation. If there is an error with the syntax, the user should get helpful feedback through either sounds, animations, or pop-up messages. At any time, the user can press cancel to discard any changes and close the window, or done to save their changes and close the window if their input is valid.

(3) Particular problems

Our information on Netflix, Shomi and iTunes availability is randomized information in the database and does not reflect actual availability in the real world.

When fetching posters corresponding to titles through the intermediate API, in the case of a more recent movie than the one in our database, the poster may be of the correct movie but will reflect a newer version than that of the database in some cases (Ex. Beowulf from 2007 poster on the 1999 version).

(4) Installation procedure

1. Simply open the executable file on a Mac, or open the project file (.xcodeproj) in the latest version of Xcode and run the program. (it will populate itself with data initially which may take about a minute, as the internal object database is being generated)