Watch With Us Readme

Jason Block, Palani Eswaran, Alison Kohl

http://watchwithus.herokuapp.com

Watch With Us is a web application built using Express, which is a Node.js web application framework. Express allows for the installation of a number of packages that provide various functionalities, including API and database calls. The file that handles all of the different routes is **app.js**. App.js assigns each of the route files to variables that are called in GET and POST requests in handlebars pages.

Our app also leverages node packages. You can view the modules we used at seniorproj/watchwithus/node-modules. Most of the modules listed here are part of Express or are Express dependencies, but the notable modules are:

- session - managing session variables
- node-touch - swipe listeners
- request - make HTTP requests and process responses, used for all API calls

The application is also built using Handlebars, which is a templating system used when passing data from server side javascript to client side javascript and HTML.
Our database management system is Firebase, whose API allows for easy data queries, modifications, and additions.

We are hosting our application on Heroku.

The last external sources we are using are two APIs - OMDB and The Movie Database (TMDb). We use OMDB to retrieve all relevant information when we have a given movie id or title. We use TMDb when we are running our algorithm and querying for a list of movies based on a user's (or group of users') preferences.

Following our code

Our process begins in index.js, which is a trivial file that renders the index.handlebars page. From this handlebars page, the user can either login or create account. If login is clicked, the get method of login.js is reached. This method clears any login data and renders login.handlebars. If create account is clicked, the get method of createaccount.js is reached. This method merely renders login.handlebars.

<u>Create Account flow</u>

We ask a user to enter a username, email address, and password when they create account. Upon creating an account, the program goes to onboarding.js, with the email passed in as

form data. Because the email passed in is not undefined (line 29), we know we are creating a new user and jump to line 232. The process starting at 232 creates a new account in Firebase, and blank *adjusted-ratings* for each movie genre and time period.

Adjusted-ratings are the way we store user ratings in Firebase: When a user rates a movie, we store the *difference* between their rating and the average rating for that movie, as pulled from TMDb. We then average this into the *adjusted-rating* for each genre that the movie was labeled with, and its decade. So, for example, after rating some movies, every user in the database has key-value pairs such as “Drama: +0.3”, “Comedy: -0.5”, and “1990s: +1.1”. When we recommend movies in “Find Movie”, we find the average of these values across all the friends that have been indicated in the group, effectively combining them into a single user whose genre and year *adjusted-ratings* are stored.

After creating the account, onboarding.js gets the first id of movies to rate (to help our algorithm) from a list in Firebase and queries both the OMDB API (line 342) and the TMDb API (line 351) to retrieve additional metadata about it. It then renders the onboarding.handlebars page, which displays the movie title, movie poster, five stars which the user uses to rate the movie, a skip button, and a progress bar. If the user rates a movie or skips, we direct to onboarding.js once again. This time, “email” will be sent to onboarding.js as “undefined”, and will be caught on line 29. This indicates that the user has just rated a movie (or skipped), and thus this should be stored instead of a new account being created. Firebase is queried on line 42 for that user. If they submitted a rating (didn’t skip), we push a new rated movie object in lines 66-71, in which we store the title, user’s rating, average user rating from TMDb, and the difference between those numbers (the *adjusted-rating* as described above). On lines 83-98, we average in the *adjusted-rating* that they just submitted for this movie into the average decade for the movie (as explained in paragraph above). Lines 102-178 do the same for genres, with the OMDB genre names that we receive from the form_data being mapped and converted to TMDb genre numbers (lines 108-160). This is necessary because we query TMDb with these genre numbers to generate the list of movies to recommend, but the genre metadata for each movie (along with some other metadata that cannot be found on TMDb) is pulled from OMDB.

On line 193, the next movie title in the list of onboarding movies to be rated is pulled up (from Firebase, line 187). OMDB is queried for this movie title to get the metadata (line 200), a further query is issued to TMDb for more metadata (line 218), and the package of data is sent back to onboarding.handlebars (line 223).

Once a user has rated 10 movies, the Get Recommendations button is enabled and the user can go to the main menu.

Login flow

This is fairly straightforward. Login.handlebars takes the user to the final else method of the post request in mainmenu.js, which authorizes the login data and directs the user to mainmenu.handlebars.

Find Movie

From the main menu the user can click “Find Movie”. This directs to addfriends.js, which gets all of the users and the current user’s recent friends. It creates these two arrays and passes them to addfriends.handlebars. Addfriends.handlebars has a textbox that displays recent friends and all other users, and using a bootstrap element allows selection of multiple users. Once the user clicks “Let’s Go”, a loading spinner is created. This directs to the post method of findmovie.js.

Findmovie.js is the backend javascript for the “Find a Movie” page, which displays the actual recommendations. It contains the algorithm for generating the recommendations and for retrieving the metadata about them (year, genre, director, etc), as well as the logic involved with saving a rating if the user decides to rate one of the recommended movies during the “Find a Movie” process (when they click the stars, not when they select “Watch This Movie”).

The code within findmovie.js will now be explained in detail:

Starting at the top of the post function (line 110), we first retrieve the user’s ID and the friends they have indicated they are watching with (line 116). If rateMovie is not “true” (line 140), then we have reached this page from the user clicking “Find Movie”, and thus we need to turn the algorithm.

The algorithm:

The algorithm begins by querying Firebase, one user at a time (line 147), to find their average *adjusted-ratings* for each movie genre (lines 155 - 189) and decade (lines 189 - 219).
The algorithm then generates array of queries for TMDb through the getQueryArr method (line 58, called on line 225). This method takes in the average genre and decade *adjusted-ratings* for the group, and combines genres and decades together into a series of single *adjusted-ratings* (i.e. “Drama: +0.3” and “1990s: +1.1” ---> “Drama, 1990s: +0.7”), effectively creating a “single user” that represents the preferences of the group. If a genre-decade possibility is a clear outlier compared to the groups usual rating trends, it is thrown out from consideration (lines 73 - 77). If it is not thrown out, then a TMDb query for it is generated (line 90) with the minimum average user rating part of the query being adjusted according to the group’s *adjusted-rating* for this genre-decade combination.

After we obtain the array of queries, we must run the queries on the TMDb API to obtain the recommendations. The number of movies to be queried for each genre-decade combination is calculated (line 228). Starting with line 246, we run the queries (directing to a random page of results, selected on line 244, since TMDb provides only 20 per page) and populate movieString (declared on line 232) with movie titles and the average user rating for that title (Format: title*rating;title2*rating;title3*rating).

The next step is adding the remaining metadata for each movie (such as genre, runtime, synopsis, directors, actors, etc.) to the movieString. This data must be obtained from a different API, OMDB, because TMDb does not have it. Starting on line 309, we run through the generated list of movie titles, query OMDB for them, and obtain the extra metadata about each. This is added to “movieStringNew” (line 363). When the entire string has been generated (Format: title*metadata1*metadata2*…metadataX;title3*metadata1*metadata2…), it is sent to findmovie.handlebars.

End algorithm.

Findmovie.handebars first uses the movie string to create a number of arrays for each attribute of a movie. The title, poster, five stars, and watch movie button are displayed. If the user taps on a movie poster, an overlay will come up with info such as the runtime, year, director, actors, and more. Our code allows the user to swipe or click left and right to get to a new movie and allows the user to tap on the poster to see more movie info. Finally, upon clicking one of the stars, a form is submitted to findmovie.js, with the “rateMovie” flag set as true. This is caught on line 370 of findmovie.js, and Ii this case we save the rating information just as we do in onboarding. We also remove that movie from the list of movies in the movie string, and finally the findmovie page is re-rendered.

If the user clicks “Watch Movie” from findmovie.handlebars, we direct the user to rateonemovie.js, which takes in the form data and passes it to rateonemovie.handlebars. Rateonemovie.handlebars displays the movie information and five stars with which a user can rate the movie. Upon submitting the rating, the program directs to the **if (fromRateOnMovie == “true”)** portion of the post method. Just like onboarding, this process enters movie rating info into the database. Mainmenu.handlebars is rendered.

<u>View/Edit Ratings</u>

From the main menu the user can click View/Edit My Ratings. This directs to viewratings.js, which gets all of the movie ratings for a particular user, and then creates four differently sorted arrays before rendering viewratings.handlebars. Viewratings.handlebars displays each movie title and each rating. This is done by creating a form with fields for each title and rating; the rating is editable. Upon clicking home, this form is submitted to the post method of

mainmenu.js. If this method detects that it is coming from the viewratings page, it updates any ratings that have been changed.

Account Page

From the main menu the user can click Account Page. This directs to the get method of accountpage.js, which finds the username and email of the logged in user and directs that information to account.handlebars. Account.handlebars displays the username and email address and allows the user to edit the email address and password. If the user tries to do either of these things, the program goes to the post method of accountpage.js and makes firebase calls to change either the username or password. Account.handlebars is then rendered again with the appropriate confirmation/error message.
A user can also log out from this page.

Style.css

The progress bar, spinner, select, and loader CSS was found online to help us implement the progress bar in onboarding and the loading spinner on the add friends page. Our UI also uses Twitter Bootstrap. However, we override many of these styles in style.css to make them match the look and feel of our app. Virtually all of the Bootstrap elements used in our application have been customized.