Summary Document

I began this project by outlining what I would need to create. The back-end consists of a Node.js Express server to handle the APIs for indexing all records to Algolia, adding, editing, and deleting single records. The frontend is written in React.js and consumes the back-end APIs.

I began by building the back-end by first installing Express and PM2 for running the server. I chose PM2 because it's a great monitoring tool to run Express. I really enjoy the features and how it works. I set up the environment variables and installed axios to call the movies json file from Github. I then created an endpoint, /api/v1/index/all-movies, to index to my Algolia account with the index movies_assignment using algoliasearch npm package. I decided to index the first 100 records from the json file to keep the assignment manageable.

I then created the APIs asked for in the assignment. Additionally, I added a GET endpoint to get a single movie record to allow me to test the other endpoints. In the POST and PUT apis, I utilized a cleanObject() function I created to only allow certain key/value pairs to be posted or updated to the Algolia index. I also used the express-validator package to sanitize values getting sent via the apis.

Next, I tackled the front-end. I opted to use React.js and not TypeScript. I don't know TypeScript well enough but am confident I can quickly learn it. As this project uses React 18.2.0, I did not have to import React from 'react', since, as of React 17, this is no longer necessary. I decided to use Bootstrap to keep the front-end app easy to layout and make it predictable from a UX perspective. I installed algoliasearch and react-instantsearch-dom npm packages. I added the search box, refinement list, and hits components and themed them using Sass.

I created the switch between grid and list layouts. I would normally like to use the useRef and onClick to change the className where the useRef is attached to to handle the grid vs list view; however, the Algolia hits adds it's own DOM elements and I could not add the ref to what I want. I was also going to remove the actors list when in grid mode since it made the cards too big, but decided it was a better UX to show the actors, especially to show the user when filtering with the actors facet to see if the search results are correct. Given more time I would layout the cards in the grid mode differently to make better use of the space.

For the edits, I added the modal form component in each result. I would rather have one modal window and pass the object id for the record to the modal window; however, this would require passing the object id to the parent which is not possible. If I had more time I would have implemented Redux for a central store. I did, however, use a forwardRef to pass the reference to the modal child component. This way I could use it for both editing and adding movies.

For adding a movie, and editing movie records, I opted to use the Formik form library along with YUP for input validation. This was to make it more standardized because these are popular

libraries. They are also well trusted for validation. I also should note that I used basic authentication on the Express server. I would prefer to use a bearer token but it was too much time to set that up. Outside of my local environment, I would also use something like Netlify functions to hide the credentials from the browser on the front-end SPA.

Finally, I started writing a Jest test but ran out of time.