1. Running the Application

In order to run the application, clone the repo to your computer, and make sure you have NPM installed. Then, using a command prompt, run the following commands in the repository:

a. npm install
b. npm start

The application should start in your browser. If it doesn't open automatically, navigate to http://localhost:3000/ to view it.

2. Component Breakdown

The components can be broken down into the different lists that compose the app.

a. **The main app:** Contains both of the lists, and provides the state management for the favorite characters list.
b. **The searchable lists:** Consists of two main components: the first one is the search bar, which uses debounce to make sure we don't send excessive requests (SearchBar.tsx), and the list display itself (SearchableList.tsx).
c. **The favorites list:** Consists of the list of all the characters which were marked as favorites (FavoritesList.tsx).
d. **The list component:** Found inside the common folder (CharacterList.tsx), and handles displaying a given list of characters, alongside supporting pagination, and utilizing character cards. The List component is also responsible for fetching home planet details. The logic for pagination is handled separately in the favorites and searchable lists.
e. **Character display components:** Consists of the character sliders (CharacterSlider.tsx) and the popup modal (CharacterModal.tsx). The character cards display the basic information for the list displays, and the modal contains the data for the current character selected to be displayed.

3. Services

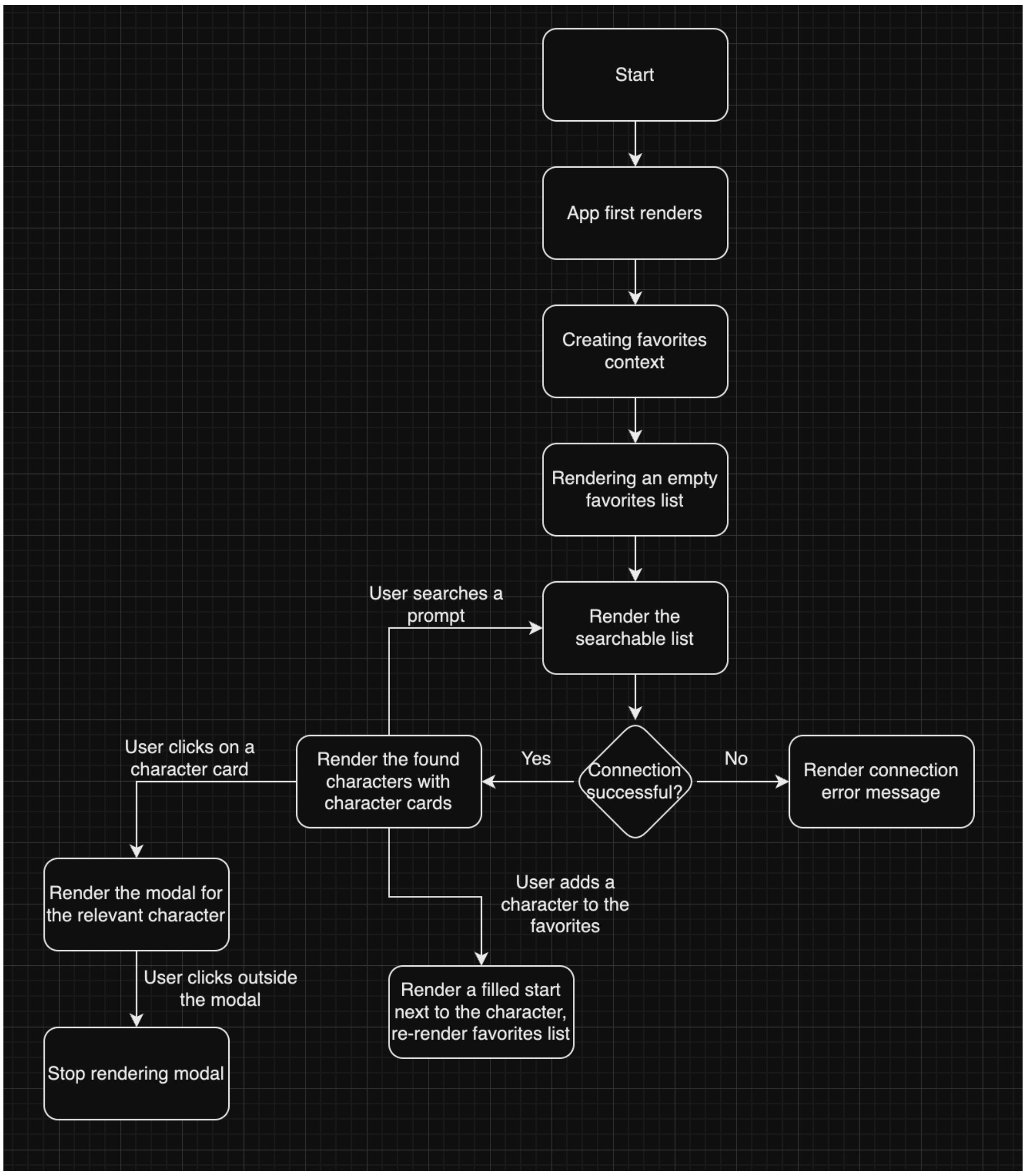
The app has internal state management for the favorites list, and it uses an API for making data requests from the SWAP API.

a. **State Management:** Most components do their own internal state management using useState and useEffect hooks. However, since the favorites list needs to be available to both lists (for the searchable lists to know which characters have already been marked as favorites, and for the favorites list itself to be able to display all characters that were selected as favorites), we utilize context to hold a record of all the favorite characters. The record is built through using the character's name as the key, and their info as the value.

b. **<u>API:</u>** We have two main functions that are called by the components, and some helper functions which are used internally inside of the API. The first function is *HandleCharacterListFetching* which is in charge of fetching the characters (utilizing the *fetchPagination* function in case of new page fetching, or *SearchForCharacters* in case of a new search prompt), matching images (i.e. calling the random image generation function, *fetchRandomImage*) for each of the characters, and making sure to use the favorites existing images to not create parity issues between the character cards. The second function used by components is *fetchHomeworld* which fetches the homeworld details for the modals.

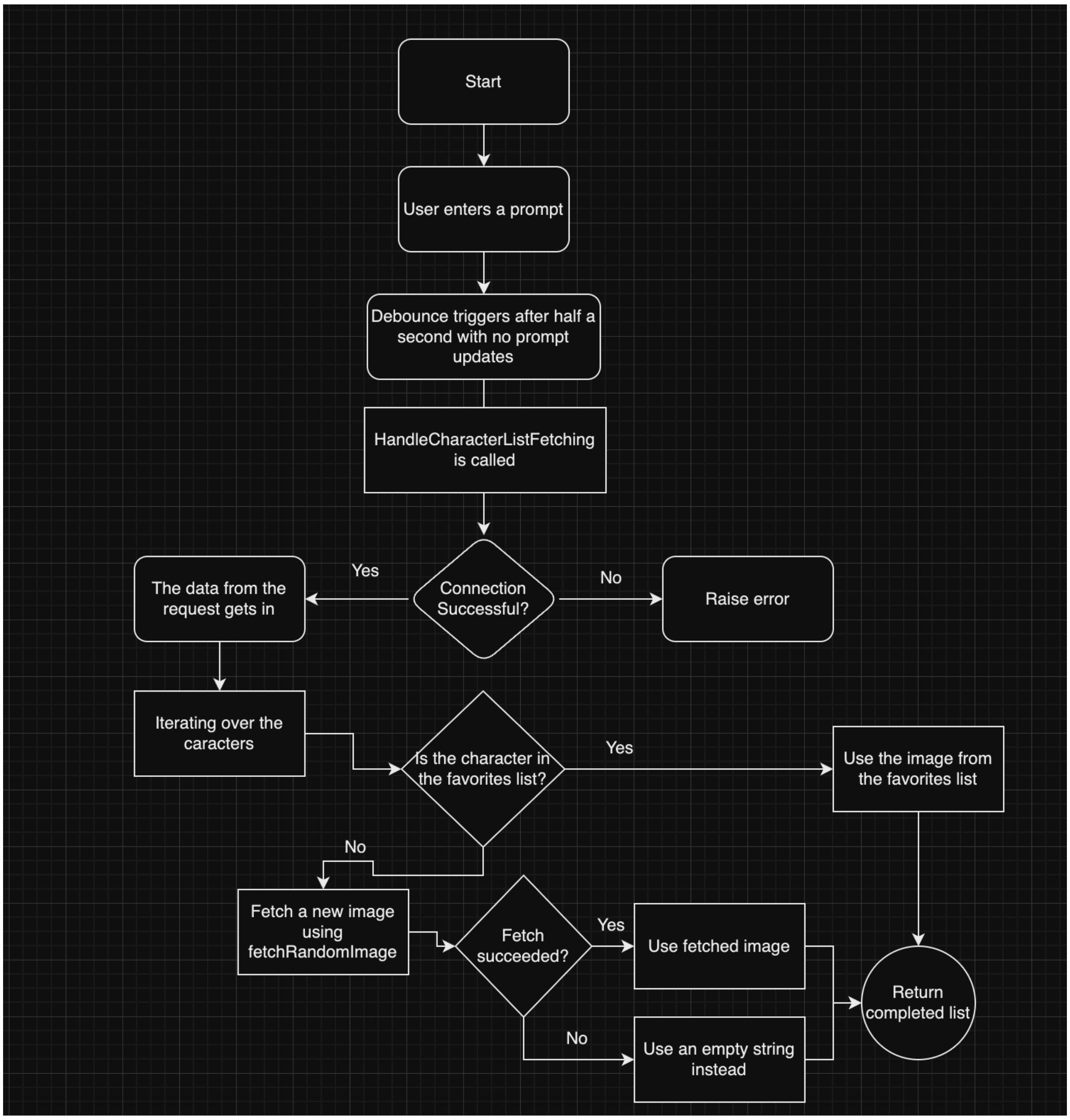
4. Interaction Flow Diagram

a. Rendering flow



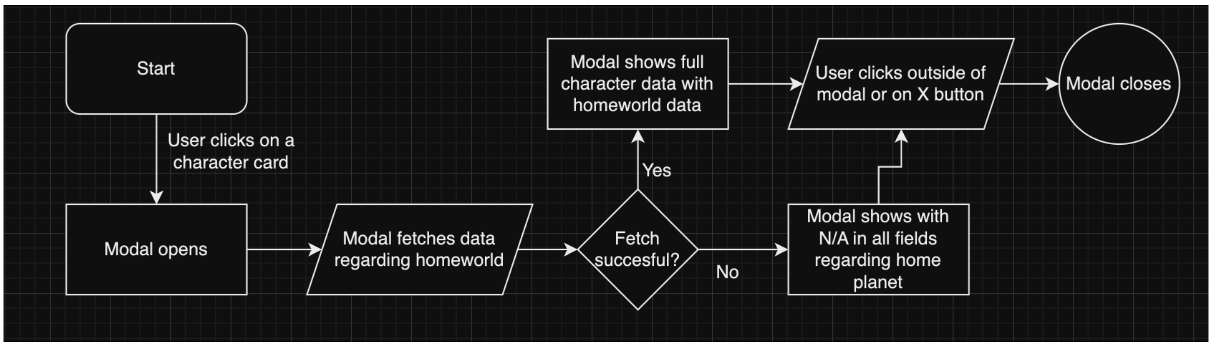
Start
App first renders
Creating favorites context
Rendering an empty favorites list
User searches a prompt
Render the searchable list
User clicks on a character card
Render the found characters with character cards
Yes
Connection successful?
No
Render connection error message
Render the modal for the relevant character
User adds a character to the favorites
User clicks outside the modal
Render a filled start next to the character, re-render favorites list
Stop rendering modal

b. Search flow



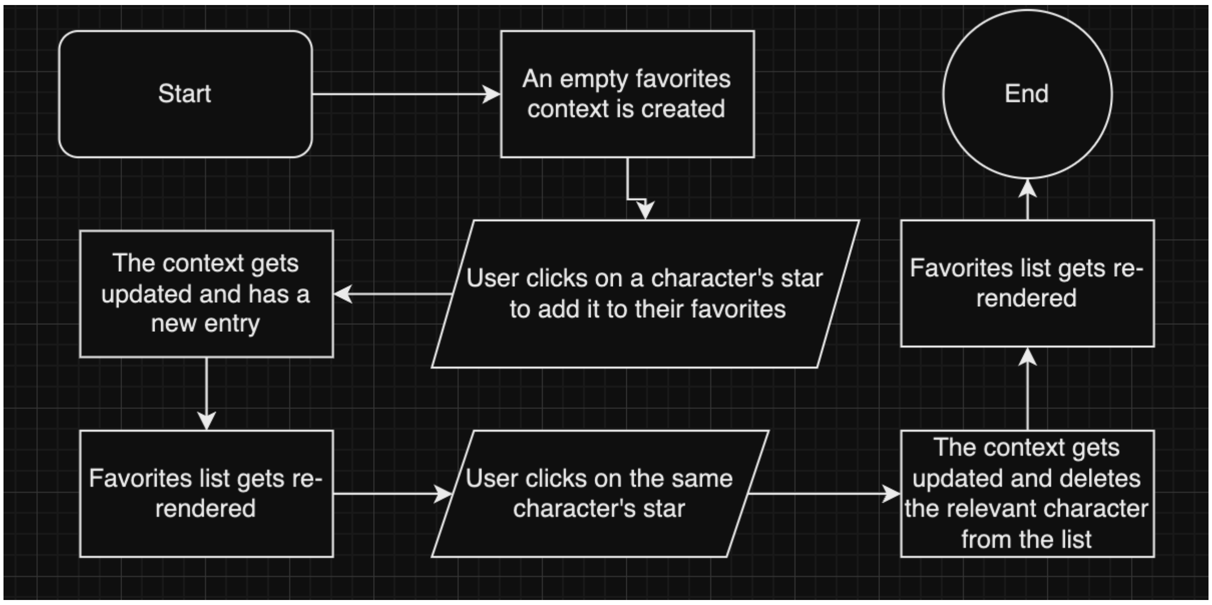
Start
User enters a prompt
Debounce triggers after half a second with no prompt updates
HandleCharacterListFetching is called
Connection Successful?
Yes
No
The data from the request gets in
Raise error
Iterating over the caracters
Is the character in the favorites list?
Yes
Use the image from the favorites list
No
Fetch a new image using fetchRandomImage
Fetch succeeded?
Yes
Use fetched image
No
Use an empty string instead
Return completed list

c. Modal interaction



Start
User clicks on a character card
Modal opens
Modal fetches data regarding homeworld
Fetch succesful?
Yes
Modal shows full character data with homeworld data
No
Modal shows with N/A in all fields regarding home planet
User clicks outside of modal or on X button
Modal closes

d. Favorites management


Start
An empty favorites context is created
End
The context gets updated and has a new entry
User clicks on a character's star to add it to their favorites
Favorites list gets re-rendered
Favorites list gets re-rendered
User clicks on the same character's star
The context gets updated and deletes the relevant character from the list