BRAINSTER

ReactJS
Thinking in React

Brainster Web Development Academy

Table of contents

Planning a React app

- React has a recommended way of how we should think about the application and the way we structure it and design its functionality.

- To that end, we're going to follow those steps and build a fully working movie listing application, following the recommended steps.

- Remember this is the recommended way, you can always tweak it your way

Start with a mock

- A mock can mean many things, but in our particular case: have some data to work against (doesn't have to be real data) and usually have a design to create the application with.

- In our case, we have the data, but we'll just wing it for the design

Exercise I

- Create new react app using typescript and name it movie list

- Import data.json into your App.tsx using:
  - *import movies from "./data/data.json";*

- Console.log the data from the json

Break UI into a component hierarchy

- That means - given a certain feature set, try to imagine all of the components that you'll need in order to make the application fully functional.

- In addition to making a list of components, try to imagine a hierarchy for them - in other words, which component would contain which other component (because later this will reflect on the state).

Features of our movie list app

- Given the JSON dataset we imported in exercise I:
    - We'd like to show some movies on screen
    - Be able to search through their titles using real API requests
    - Be able to display additional data for the movie (poster etc) in separate page (pagedetails)

Initial set of components

- App.tsx - initial wrapper of whole application, here we will keep the router and the initial state of the movies

Pages

- Home.tsx - Page that will display the initial movies
- Details.tsx - Page that will display single movie

Sub Components

- MovieList.tsx - component that will use props to receive all movies and render them as cards (using Movie component)
- Movie.tsx - renders individual movie card
- Header.tsx that holds the header of the application
- SearchBar.tsx which renders input that will allow us to search through movie titles using the API

- Add more if needed

Build a static version in React

- Which means - just place the components on screen in their proper hierarchy. Don’t worry about state just yet.

- If your component requires some props, just pass some dummy props for the time being.

- The benefit of this approach is that by the time you get to state and data, you already have a library of reusable components to build the final product with.

Exercise II

- Design and write all of the components listed in the “Set of components”slide.
- Since we don’t have designer available, feel free to design and style the app yourself.
- Remember not to worry about the state yet, just create some dummy props if needed.
- Feel free to use any CSS/UI library that you want.

Minimal representation of UI State and where should the state live

- To make your UI interactive, you need to be able to trigger changes to your underlying data model. React achieves this with state.
- To build your app correctly, you first need to think of the minimal set of mutable state that your app needs.

- Think of all the pieces of data in our example application. We have:
  - The List of movies
  - The Search text the user has entered
  - The filtered list of movies
  - The single movie data when on details page

- To filter out if all this is state we need to ask ourselves this 3 questions:
  - Is it passed in from a parent via props? If so, it probably isn't state.
  - Does it remain unchanged over time? If so, it probably isn't state.
  - Can you compute it based on any other state or props in your component? If so, it isn't state.

- Some of the above is not state, figure out which one.

Our application state

- In other words, start asking yourself questions as:
  - What are the possible inputs?
  - What are the default values of those inputs?
  - What's the data I'm trying to show / interact with?
  - Is there state that we pass down as props to a component?

  - We only have a single input for now - the search box.
  - It's state has to live somewhere - since the App component is the highest-level container, that's probably a good place to put it.
  - We also have a JSON file describing the movie data. That's exactly the data we're trying to show and interact with by way of our application.

- To put it in practical terms:
  - We need a field for the search input in App's state (string)
  - We need a field for the movie data in the App's state (array)

Exercise III

- Since we are starting to use some real data, Probably now is the best time to add the typings

- Create separate file types.ts and define all the application types there.

- Start with
  - MovieType this should match one object from the data.json
  - Add more as you need

Exercise IV

- Add the state described in the previous slides to the App component.
- Load the movie data into the appropriate field.
- Pass as props the search input value to the search component, and add an appropriate event handler for the changes of its state.

Exercise V

- Add a router using npm install react-router-dom

- Add routes for:
  - Home - “/”
  - Details - “/details/:id”

- Render respective components to match the routes

Exercise VI

- Create the Header component with the search input
- Lift the state from search to the App component where we already created local state

Exercise VII

- Attaching real data to the application:
  - Follow this link
  - Open the API Key in the menu
  - Choose free option
  - Enter your valid email, and hit Submit

- Open your email and follow the instructions
  - Activate the key
  - Copy and remember your free movie list API Key
  - Than follow the documentation, to get familiar how your API works

- In your App.tsx try to fetch some data for the API

- If successful replace the dummy data with original

  Hint: Since the API requires some search string, use whatever initial title of movie you want

Building and deploying a React app

- Building an app bootstrapped with create-react-app is a one-command deal: you only need to run “npm run build” in the project directory.

- In turn, that provides us with a directory named build, which happens to be everything we need to deploy the app to a server.

- We can upload the folder directly to a host such as netlify to show it live!

- For every new feature that we build after the deploy we can just repeat the process.

- Build and upload again

Exercise IX

- Make the Movie Card clickable
  - It should redirect to “/details/[movieId]”

Exercise X

- In Details component, read query parameter that we provided in previous exercises, and based on the movie id from query param send fetch request to

  http://www.omdbapi.com/?i=[moveiId]&plot=full&apikey=[YourApiKey]

- Out of this API, create movie details and display more data about the movie, add the types also

- Feel free to style it yourself

Rebuilding and redeploying

- To rebuild, just run “npm run build” again.
- To redeploy, just replace the existing folder with the new build folder :)