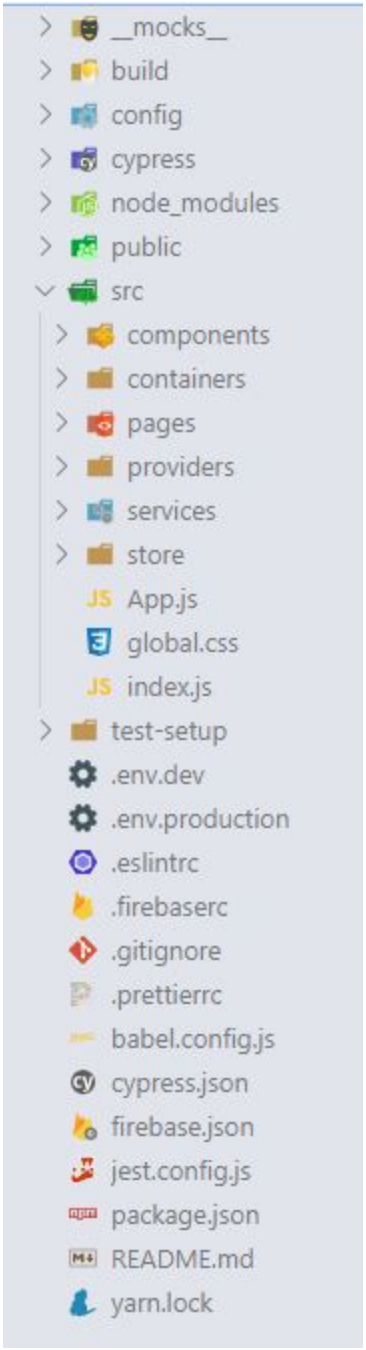
Github Explorer SPA

Short explanation about the assignment

Technologies used:

- Webpack, Babel
- Eslint, Prettier
- React, Redux, ReduxThunk, React router
- Material UI
- Github API and Firebase Auth
- Jest + React Testing Library
- Cypress + Cypress Testing Library



Folder structure:

The application's source code is laid out in the following structure (see image on right):

- **config** - webpack configuration

Custom webpack configuration files that include a common configuration that extends to a dev or production environment depending on the mode. The production configuration covers minification, optimization, and code splitting issues. It also includes configuration for addons for analyzing build code. Webpack outputs the built code in the **build** folder.

- **src** - where all the components and services are stored
    - components - stateless and presentational components
    - containers - wrappers for the components where all the logic and integration is done
    - pages - client-side route pages
    - providers - folder for custom react context providers (currently only has an Auth provider)
    - services - API services, browser storage and helpers
    - store - Redux store with reducers and actions
- **test-setup** - setup and mock data for jest testing
- **cypress** - setup and e2e tests for cypress runner

Application Functionalities

The application has two pages. The first one being the home page where the user can search for Github repositories. Once the user searches for a query, the application fetches data from the Github API and stores this data (per page) in the redux store, along with other information like the pages (which are received from the response header link) and the API status (loading, error, success). The pages show 9 results per page and have a pagination included at the bottom. Once a page is changed another fetch is done with the next set of data, which is again stored in the redux store.

The user can click on one of the results and the second page will open with more details about the repository as well as part of the readme. This page initially displays data from the redux store. If no data is found from the store, it fetches data from the Github API (and puts them in the store). This page has the following route '/facebook/react' which is the same as the API link that the Github API exposes per repo. This is how the fetching is done if no results exist in the store. This is done so in case the user refreshes the whole page (the redux store is empty) data can be fetched and displayed correctly.

Testing

I did different types of tests, snapshots, integration with mocking an API, testing a reducer, and an action, as well as an e2e test. These can be expanded to get full coverage of the application, but for time purposes I wanted to cover different types of tests. For unit and integration tests I used Jest with React Testing Library and for the e2e I used cypress.

Authentication

In order to see your private repos, you need to be authenticated with the Github API.
Authentication with the API works in two steps.

1. GET 'https://github.com/login/oauth/authorize' to get a temporary code
2. POST 'https://github.com/login/oauth/access_token' to exchange the code with an access token.

And finally, you send the access code as an Authentication header within your requests.

The first link takes you to a GitHub page where you can click to authorize your app on GitHub. Here you provide a callback that should exchange the temporary code with an access token.
For this, I used Firebase Authentication that offers a window for the authorization in GitHub and handles the callback. In return, it gives the access token that is kept in local storage. The GitHub API tokens as far as I could find do not expire but can be revoked through your GitHub settings for each app. To build out the authentication in react, I used a context provider that holds the token (from local storage) and exposes login and logout functions (from firebase). In the application, I use these with useContext() hook to provide the token to the requests and the functions to the header component.

Localization
Localization is done with the following library https://formatjs.io/docs/react-intl.
React-intl provides an IntlProvider to which you can provide a locale and packed translations. In the code after that wherever is needed I use useIntl hook to get the messages for the current locale. The provided locale is from the navigator object.