**Step Two: Make an API Helper**

* ApplyToJob function???? How does it work
* How does the API call been made - it is made thru the request() function, in the await axios() call)

// Send a POST request

Await axios({

method: 'post',

url: '/user/12345',

data: {

firstName: 'Fred',

lastName: 'Flintstone'

}

});

<https://www.npmjs.com/package/axios>

Step Three: Make Your Routes File

* Homepage.js: which file is the UserContext been passed over to Homepage.js?

/login

Login/signup

/signup

Signup form

/profile

Edit profile page

Step Four: Companies & Company Detail

CompanyDetail: showing detail on a company,

CompanyList: showing the list of all companies,

CompanyCard: showing simple info about a company on the list

SearchForm: Make your companies list have a search box, which filters companies to those matching the search (remember: there’s a backend endpoint for this!). Do this filtering in the backend — not by loading all companies and filtering in the front end!

## **Step Five: Jobs**

Similarly, flesh out the page that lists all jobs, and the “job card”, which shows info on a single job. You can use this component on both the list-all-jobs page as well as the show-detail-on-a-company page.

Don’t worry about the “apply” button for now — you’ll add that later, when there’s authentication for the app.

- where (when) is title variable passed into the search() function in JobList.js?

- appliedJob is stored in useContext, where is Context.Provider at? (auth folder)

- setApplied(hasAppliedToJob(id)); ?????

**Step Six: Current User**

Add user log in, sign up, and log out, use the backend routes design for authentication and registration. When the user logs in or registers, retrieve information about that user and keep track of it somewhere easily reached elsewhere in the application.

Things to do:

- Make forms for logging in and signing up

- In the navigation, show links to the login and signup forms if a user is not currently logged in.

- If someone is logged in, show their username in the navigation, along with a way to log out.

- Have the homepage show different messages if the user is logged in or out.

When you get a token from the login and register processes, store that token on the JoblyApi class, instead of always using the hardcoded test one. You should also store the token in state high up in your hierarchy; this will let use use an effect to watch for changes to that token to kick off a process of loading the information about the new user.

For example, in the LoginForm component, its better design that this doesn’t handle directly all of the parts of logging in (authenticating via API, managing the current user state, etc). The logic should be more centrally organized, in the App component or a specialized component.

While writing this, your server will restart often, which will make it tedious to keep typing in on the login and signup forms. A good development tip is to hardcode suitable defaults onto these forms during development; you can remove those defaults later.

- Where was login function passed to LoginForm?

**Step Seven: Using localStorage and Protecting Routes**

If the user refreshes their page or closes the browser window, they’ll lose their token. Find a way to add localStorage to your application so instead of keeping the token in simple state, it can be stored in localStorage. This way, when the page is loaded, it can first look for it there.

Be thoughtful about your design: it’s not great design to have calls to reading and writing localStorage spread around your app. Try to centralize this concern somewhere.

As a bonus, you can write a generalized useLocalStorage hook, rather than writing this tied specifically to keeping track of the token.

Protecting Routes

Once React knows whether or not there’s a current user, you can start protecting certain views! Next, make sure that on the front-end, you need to be logged in if you want to access the companies page, the jobs page, or a company details page.

- where is useLocalStorage.js called?

**Step Eight: Profile Page**

Add a feature where the logged-in user can edit their profile. Make sure that when a user saves changes here, those are reflected elsewhere in the app.

- currentUser is stored in useContext, where is Context.Provider at? @App

**Step Nine: Job Applications**

A user should be able to apply for jobs (there’s already a backend endpoint for this!).

On the job info (both on the jobs page, as well as the company detail page), add a button to apply for a job. This should change if this is a job the user has already applied to.

**Step Ten:**

**App.js**

**(JoblyApi)**

let token = await JoblyApi.signup(signupData);