- ⏱️ 5 hours
- ✅ Completed

Pokedex Thunk

In today's project you will connect an existing application to use thunk actions.

Phase 0: Getting started

You'll need the backend for the Pokedex application. Take a moment to clone it from https://github.com/appacademy-starters/pokedex-backend and get it set up.

The API for the backend is also documented in repository's README.

Follow the instructions to setup the backend server in the backend repo's README, then start the backend server by running `npm start`.
Once you have that up and running, clone the frontend starter from https://github.com/appacademy-starters/react-hooks-pokedex-starter.
Run `npm start` in the frontend starter to start your frontend development server.

Explore the reference application

The current application is comprised of the following components:

- `App`: Does the browser routing
- `PokemonBrowser`: The browser that draws the list on the left and has a route to the `PokemonDetail` when the route matches "/pokemon/:pokemonId"
- `PokemonDetail`: Makes a fetch to the API on mount and update to load the details of the selected Pokemon
- `Fab`: The "+" button that prompts the `CreatePokemonForm` to show
- `CreatePokemonForm`: Create Pokemon form rendered on `PokemonBrowser`
- `EditPokemonForm`: Edit Pokemon form rendered on the `PokemonDetail` component only if the Pokemon is captured
- `PokemonItems`: Renders the list of items on the `PokemonDetail` component
- `EditItemForm`: Edit item form rendered on the `PokemonDetail` component when editing an item

Take the time to review over the components to see how the component tree is structured (parent-child relationships and where each component is being used).

Proxy

In this project, you will run two servers using these addresses: `http://localhost:3000` on your frontend and `http://localhost:5000` on your backend. You will make api calls from your frontend to your backend server. When making api calls to your backend, you will want to write your fetch calls like this: `fetch('/api/pokemon')` instead of having to write out your baseURL for every call. In the `package.json` file on your frontend, notice the `"proxy":"http://localhost:5000`. This is telling the development server to proxy any unknown requests to your backend server port. So you must always coordinate your `PORT` variable in your backend `.env` file to have the same port number as the proxy setting in your frontend `package.json`. Remember. This approach only works in development using `npm start`

Phase 1: Dispatch thunk actions in `PokemonBrowser`

As you're connecting your application's components, you'll most likely hit bugs and break your application. While you're connecting each component, make sure to test that your connected code is working before moving on to connect the next component.

There is a thunk action creator made for you already in the `src/store/pokemon.js` file called `getPokemon`. The thunk action it returns fetches all the Pokemon as a list from the `GET /api/pokemon` backend API route. Then it dispatches the action returned from the `load` action creator in the same file. The reducer normalizes the Pokemon data. Dispatch the thunk action returned from the `getPokemon` thunk action creator after the `PokemonBrowser` component first renders.

If done correctly, you should see the list of all the Pokemon in the side of the browser.

Phase 2: Create and dispatch a thunk action for `PokemonDetail`

Create a thunk action creator for fetching a single Pokemon's details based on their `id` by hitting the `GET /api/pokemon/:id` backend API route. Using the data returned from that, dispatch the return of the `addOnePokemon` action creator. Dispatch the thunk action you just created whenever the `pokemonId` in the `PokemonDetail` component changes.

Phase 3: `CreatePokemonForm`

Create a thunk action creator for creating a Pokemon in the `CreatePokemonForm`. The thunk action creator should hit the `POST /api/pokemon` backend API route. Format the fetch request to have a `Content-Type` header of `application/json` and the correct request body using the submitted form information.

After the response comes back, add the newly created Pokemon to the Redux store by dispatching the appropriate regular POJO action.

Dispatch the thunk action you just created on the submission of the `CreatePokemonForm`.

Phase 4: `EditPokemonForm`

Create a thunk action creator for editing a Pokemon in the `EditPokemonForm`. Check out the API docs for which route to hit and how to format the URL path and the request body in the fetch request. After the response comes back, add the updated information Pokemon to the Redux store by dispatching the `addOnePokemon` action.
Dispatch the thunk action you just created on the submission of the `EditPokemonForm`.

Phase 5: `PokemonItems`

The thunk actions in the following phases will be created in the `src/store/items.js` file.
Create a thunk action creator for fetching the items for a single Pokemon based on the `id` of the Pokemon in the `PokemonItems` component. Check out the API docs for which route to hit and how you should format the URL path for this information. After the response comes back, use the data to dispatch the return of the `load` action creator for items.
Dispatch the thunk action you just created when the `id` of the Pokemon changes in the `PokemonItems` component.

Phase 6: `EditItemForm`

Create a thunk action creator for editing an item in the `EditItemForm`. Check out the API docs for which route to hit and how to format the URL path and the request body in the fetch request. After the response comes back, use the data to dispatch the return of the `update` action creator for items.
Dispatch the thunk action you just created on the submission of the `EditItemForm`.
Did you find this lesson helpful?
No
Yes
✓

Continue To Next Page →

Great work, We have received Your Submission and marked this task as complete.

Click continue to move on to the next task!