TypeScript Chirper

The purpose of this lab is to build your "half stack" (frontend, server backend, but no database yet!) Chirper lab. You'll use a React TypeScript frontend with an Express TypeScript backend. The backend will be utilizing `chirpsstore.js` from the Express API Lab along with your route logic.

Steps

Initial Setup and API

1. Use the barebones template to get started
2. Use `chirpsstore.js` from the Express API Lab. You can leave it as a `.js` file or convert it to `.ts` if you wish
3. Copy over or rewrite the same `chirps.js` route logic from the Express API Lab to:
    - GET all chirps
    - GET one chirp
    - POST add a new chirp
    - PUT edit a chirp
    - DELETE delete a chirp
    - *Note:* Convert this to a `.ts` file and update your `require()` syntax to `import/export` if you copy your code over
4. Make sure all route endpoints work in Postman

Frontend

1. Create a React frontend that uses `App.tsx` as your main routing component, where your `BrowserRouter` and `Switch` should be located
2. Your Chirper app should show a list of all chirps on the home screen
3. Your app should have a page with a form to `POST` new chirps with using an `onClick` handler on a button element
4. Each chirp in the list should have a button, `Admin Options`, that will go to a page that specializes in displaying a single chirp
5. On the page for a single chirp, you should make a `GET` request for the specific chirp. Display the chirp information on the page in a pre-filled form (see demo at the bottom of this lab if you're confused). Have a Delete button on the page that will trigger a `DELETE` request to the API for that chirp. Send the user back to the main page when the chirp is successfully deleted. You should also have an Save Edit button that will trigger a `PUT` request to the API for that chirp. This should also send the user back to the main page when the chirp is successfully edited

Hints

- Use the Fetch API in your front-end code for making all your API requests (`GET`, `POST`, `PUT`, `DELETE`)
    - See MDN, specifically "Uploading JSON Data" in the "Making Fetch Requests" section
- Make sure to use `RouteComponentProps` from `react-router-dom` as a generic type in order to access the `history` and `match` props on your components that are routed to
- You will find frontend route params and `this.props.match.params` to be useful in this lab
- Recommended frontend paths are as follows:
    - `/` for the main page that displays the list of chirps and a form
    - `/chirp/:id/admin` for the page that displays a chirp edit form
    - `/chirp/add` for the page to add a new chirp
- Any component that is presented by the Router (e.g. your "pages") will have access to `this.props.history`. This is necessary to kick off navigation from your code. (in response to something being completed, etc.)
    - `this.props.history.push('/something')` allows you to navigate to the page that responds to the path `/something`
    - `this.props.history.replace('/something')` can be used to navigate to that path, but not keep a record of where we currently are (we are replacing the current browser history entry, with this new page we are going to. This is useful if we don't want someone to be able to click the back button and return to this page)
    - `this.props.history.goBack()` can be used to navigate back one page in the browser history

Demo

Lab Demo