21.3.4 Install and Set Up Apollo Client

Even with a GraphQL-powered API, you can still use the browser's built-in Fetch API to make requests. Doing so would leverage a skill you already have, but it also would complicate the process and probably feel unnatural in the context of a React application.

Instead, we'll use Apollo Boost, a library that Apollo created for making requests to GraphQL APIs using Apollo Server. We can use this library anywhere, but some of its features were specifically designed to support managing state for a React front end. Luckily, we're building a React front end, so let's get it installed!

Navigate to the `client` directory from the command line and terminate the development server so that we can install some libraries with npm. To make everything work in unison with Apollo Boost, we'll need more than one library. Install them with the following command, and then we'll discuss each one:

```
npm i apollo-boost graphql graphql-tag @apollo/react-hooks
```

We'll use all of the following libraries together to create the overall client-side functionality:

- A variation of the standard Apollo Client library that also includes other helper libraries, `apollo-boost` will serve as the means of making requests to the server.

- GraphQL (`graphql`) is a dependency much like MySQL2 was for Sequelize. We don't use it directly, but it needs to be present for the other libraries to work.

- `graphql-tag` is a package that will help parse the client-side tagged template literal statements for queries and mutations.

- `@apollo/apollo-hooks` is Apollo's library for using React Hooks functionality with Apollo queries and mutations. We have to include the `@apollo/` prefix because `apollo-hooks` is never used on its own—only alongside other Apollo libraries.

DEEP DIVE

DEEP DIVE

To learn more, read the **Apollo documentation on Apollo Boost (https://www.apollographql.com/docs/react/get-started/#apollo-boost)**.

Now that we have these libraries in place, let's integrate them!

Create Apollo Provider

To begin integrating Apollo into the front end of the application, navigate to `App.js` in the `client` directory. Add the following import statements to the top of the file:

```
import React from 'react';

// add these two library import statements
import { ApolloProvider } from '@apollo/react-hooks';
import ApolloClient from 'apollo-boost';
```

So we just imported two key pieces to the application. The first, `ApolloProvider`, is a special type of React component that we'll use to provide data to all of the other components. We'll use the second, `ApolloClient`, to get that data when we're ready to use it.

IMPORTANT

> Provider components are not unique to Apollo. In fact, Apollo Provider is just a specialized use case of a built-in React tool. You'll learn more about it in your next project!

Next, we need to establish the connection to the back-end server's `/graphql` endpoint. Before we do that, let's consider how. What's the address of the current React client's server? What about the back-end server?

Add the following code right above the `App` function declaration in `App.js`:

```
const client = new ApolloClient({
  uri: 'http://localhost:3001/graphql'
});
```

With this code, we establish a new connection to the GraphQL server using Apollo. We could pass many other options and configuration settings into this constructor function. We'll use another one down the road, but for now this is all we need.

Notice how we have to use an absolute path to the server? The React environment runs at `localhost:3000`, and the server environment runs at `localhost:3001`. So if we just used `/graphl`, as we've done previously, the requests would go to `localhost:3000/graphql`—which isn't the address for the back-end server. We'll touch more on this soon; for now, let's get up and running.

Lastly, we need to put the two imported libraries to use in the `App` component. Let's update the returning JSX code in the `App` function as follows:

```
function App() {
  return (
    <ApolloProvider client={client}>
      <div className="flex-column justify-flex-start min-100-vh">
        <Header />
        <div className="container">
          <Home />
        </div>
        <Footer />
      </div>
    </ApolloProvider>
  );
}
```

Note how we wrap the entire returning JSX code with `<ApolloProvider>`. Because we're passing the `client` variable in as the value for the `client` prop in the provider, everything between the JSX tags will eventually have access to the server's API data through the `client` we set up.

You'll soon learn more about the purpose of Apollo Provider, but there are no front-end queries in place just yet. Time to get the homepage to print out everyone's thoughts!