Shortcomings of GraphQL in React without a GraphQL Client library

Before we move on to the Apollo Client, we discuss why it is important to use GraphQL in React with a GraphQL client library rather than without one.

We implemented a simple GitHub issue tracker that uses React and GraphQL without a dedicated library for GraphQL, using only axios to communicate with the GraphQL API with HTTP POST methods. I think it is important to work with raw technologies, in this case GraphQL, using plain HTTP methods, before introducing another abstraction. The Apollo library offers an abstraction that makes using GraphQL in React much easier, so you will use Apollo for your next application. For now, using GraphQL with HTTP has shown you two important things before introducing Apollo:

- How GraphQL works when using a puristic interface such as HTTP.
- The shortcomings of using no sophisticated GraphQL Client library in React, because you have to do everything yourself.

Before we move on, I want to address the shortcomings of using puristic HTTP methods to read and write data to your GraphQL API in a React application:

- **Complementary**: To call a GraphQL API from your client application, use HTTP methods. There are several quality libraries out there for HTTP requests, one of which is axios. That's why you have used axios for the previous application. However, using axios (or any other HTTP client library) doesn't feel like the best fit to complement a GraphQL centered interface. For instance, GraphQL doesn't use the full potential of HTTP. It's just fine to default to HTTP POST and only one API endpoint. It doesn't use resources and methods on those resources like a RESTful interface, so it makes no sense to specify an HTTP method and an API endpoint with every request, but to set it up once in the beginning instead. GraphQL comes with its own constraints. You could see it as a layer on top of HTTP when it's not as important for a developer to know about the underlying

HTTP.

- **Declarative**: Every time you make a query or mutation when using plain HTTP requests, you have to make a dedicated call to the API endpoint using a library such as axios. It's an imperative way of reading and writing data to your backend. However, what if there was a declarative approach to making queries and mutations? What if there was a way to co-locate queries and mutations to your view-layer components? In the previous application, you experienced how the query shape aligned perfectly with your component hierarchy shape. What if the queries and mutations would align in the same way? That's the power of co-locating your data-layer with your view-layer, and you will find out more about it when you use a dedicated GraphQL client library for it.

- **Feature Support**: When using plain HTTP requests to interact with your GraphQL API, you are not leveraging the full potential of GraphQL. Imagine you want to split your query from the previous application into multiple queries that are co-located with their respective components where the data is used. That's when GraphQL would be used in a declarative way in your view-layer. But when you have no library support, you have to deal with multiple queries on your own, keeping track of all of them, and trying to merge the results in your state-layer. If you consider the previous application, splitting up the query into multiple queries would add a whole layer of complexity to the application. A GraphQL client library deals with aggregating the queries for you.

- **Data Handling**: The naive way for data handling with puristic HTTP requests is a subcategory of the missing feature support for GraphQL when not using a dedicated library for it. There is no one helping you out with normalizing your data and caching it for identical requests. Updating your state-layer when resolving fetched data from the data-layer becomes a nightmare when not normalizing the data in the first place. You have to deal with deeply nested state objects which lead to the verbose usage of the JavaScript spread operator. When you check the implementation of the application in the GitHub repository again, you will see that the updates of React's local state after a mutation and query are not nice to look at. A normalizing library such as normalizr could help you to improve the structure of your local state. You learn more about normalizing your state in the book Taming the State in React. In addition to a lack of caching and normalizing support, avoiding libraries means missing out on functionalities for pagination and optimistic updates. A dedicated GraphQL library makes all these features available to you.

- **GraphQL Subscriptions**: While there is the concept of a query and mutation to read and write data with GraphQL, there is a third concept of a GraphQL subscription for receiving real-time data in a client-side application. When you would have to rely on plain HTTP requests as before, you would have to introduce WebSockets next to it. It enables you to introduce a long-lived connection for receiving results over time. In conclusion, introducing GraphQL subscriptions would add another tool to your application. However, if you would introduce a GraphQL library for it on the client-side, the library would probably implement GraphQL subscriptions for you.

I am looking forward to introducing Apollo as a GraphQL client library to your React application. It will help with the aforementioned shortcomings. However, I do strongly believe it was good to learn about GraphQL in React without a GraphQL library in the beginning.

You can find the final repository on GitHub. The repository showcases most of the exercise tasks too. The application is not feature complete since it doesn't cover all edge cases and isn't styled. However, I hope the implementation walkthrough with plain GraphQL in React has helped you to understand using only GraphQL client-side in React using HTTP requests. I feel it's important to take this step before using a sophisticated GraphQL client library such as Apollo or Relay.

I've shown how to implement a React application with GraphQL and HTTP requests without using a library like Apollo. Next, you will continue learning about using GraphQL in React using Apollo instead of basic HTTP requests with axios. The Apollo GraphQL Client makes caching your data, normalizing it, performing optimistic updates, and pagination effortless.