Introduction to Writing GraphQL App with React

Learn how to use GraphQL in React with HTTP.

WE'LL COVER THE FOLLOWING

The previous chapter was a basic introduction to GraphQL while in this chapter, we will learn how to combine React with GraphQL. There is no smart library like Apollo Client or Relay to help get started, so instead, we will have to perform GraphQL queries and mutations with basic HTTP requests.

Along the way, we will build a simplified **GitHub client**, basically an ***issue tracker*** for GitHub, that consumes GitHub's GraphQL API. We will perform GraphQL queries and mutations to read and write data, and by the end, we will be able to showcase a **GraphQL in React example** that can be used by other developers as a learning tool. The code snippets along the way will guide you in your development.

As a reference, the final application we are going to build can be found here.

Getting Started

Let's get started with the App component in `app.js`.

```
import React from 'react';

import ReactDOM from 'react-dom';
import App from './app.js';

ReactDOM.render(
  <App />
```

```
  document.getElementById('root')
);
```

The component only renders a `title` (**React GraphQL GitHub Client**) as a headline. Before implementing any more React components, we need a library to handle GraphQL requests. This is vital as it will enable us to execute queries and mutations and to use an HTTP POST method. For this, we will use ***axios***.

Configuring Axios

Axios is perfect for the following application because we want to configure it only once with our personal access token and GitHub's GraphQL API. We will now import `axios` next to our App component and configure it.

First, define a base URL for `axios` when creating a configured instance from it. As mentioned before, we don't need to define GitHub's URL endpoint every time we make a request because all queries and mutations point to the same URL endpoint in GraphQL. Instead, you get the flexibility from your query and mutation structures using objects and fields.

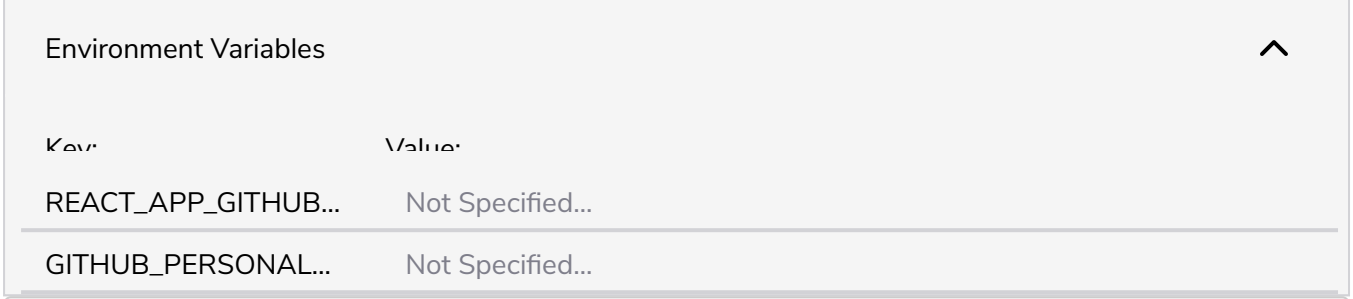
Environment Variables

Key:	Value:
REACT_APP_GITHUB...	Not Specified...
GITHUB_PERSONAL...	Not Specified...

```
import React, { Component } from 'react';
import axios from 'axios';

const axiosGitHubGraphQL = axios.create({
  baseURL: 'https://api.github.com/graphql',
});
...
export default App;
```

Now, we can pass the personal access token as a header to the configuration. The header is used by each request made with this `axios` instance. Since we have already entered our personal access token previously in the course's environment, we will just pass that environment variable to our `axios` configuration with string interpolation (template literals) to create a configured `axios` instance.




```
...
const axiosGitHubGraphQL = axios.create({
  baseURL: 'https://api.github.com/graphql',
  headers: {
    Authorization: `bearer ${
      process.env.REACT_APP_GITHUB_PERSONAL_ACCESS_TOKEN
    }`,
  },
});
...
```

Setting up a Form to Capture Details

Now, we will set up a form for capturing details about a **GitHub organization and repository** from a user. By filling out an input field, we can request a paginated list of issues for a specific GitHub repository. The input field within a `<form>` will allow us to enter the organization and repository to track their issues. Once the organization and the repository are entered, the component has to update React's local state.

Secondly, the form needs a **submit button** to generate a request for the data about the organization and repository that the user provided in the input field, which has been updated in the component's local state.

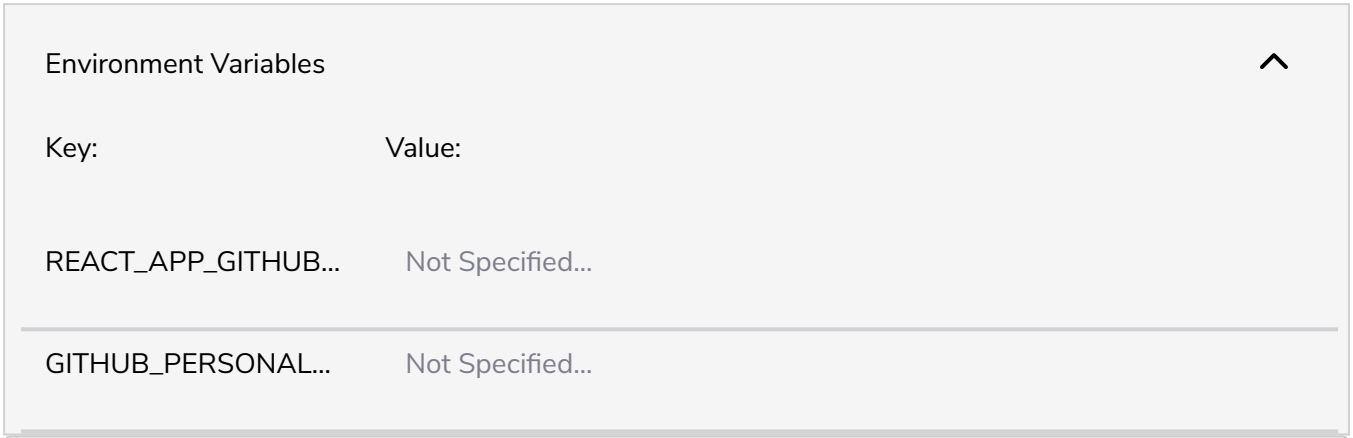
Third, it would be convenient to have an initial local state for the organization and repository so we can make a request with that initial data when the component mounts for the first time.

Let's tackle implementing this scenario in the following steps:

1. The render method has to render a form with an input field.
2. The form has to have an `onSubmit` handler.
3. The input field needs an `onChange` handler.
4. The input field uses the `path` from the local state as a value to be a controlled component.
5. The `path` value in the local state updates in the `onChange` handler.

The following code snippet shows the implementation of the first three steps:

The following code snippet shows the implementation of the first three steps:




```
class App extends Component {
  render() {
    return (
      <div>
        <h1>{TITLE}</h1>
        <form onSubmit={this.onSubmit}> //form with onSubmit handler
          <label htmlFor="url">
            Show open issues for https://github.com/
          </label>
          <input //input field with onChange handler
            id="url"
            type="text"
            onChange={this.onChange}
            style={{ width: '300px' }}
          />
          <button type="submit">Search</button>
        </form>
        <hr /> //breaking the user-input part and the results part
        {/* Here comes the result! */}
      </div>
    );
  }
}
```

Next, we declare the class methods to be used in the render method and implement the last two steps:

```
import React from 'react';

import ReactDOM from 'react-dom';
import App from './app.js';

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

When you start the application, you should see the initial state for the `path` in the input field. You should be able to change the state by entering something else in the input field, but nothing happens with `componentDidMount()` and

submitting the form yet.

The `componentDidMount()` lifecycle method can be used to make an initial request when the App component mounts. There needs to be an initial state for the input field to make an initial request in this lifecycle method.

> **Note**: The previous implementation uses a React class component syntax that you might not have used before. If you are not familiar with it, give a look at this GitHub repository to gain more understanding of React's class component syntax. *Using class field declarations lets you omit the constructor statement for initializing the local state, and eliminates the need to bind class methods. Instead, arrow functions will handle all the binding.*

The following lines of code make the input field a controlled component. The input element shouldn't be used to handle its internal state using native HTML behavior; it should be React.

```
const { path } = this.state;
...
<input
  id="url"
  type="text"
  value={path}
  onChange={this.onChange}
  style={{ width: '300px' }}
/>
```

The previous setup for the form–using input field(s), a submit button, `onChange()` and `onSubmit()` class methods–is a common way to implement forms in React. The only addition is the initial data fetching in the `componentDidMount()` lifecycle method to improve user experience by providing an initial state for the query to request data from the backend. It is a useful foundation for fetching data from a third-party API in React.

You might wonder why there is only one input field to grab the information about the organization and repository. When opening up a repository on GitHub, you can see that the organization and repository are encoded in the URL, so it becomes a convenient way to show the same URL pattern for the input field. You can also split the `organization/repository` later at the `/` to get

these values and perform the GraphQL query request.

Now that we have set up a form in React to get input from a user, we will further move on to generating a GraphQL query request using `axios` in the next lesson.

Exercise

1. Confirm your source code for the last section