GraphQL Query with Apollo Client in React

Let's implement and execute GraphQL queries!

WE'LL COVER THE FOLLOWING

In this lesson, we will implement our first GraphQL query using Apollo Client in React!

You've previously seen how different entities, such as the current user (viewer) or repositories, can be queried from GitHub's GraphQL API. This time you will do it in React.

A `Profile` component might be the best place to render the current user and its associated repositories. We'll start by using the not-yet-implemented `Profile` component in our `App` component in the *src/App/index.js* file, which we'll take care of next. It makes sense to extract the `Profile` component now because the `App` component will be the static frame around the application later.

Components like `Navigation` and `Footer` are static, and components such as `Profile` and `Organization` (which you'll see later) are dynamically rendered based on routing (URLs).

Environment Variables

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

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

```
import Profile from '../Profile';

class App extends Component {

  render() {
    return <Profile />;
  }
}

export default App;
```

src/App/index.js

In our *src/Profile/index.js* file, we have added a simple functional stateless component. In the next step, we will extend it with a GraphQL query.

Environment Variables

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

```
import React from 'react';

const Profile = () =>
  <div>Profile</div>

export default Profile;
```

src/Profile/index.js

Now we'll learn to query data with GraphQL and Apollo Client. The Apollo Client was added in a previous lesson with React's Context API in a top-level component. We have implicit access to the Apollo Client but we never use it directly for standard queries and mutations. It says *"standard"* here because there will be situations where we will use the Apollo Client instance *directly* while implementing this application.

The **React Apollo** package grants access to a `Query` component, which takes a query as a prop and executes it when it is rendered.

That's the important part: it executes the query when it is rendered!

It uses React's render props pattern which uses a child as a function implementation where we can access the result of the query as an argument.

Environment Variables

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

```
import React from 'react';
import { Query } from 'react-apollo';

const Profile = () => (
  <Query query={}>
    {() => <div>My Profile</div>}
  </Query>
);

export default Profile;
```

src/Profile/index.js

This is a function that returns only JSX, but we have access to additional information in the function arguments. Moving on, at first, we will define the GraphQL query to request our authorizations using the previously installed utility package (graphql-tag) to define the query.

Environment Variables

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

```
import React from 'react';
import gql from 'graphql-tag';
import { Query } from 'react-apollo';

const GET_CURRENT_USER = gql`
  {
    viewer {
      login
      name
    }
  }
`;

const Profile = () => (
  <Query query={GET_CURRENT_USER}>
    {() => <div>My Profile</div>}
  </Query>
);

export default Profile;
```

src/Profile/index.js

Now, we will use the children as a function pattern to retrieve the query result as a data object and render the information in our JSX.

Environment Variables

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

```
import React from 'react';
import gql from 'graphql-tag';
import { Query } from 'react-apollo';

const GET_CURRENT_USER = gql`
  {
    viewer {
      login
      name
    }
  }
`;

const Profile = () => (
  <Query query={GET_CURRENT_USER}>
    {({ data }) => {
      const { viewer } = data;

      return (
        <div>
          {viewer.name} {viewer.login}
        </div>
      );
    }}
  </Query>
);

export default Profile;
```

src/Profile/index.js

Also, we have to make sure to give some type of visual feedback until our view-layer can be rendered with actual data:

Environment Variables

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

```
const Profile = () => (
  <Query query={GET_CURRENT_USER}>
    {({ data }) => {
      const { viewer } = data;

      if (!viewer) {
        return null;
      }

      return (
        <div>
          {viewer.name} {viewer.login}
        </div>
      );
    }}
  </Query>
);
```

src/Profile/index.js

That's how you define a GraphQL query in a declarative way in React. Once the `Query` component renders, the request is executed. The Apollo Client, provided in a top-level component, is used to perform the query. The render props pattern makes it possible to access the result of the query in the child function.

Run the application below to check if you can see your profile or not:

Environment Variables

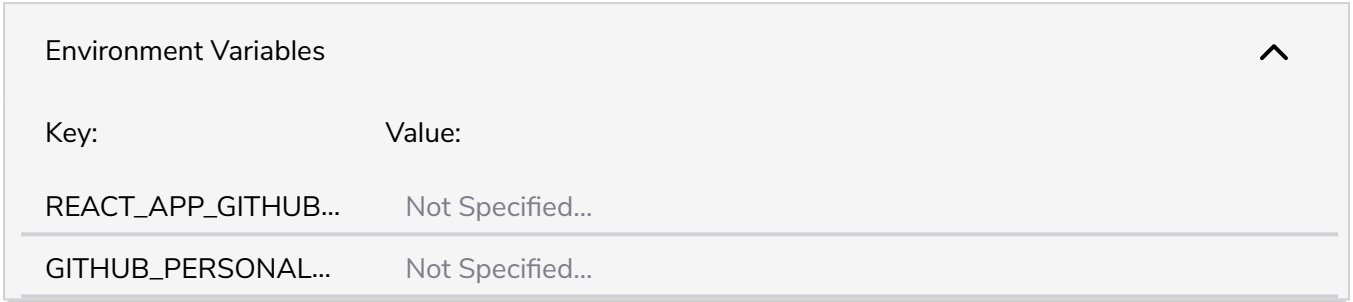
Key:	Value:
REACT_APP_GITHUB...	Not Specified...
GITHUB_PERSONAL...	Not Specified...

```
import React from 'react';

import Link from '../../Link';

import './style.css';

const Footer = () => (
  <div className="Footer">
    <div>
      <small>
        <span className="Footer-text">Built by</span>{' '}
        <Link
          className="Footer-link"
          href="https://www.robinwieruch.de"
        >
          Robin Wieruch
        </Link>{' '}
        <span className="Footer-text">with &hearts;</span>
```

```
      </small>
    </div>
    <div>
      <small>
        <span className="Footer-text">
          Interested in GraphQL, Apollo and React?
        </span>{' '}
        <Link
          className="Footer-link"
          href="https://www.getrevue.co/profile/rwieruch"
        >
          Get updates
        </Link>{' '}
        <span className="Footer-text">
          about upcoming articles, books &
        </span>{' '}
        <Link className="Footer-link" href="https://roadtoreact.com">
          courses
        </Link>
        <span className="Footer-text">.</span>
      </small>
    </div>
  </div>
);

export default Footer;
```

There is more information found in the render prop function. Check the official React Apollo API for additional information beyond the examples in this application.

Next, let's show a loading indicator when a query is pending:




```
const Profile = () => (
  <Query query={GET_CURRENT_USER}>
    {({ data, loading }) => {
      const { viewer } = data;

      if (loading || !viewer) {
        return <div>Loading ...</div>;
      }

      return (
        <div>
          {viewer.name} {viewer.login}
        </div>
      );
    }}
  </Query>
```

```
);
```

src/Profile/index.js

The application now shows a *loading indicator* when there is no `viewer` object or the `loading` boolean is set to true. As you can assume that the request will be pending when there is no `viewer`, we will show the loading indicator from the beginning. At this point, it's best to extract the loading indicator as its own component because we will have to reuse it later for other queries.

The `Loading` component will be placed in the Loading folder, that is in *src/Loading/index.js* file. Then, we can use it in our Profile component.

Environment Variables

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

```
import React from 'react';

const Loading = () =>
  <div>Loading ...</div>

export default Loading;
```

src/Loading/index.js

Next, we extend the query with a nested list field for querying our own GitHub repositories. You have done it a few times before. The query structure is not any different now. The following query requests a lot of information which we will use in this application:

Environment Variables

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

```
const GET_REPOSITORIES_OF_CURRENT_USER = gql`
  {
    viewer {
      repositories(
        first: 5
```

```
        orderBy: { direction: DESC, field: STARGAZERS }
      ) {
        edges {

          node {
            id
            name
            url
            descriptionHTML
            primaryLanguage {
              name
            }
            owner {
              login
              url
            }
            stargazers {
              totalCount
            }
            viewerHasStarred
            watchers {
              totalCount
            }
            viewerSubscription
          }
        }
      }
    }
  }
`;
```

src/Profile/index.js

Now we will use this extended and renamed query in our `Query` component placed in the `Profile` component to request additional information about repositories. We pass these repositories from the query result to the new `RepositoryList` component which should do all the rendering for the repositories. It's not the responsibility of the `Profile` component now, and therefore we have to render a list of repositories somewhere else.

Environment Variables

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

```
...

import RepositoryList from '../Repository';
import Loading from '../Loading';

...

const Profile = () => (
```

```
  <Query query={GET_REPOSITORIES_OF_CURRENT_USER}>
    {({ data, loading }) => {
      const { viewer } = data;


      if (loading || !viewer) {
        return <Loading />;
      }

      return <RepositoryList repositories={viewer.repositories} />;
    }}
  </Query>
);
In
```

src/Profile/index.js

In *src/Repository/index.js* file, we create our first import/export statements for the `RepositoryList` component from a dedicated file in this folder. The *index.js* file is used as the entry point to this Repository module. Everything used from this module should be accessible by importing it from this *index.js* file.

Environment Variables

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

```
import RepositoryList from './RepositoryList';

export default RepositoryList;
```

src/Repository/index.js

Next, let's define the `RepositoryList` component in *src/Repository/RepositoryList/index.js file*. The component only takes an array of repositories as props, which will be retrieved by the GraphQL query to render a list of `RepositoryItem` components. The identifier of each repository can be passed as a key attribute to the rendered list. Otherwise, all props from one repository node are passed to the `RepositoryItem` using the *JavaScript spread operator*.

Environment Variables

Key:	Value:

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

```
import React from 'react';

import RepositoryItem from '../RepositoryItem';

import '../style.css';

const RepositoryList = ({ repositories }) =>
  repositories.edges.map(({ node }) => (
    <div key={node.id} className="RepositoryItem">
      <RepositoryItem {...node} />
    </div>
  ));

export default RepositoryList;
```

src/Repository/RepositoryList/index.js

Finally, we'll define the `RepositoryItem` component in the *src/Repository/RepositoryItem/index.js* file to render all the queried information about each repository. The file already uses a couple of stylings which are defined in a CSS file. Otherwise, the component renders only static information for now.

Environment Variables

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

```
import React from 'react';

import Link from '../../Link';

import '../style.css';

const RepositoryItem = ({
  name,
  url,
  descriptionHTML,
  primaryLanguage,
  owner,
  stargazers,
  watchers,
  viewerSubscription,
  viewerHasStarred,
}) => (
  <div>
    <div className="RepositoryItem-title">
      <h2>
        <Link href={url}>{name}</Link>
```

```
      </h2>

      <div className="RepositoryItem-title-action">
        {stargazers.totalCount} Stars
      </div>
    </div>

    <div className="RepositoryItem-description">
      <div
        className="RepositoryItem-description-info"
        dangerouslySetInnerHTML={{ __html: descriptionHTML }}
      />
      <div className="RepositoryItem-description-details">
        <div>
          {primaryLanguage && (
            <span>Language: {primaryLanguage.name}</span>
          )}
        </div>
        <div>
          {owner && (
            <span>
              Owner: <a href={owner.url}>{owner.login}</a>
            </span>
          )}
        </div>
      </div>
    </div>
  </div>
);

export default RepositoryItem;
```

src/Repository/RepositoryItem/index.js

The anchor element to link to the repository is already extracted as a `Link` component. The `Link` component in the *src/Link/index.js* file could look like the following, to make it possible to open those URLs in an extra browser tab:

Environment Variables

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

```
import React from 'react';

const Link = ({ children, ...props }) => (
  <a {...props} target="_blank" rel="noopener noreferrer">
    {children}
  </a>
);

export default Link;
```

src/Link/index.js

Once you run the application below, you should see a styled list of repositories with a name, url, description, star count, owner, and the project's implementation language. If you can't see any repositories, check to see if your GitHub account has any public repositories. If it doesn't, then it's normal if nothing showed up.

I recommend you make yourself comfortable with GitHub by creating a couple of repositories, both for the sake of learning about GitHub and to use this data to practice with this tutorial. Another way to create repositories for your own account is forking repositories from other people.

What we have done in the last steps of this section was pure React implementation, but this is only one opinionated way on how to structure components. The most important part of this section though happens in the `Profile` component. There, we introduced a `Query` component that takes a query as a prop. Once the `Query` component renders, it executes the GraphQL query. The result of the query is made accessible as an argument within React's render props pattern.

Environment Variables

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

```
import React from 'react';

import Link from '../../Link';

import './style.css';

const Footer = () => (
  <div className="Footer">
    <div>
      <small>
        <span className="Footer-text">Built by</span>{' '}
        <Link
          className="Footer-link"
          href="https://www.robinwieruch.de"
        >
          Robin Wieruch
        </Link>{' '}
        <span className="Footer-text">with &hearts;</span>
      </small>
    </div>
```

```
    <div>
      <small>
        <span className="Footer-text">
          Interested in GraphQL, Apollo and React?
        </span>{' '}
        <Link
          className="Footer-link"
          href="https://www.getrevue.co/profile/rwieruch"
        >
          Get updates
        </Link>{' '}
        <span className="Footer-text">
          about upcoming articles, books &
        </span>{' '}
        <Link className="Footer-link" href="https://roadtoreact.com">
          courses
        </Link>
        <span className="Footer-text">.</span>
      </small>
    </div>
  </div>
);

export default Footer;
```

Exercise

1. Confirm your source code for the last section

Reading Task

1. Read more about queries with Apollo Client in React