GraphQL Mutation with Apollo Client in React

We will cover GraphQL Mutations regarding some repository operations using Apollo Client in React in this lesson.

WE'LL COVER THE FOLLOWING

The previous lessons have taught you how to query data with **React Apollo** and the **Apollo Client**. In this lesson, you will learn about mutations. As in other applications before, we have implemented starring a repository with GitHub's exposed `addStar` mutation.

The mutation starts out with a variable to identify the repository to be *starred*. We haven't used a variable in `Query` component yet, but the following mutation works the same way, which can be defined in the *src/Repository/RepositoryItem/index.js* file.

Environment Variables

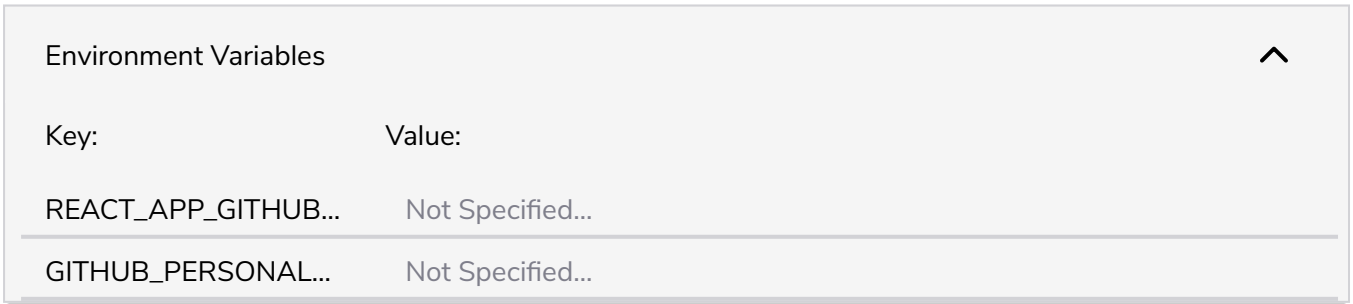
Key:	Value:
REACT_APP_GITHUB...	Not Specified...
GITHUB_PERSONAL...	Not Specified...

```
import React from 'react';
import gql from 'graphql-tag';

...

const STAR_REPOSITORY = gql`
  mutation($id: ID!) {
    addStar(input: { starrableId: $id }) {
      starrable {
        id
        viewerHasStarred
      }
    }
  }
```

```
`;

...
```

src/Repository/RepositoryItem/index.js

The mutation definition takes the `id` variable as input for the `addStar` mutation. As before, we can decide what should be returned in case of a successful mutation and incorporate that into our query. Next, we will use a `Mutation` component that represents the previously used `Query` component, but this time the `Query` component will be used for mutations. You have to pass the mutation prop, but also a variable prop for passing the identifier for the repository.




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

...

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

      <div>
        <Mutation mutation={STAR_REPOSITORY} variables={{ id }}>
          {addStar => <div>{stargazers.totalCount} Star</div>}
        </Mutation>
      </div>
    </div>

    <div className="RepositoryItem-description">
      ...
```

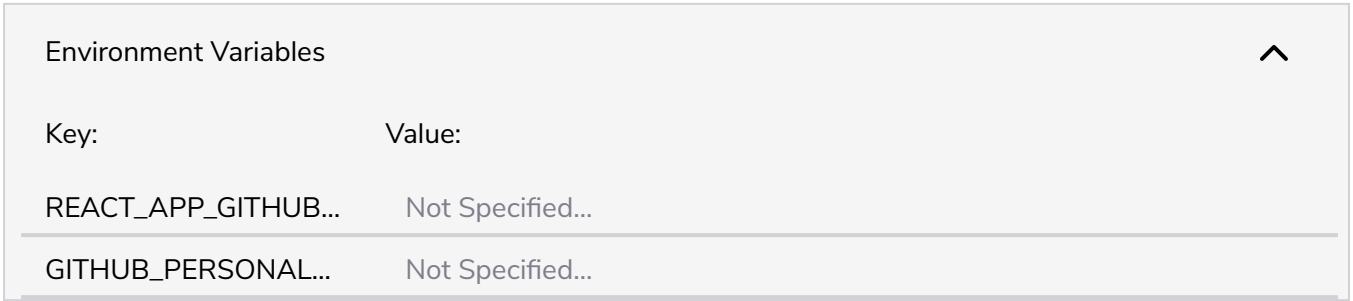
```
    </div>
  </div>
);
```

src/Repository/RepositoryItem/index.js

> **Note:** The div element surrounding the `Mutation` component is there because we will implement other mutations too in this lesson.

The `id` for each repository should be available due to the previous query result. It has to be used as a variable for the mutation to identify the repository.

The `Mutation` component is used like the `Query` component because it implements the render prop pattern as well. The first argument is different, though, as it is the mutation itself instead of the mutation result. We will use this function to trigger the mutation before expecting a result. Later, we will see how to retrieve the mutation result; for now, the mutating function can be used in a button element. In this case, it is already in a `Button` component:




```
...

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

...

const RepositoryItem = ({ ... }) => (
  <div>
    <div className="RepositoryItem-title">
      ...

      <div>
        <Mutation mutation={STAR_REPOSITORY} variables={{ id }}>
          {(addStar) => (
            <Button
              className={'RepositoryItem-title-action'}
              onClick={addStar}
            >
              {stargazers.totalCount} Star
            </Button>
          )}
```

```
      </Mutation>
    </div>
  </div>

  ...
 </div>
);
```

src/Repository/RepositoryItem/index.js

The styled `Button` component could be implemented in the *src/Button/index.js* file. It's already extracted because you will use it in this application later.

Environment Variables

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

```
import React from 'react';

import './style.css';

const Button = ({
  children,
  className,
  color = 'black',
  type = 'button',
  ...props
}) => (
  <button
    className={`${className} Button Button_${color}`}
    type={type}
    {...props}
  >
    {children}
  </button>
);

export default Button;
```

src/Button/index.js

Now, let's get to the mutation result which was left out before. We'll access it as the second argument in our child function of the render prop.

Environment Variables

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

GITHUB_PERSONAL... Not Specified...

```
const RepositoryItem = ({ ... }) => (
  <div>
    <div className="RepositoryItem-title">
      ...

      <div>
        <Mutation mutation={STAR_REPOSITORY} variables={{ id }}>
          {(addStar, { data, loading, error }) => (
            <Button
              className={'RepositoryItem-title-action'}
              onClick={addStar}
            >
              {stargazers.totalCount} Star
            </Button>
          )}
        </Mutation>
      </div>
    </div>

    ...
  </div>
);
```

src/Repository/RepositoryItem/index.js

A mutation works like a query when using React Apollo. It uses the render prop pattern to access the mutation and the result of the mutation. The mutation can be used as a function in the UI. It has access to the variables that are passed in the `Mutation` component, but it can also override the variables when you pass them in a configuration object to the function (e.g. `addStar({ variables: { id } })`). That's a general pattern in React Apollo: You can specify information like variables in the `Mutation` component, or when you call the mutating function to override it.

Note that if you use the `viewerHasStarred` boolean from the query result to show either a **"Star"** or **"Unstar"** button, you can do it with a conditional rendering:

Environment Variables

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

```
const RepositoryItem = ({ ... }) => (
  <div>
    <div className="RepositoryItem-title">
      ...
```

```
      <div>
        {!viewerHasStarred ? (
          <Mutation mutation={STAR_REPOSITORY} variables={{ id }}>
            {(addStar, { data, loading, error }) => (
              <Button
                className={'RepositoryItem-title-action'}
                onClick={addStar}
              >
                {stargazers.totalCount} Star
              </Button>
            )}
          </Mutation>
        ) : (
          <span>{/* Here comes your removeStar mutation */}</span>
        )}

      {/* Here comes your updateSubscription mutation */}
      </div>
    </div>

    ...
  </div>
);
```

src/Repository/RepositoryItem/index.js

When you star a repository as above at this point, the **"Star"** button disappears. This is what we want because it means the `viewerHasStarred` boolean has been updated in Apollo Client's cache for the identified repository. Apollo Client was able to match the mutation result with the repository identifier to the repository entity in Apollo Client's cache, the props were updated, and the UI re-rendered. Yet, on the other side, the count of stargazers who has starred the repository isn't updated because it cannot be retrieved from GitHub's API. The count must be updated the count in Apollo Client's cache.

You will find out more about this topic in one of the following lessons.

Run the code below and play around with the implementations of the following mutations:

- STAR_REPOSITORY
- UNSTAR_REPOSITORY
- WATCH_REPOSITORY

Environment Variables

Key: Value:

REACT_APP_GITHUB... 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;
```

Exercises

1. Confirm your source code for the last section.
2. Implement the `updateSubscription` mutation from GitHub's GraphQL API to watch/unwatch a repository based on the `viewerSubscription` status

Reading Tasks

1. Read more about mutations with Apollo Client in React
2. Implement other mutations in the `RepositoryItem` component.