Apollo Client and a GraphQL Mutation

This lesson will brief about how to go about GraphQL Mutation using Apollo Client.

WE'LL COVER THE FOLLOWING

- Exercise

Previously, you learned how to query data from GitHub's GraphQL API using the Apollo Client. Once the client is set up with a configuration, you can use its `query()` method to send a GraphQL `query` with optional `variables`. As you have learned, reading data with GraphQL is not everything, because there are mutations for writing data as well.

In this lesson, you are going to define a mutation to star a repository on GitHub. The Apollo Client instance sends the mutation, but at first, you have to define it.

Environment Variables

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

```
const ADD_STAR = gql`
  mutation AddStar($repositoryId: ID!) {
    addStar(input: { starrableId: $repositoryId }) {
      starrable {
        id
        viewerHasStarred
      }
    }
  }
`;
```

index.js

The identifier for the repository is required, or GitHub's GraphQL server

wouldn't know which repository you want to star. In the next code snippet, the Apollo Client is used to star a specific GitHub repository with a given identifier. The identifier can be retrieved by adding the `id` field to your repository `node` field in the query. We use the `mutate()` method on the Apollo Client to send the `mutation` in a mutation and `variables` payload. Anything can be done with the result to fit our application, but in this case, the result is simply logged on the console.

Environment Variables

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

```
client
  .mutate({
    mutation: ADD_STAR,
    variables: {
      repositoryId: 'MDEwOlJlcG9zaXRvcnk2MzM1MjkwNw==',
    },
  })
  .then(console.log);
```

The result should be encapsulated in an `addStar` object (the name of the mutation), which should reflect exactly the objects and fields that you have defined in the mutation: `starrable`, `id` and `viewerHasStarred`.

Run the code below to find out the result:

Environment Variables

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

index.js

.ed_required_keys

.ed_keys

```
import 'dotenv/config';
import 'cross-fetch/polyfill';
import ApolloClient, { gql } from 'apollo-boost';
```

```
const client = new ApolloClient({
  uri: 'https://api.github.com/graphql',
  request: operation => {
    operation.setContext({
      headers: {
        authorization: 'Bearer ==GITHUB_PERSONAL_ACCESS_TOKEN==',
      },
    });
  },
});

const GET_REPOSITORIES_OF_ORGANIZATION = gql`
  query($organization: String!, $cursor: String) {
    organization(login: $organization) {
      name
      url
      repositories(
        first: 5
        orderBy: { direction: DESC, field: STARGAZERS }
        after: $cursor
      ) {
        edges {
          node {
            ...repository
          }
        }
        pageInfo {
          endCursor
          hasNextPage
        }
      }
    }
  }
  fragment repository on Repository {
    name
    url
  }
`;

client
  .query({
    query: GET_REPOSITORIES_OF_ORGANIZATION,
    variables: {
      organization: 'the-road-to-learn-react',
      cursor: undefined,
    },
  })
  // resolve first page
  .then(result => {
    const { pageInfo, edges } = result.data.organization.repositories;
    const { endCursor, hasNextPage } = pageInfo;

    console.log('second page', edges.length);
    console.log('endCursor', endCursor);

    return pageInfo;
  })
  // query second page
  .then(({ endCursor, hasNextPage }) => {
    if (!hasNextPage) {
      throw Error('no next page');
    }
```

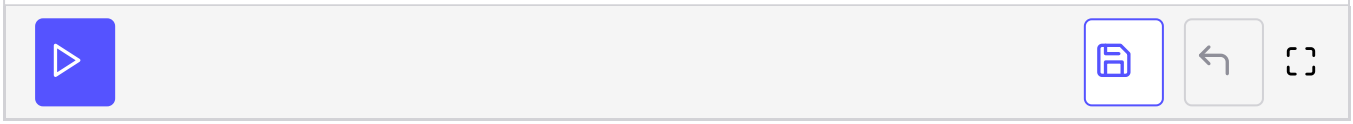
```
    return client.query({
      query: GET_REPOSITORIES_OF_ORGANIZATION,
      variables: {
        organization: 'the-road-to-learn-react',
        cursor: endCursor,
      },
    });
  })
  // resolve second page
  .then(result => {
    const { pageInfo, edges } = result.data.organization.repositories;
    const { endCursor, hasNextPage } = pageInfo;

    console.log('second page', edges.length);
    console.log('endCursor', endCursor);

    return pageInfo;
  })
  // log error when there is no next page
  .catch(console.log);

const ADD_STAR = gql`
  mutation AddStar($repositoryId: ID!) {
    addStar(input: { starrableId: $repositoryId }) {
      starrable {
        id
        viewerHasStarred
      }
    }
  }
`;


client
  .mutate({
    mutation: ADD_STAR,
    variables: {
      repositoryId: 'MDEwOlJlcG9zaXRvcnk2MzM1MjkwNw==',
    },
  })
  .then(console.log);
```



You've just learned how to use Apollo Client only without any view-layer library. This is to avoid confusing the features of Apollo Client and React Apollo.

Remember, Apollo Client can be used as a standalone GraphQL client without connecting it to a view-layer like React, though it may seem a bit dull to see the data only on the console. We'll see how Apollo connects the data-layer to a React view-layer in the next chapter.

Exercise

1. Confirm your source code for the last section.