React GraphQL Mutation

In this lesson, you will learn how to implement GraphQL Mutations in React.

The major use of GraphQL is to fetch data. However, there are always two sides to such an interface: read and write. That's where GraphQL mutations complement the interface. Previously, you learned about GraphQL mutations using GraphQL without React. In this section, you will implement such a mutation in your React GraphQL application.

`addStar` Mutation

We have executed GitHub's `addStar` mutation before in GraphQL. Now, let's implement this mutation in React. Before implementing the mutation, you should query additional information about the repository, which is partially required to star the repository in a mutation.

Environment Variables		∧
Key:	Value:	
REACT_APP_GITHUB...	Not Specified...	
GITHUB_PERSONAL...	Not Specified...	

```
const GET_ISSUES_OF_REPOSITORY = `
  query (
    $organization: String!,
    $repository: String!,
    $cursor: String
  ) {
    organization(login: $organization) {
      name
      url
      repository(name: $repository) {
        id
        name
```

```
        name
        url
        viewerHasStarred
        issues(first: 5, after: $cursor, states: [OPEN]) {
          ...
        }
      }
    }
  }
`;
```

The `viewerHasStarred` field returns a boolean that tells whether the viewer has starred the repository or not. This boolean helps determine whether to execute a `addStar` or `removeStar` mutation in the next steps. For now, you will only implement the `addStar` mutation. The `removeStar` mutation will be left off as part of the exercise. Also, the `id` field in the query returns the identifier for the repository, which you will need to clarify the target repository of your mutation.

The best place to trigger the mutation is a button that stars or un-stars the repository. That's where the `viewerHasStarred` boolean can be used for a conditional rendering to show either a "Star" or "Un-star" button. Since you are going to star a repository, the Repository component is the best place to trigger the mutation.

Environment Variables   ⌃

Key: Value:

REACT_APP_GITHUB...   Not Specified...

GITHUB_PERSONAL...   Not Specified...

```
const Repository = ({
  repository,
  onFetchMoreIssues,
  onStarRepository,
}) => (
  <div>
    ...
    <button
      type="button"
      onClick={() => onStarRepository()}
    >
      {repository.viewerHasStarred ? 'Unstar' : 'Star'}
    </button>
    <ul>
      ...
    </ul>
  </div>
);
```

To identify the repository that has to be starred, the mutation needs to know the `id` of the repository. Pass the `viewerHasStarred` property as a parameter to the handler, since you'll use the parameter to determine whether you want to execute the star or un-star mutation later.

Environment Variables ⌄

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

```
const Repository = ({ repository, onStarRepository }) => (
  <div>
    ...
    <button
      type="button"
      onClick={() =>
        onStarRepository(repository.id, repository.viewerHasStarred)
      }
    >
      {repository.viewerHasStarred ? 'Unstar' : 'Star'}
    </button>
    ...
  </div>
);
```

The handler should be defined in the App component. It passes through each component until it reaches the Repository component, also reaching through the Organization component on its way.

Environment Variables ⌄

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

```
const Organization = ({
  organization,
  errors,
  onFetchMoreIssues,
  onStarRepository,
}) => {
  ...
  return (
    <div>
      ...
```

```
      <Repository
        repository={organization.repository}
        onFetchMoreIssues={onFetchMoreIssues}

        onStarRepository={onStarRepository}
      />
    </div>
  );
};
```

Now it can be defined in the App component. Note that the `id` and the `viewerHasStarred` information can be de-structured from the App's local state, too. This is why you wouldn't need to pass this information in the handler, but use it from the local state instead. However, since the Repository component knew about the information already, it is fine to pass the information in the handler, which also makes the handler more explicit. It's also good preparation for dealing with multiple repositories and repository components later, since the handler will need to be more specific in these cases.

Environment Variables ⌃

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

```
class App extends Component {
  ...
  onStarRepository = (repositoryId, viewerHasStarred) => {
    ...
  };

  render() {
    const { path, organization, errors } = this.state;
    return (
      <div>
        ...
        {organization ? (
          <Organization
            organization={organization}
            errors={errors}
            onFetchMoreIssues={this.onFetchMoreIssues}
            onStarRepository={this.onStarRepository}
          />
        ) : (
          <p>No information yet ...</p>
        )}
      </div>
    );
  }
}
```

Now, we can implement the handler. The mutation can be outsourced from the component. Later, you can use the `viewerHasStarred` boolean in the

handler to perform a `addStar` or `removeStar` mutation. Executing the mutation looks similar to the GraphQL query from before. The API endpoint is not needed, because it was set in the beginning when you configured axios. The mutation can be sent in the `query` payload, which we'll cover later. The `variables` property is optional, but you need to pass the identifier.

Environment Variables ∧

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

```
const addStarToRepository = repositoryId => {
  return axiosGitHubGraphQL.post('', {
    query: ADD_STAR,
    variables: { repositoryId },
  });
};

class App extends Component {
  ...
  onStarRepository = (repositoryId, viewerHasStarred) => {
    addStarToRepository(repositoryId);
  };
  ...
}
```

Before you define the `addStar` mutation, check GitHub's GraphQL API again. There, you will find all information about the structure of the mutation, the required arguments, and the available fields for the result. For instance, you can include the `viewerHasStarred` field in the returned result to get an updated boolean of a starred or unstarred repository.

Environment Variables ∧

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

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

```
      }
    }

  }
`;
```

You could already execute the mutation in the browser by clicking the button. If you haven't starred the repository before, it should be starred after clicking the button. You can visit the repository on GitHub to get visual feedback, though you won't see any results reflected yet. The button still shows the "Star" label when the repository wasn't starred before, because the `viewerHasStarred` boolean wasn't updated in the local state of the App component after the mutation. That's the next thing you are going to implement. Since axios returns a promise, you can use the `then()` method on the promise to resolve it with your own implementation details.

Environment Variables ⌃

Key: Value:

REACT_APP_GITHUB...   Not Specified...

GITHUB_PERSONAL...   Not Specified...

```
const resolveAddStarMutation = mutationResult => state => {
  ...
};
class App extends Component {
  ...
  onStarRepository = (repositoryId, viewerHasStarred) => {
    addStarToRepository(repositoryId).then(mutationResult =>
      this.setState(resolveAddStarMutation(mutationResult)),
    );
  };
  ...
}
```

When resolving the promise from the mutation, you can find out about the `viewerHasStarred` property in the result. That's because you defined this property as a field in your mutation. It returns a new state object for React's local state, because you used the function in `this.setState()`. The spread operator syntax is used here, to update the deeply nested data structure. Only the `viewerHasStarred` property changes in the state object, because it's the only property returned by the resolved promise from the successful request. All other parts of the local state stay intact.

Kau: Value:

REACT_APP_GITHUB...   Not Specified...

GITHUB_PERSONAL...   Not Specified...

```
const resolveAddStarMutation = mutationResult => state => {
  const {
    viewerHasStarred,
  } = mutationResult.data.data.addStar.starrable;

  return {
    ...state,
    organization: {
      ...state.organization,
      repository: {
        ...state.organization.repository,
        viewerHasStarred,
      },
    },
  };
};
```

Now try to star the repository again. You may have to go on the GitHub page and unstar it first. The button label should adapt to the updated `viewerHasStarred` property from the local state to show a "Star" or "Unstar" label. You can use what you've learned about starring repositories to implement a `removeStar` mutation.

We also want to show the current number of people who have starred the repository, and update this count in the `addStar` and `removeStar` mutations. First, retrieve the total count of stargazers by adding the following fields to your query:

Key: Value:

REACT_APP_GITHUB...   Not Specified...

GITHUB_PERSONAL...   Not Specified...

```
const GET_ISSUES_OF_REPOSITORY = `
  query (
    $organization: String!,
    $repository: String!,
    $cursor: String
  ) {
    organization(login: $organization) {
      name
      url
```

```
        repository(name: $repository) {
          id
          name

          url
          stargazers {
            totalCount
          }
          viewerHasStarred
          issues(first: 5, after: $cursor, states: [OPEN]) {
            ...
          }
        }
      }
    }
  }
`;
```

Second, you can show the count as a part of your button label:

```
const Repository = ({
  repository,
  onFetchMoreIssues,
  onStarRepository,
}) => (
  <div>
    ...
    <button
      type="button"
      onClick={() =>
        onStarRepository(repository.id, repository.viewerHasStarred)
      }
    >
      {repository.stargazers.totalCount}
      {repository.viewerHasStarred ? ' Unstar' : ' Star'}
    </button>
    <ul>
      ...
    </ul>
  </div>
);
```

Now we want the count to update when we star (or un-star) a repository. It is
the same issue as the missing update for the `viewerHasStarred` property in the
local state of the component after the `addStar` mutation succeeded. Return to
your mutation resolver and update the total count of stargazers there as well.
While the stargazer object isn't returned as a result from the mutation, you
can increment and decrement the total count after a successful mutation

manually using a counter along with the `addStar` mutation.

```
const resolveAddStarMutation = mutationResult => state => {
  const {
    viewerHasStarred,
  } = mutationResult.data.data.addStar.starrable;
  const { totalCount } = state.organization.repository.stargazers;
  return {
    ...state,
    organization: {
      ...state.organization,
      repository: {
        ...state.organization.repository,
        viewerHasStarred,
        stargazers: {
          totalCount: totalCount + 1,
        },
      },
    },
  };
};
```

We have implemented our first mutation in React with GraphQL. So far, we have just implemented the `addStar` mutation. Even though the button already reflects the `viewerHasStarred` boolean by showing a "Star" or "Un-star" label, the button showing "Un-star" should still execute the `addStar` mutation. The `removeStar` mutation to un-star the repository is one of the practice exercises mentioned below.

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

const TITLE = 'React GraphQL GitHub Client';

const axiosGitHubGraphQL = axios.create({
  baseURL: 'https://api.github.com/graphql',
  headers: {
```

```
      Authorization:  bearer ${
        process.env.REACT_APP_GITHUB_PERSONAL_ACCESS_TOKEN
      }`,
  },
});

const GET_ISSUES_OF_REPOSITORY = `
  query ($organization: String!, $repository: String!, $cursor: String) {
    organization(login: $organization) {
      name
      url
      repository(name: $repository) {
        id
        name
        url
        stargazers {
          totalCount
        }
        viewerHasStarred
        issues(first: 5, after: $cursor, states: [OPEN]) {
          edges {
            node {
              id
              title
              url
              reactions(last: 3) {
                edges {
                  node {
                    id
                    content
                  }
                }
              }
            }
          }
          totalCount
          pageInfo {
            endCursor
            hasNextPage
          }
        }
      }
    }
  }
`;

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

const REMOVE_STAR = `
  mutation ($repositoryId: ID!) {
    removeStar(input:{starrableId:$repositoryId}) {
      starrable {
        viewerHasStarred
      }
```

```
    }
  }
`;

const getIssuesOfRepository = (path, cursor) => {
  const [organization, repository] = path.split('/');

  return axiosGitHubGraphQL.post('', {
    query: GET_ISSUES_OF_REPOSITORY,
    variables: { organization, repository, cursor },
  });
};

const resolveIssuesQuery = (queryResult, cursor) => state => {
  const { data, errors } = queryResult.data;

  if (!cursor) {
    return {
      organization: data.organization,
      errors,
    };
  }

  const { edges: oldIssues } = state.organization.repository.issues;
  const { edges: newIssues } = data.organization.repository.issues;
  const updatedIssues = [...oldIssues, ...newIssues];

  return {
    organization: {
      ...data.organization,
      repository: {
        ...data.organization.repository,
        issues: {
          ...data.organization.repository.issues,
          edges: updatedIssues,
        },
      },
    },
    errors,
  };
};

const addStarToRepository = repositoryId => {
  return axiosGitHubGraphQL.post('', {
    query: ADD_STAR,
    variables: { repositoryId },
  });
};

const resolveAddStarMutation = mutationResult => state => {
  const {
    viewerHasStarred,
  } = mutationResult.data.data.addStar.starrable;

  const { totalCount } = state.organization.repository.stargazers;

  return {
    ...state,
    organization: {
      ...state.organization,
      repository: {
        ...state.organization.repository,
```

```
          viewerHasStarred,
          stargazers: {
            totalCount: totalCount + 1,
          },
        },
      },
    };
};

const removeStarFromRepository = repositoryId => {
  return axiosGitHubGraphQL.post('', {
    query: REMOVE_STAR,
    variables: { repositoryId },
  });
};

const resolveRemoveStarMutation = mutationResult => state => {
  const {
    viewerHasStarred,
  } = mutationResult.data.data.removeStar.starrable;

  const { totalCount } = state.organization.repository.stargazers;

  return {
    ...state,
    organization: {
      ...state.organization,
      repository: {
        ...state.organization.repository,
        viewerHasStarred,
        stargazers: {
          totalCount: totalCount - 1,
        },
      },
    },
  };
};

class App extends Component {
  state = {
    path: 'the-road-to-learn-react/the-road-to-learn-react',
    organization: null,
    errors: null,
  };

  componentDidMount() {
    this.onFetchFromGitHub(this.state.path);
  }

  onChange = event => {
    this.setState({ path: event.target.value });
  };

  onSubmit = event => {
    this.onFetchFromGitHub(this.state.path);

    event.preventDefault();
  };

  onFetchFromGitHub = (path, cursor) => {
    getIssuesOfRepository(path, cursor).then(queryResult =>
      this.setState(resolveIssuesQuery(queryResult, cursor)),
```

```
    );
  };

  onFetchMoreIssues = () => {
    const {
      endCursor,
    } = this.state.organization.repository.issues.pageInfo;

    this.onFetchFromGitHub(this.state.path, endCursor);
  };

  onStarRepository = (repositoryId, viewerHasStarred) => {
    if (viewerHasStarred) {
      removeStarFromRepository(repositoryId).then(mutationResult =>
        this.setState(resolveRemoveStarMutation(mutationResult)),
      );
    } else {
      addStarToRepository(repositoryId).then(mutationResult =>
        this.setState(resolveAddStarMutation(mutationResult)),
      );
    }
  };

  render() {
    const { path, organization, errors } = this.state;

    return (
      <div>
        <h1>{TITLE}</h1>

        <form onSubmit={this.onSubmit}>
          <label htmlFor="url">
            Show open issues for https://github.com/
          </label>
          <input
            id="url"
            type="text"
            value={path}
            onChange={this.onChange}
            style={{ width: '300px' }}
          />
          <button type="submit">Search</button>
        </form>

        <hr />

        {organization ? (
          <Organization
            organization={organization}
            errors={errors}
            onFetchMoreIssues={this.onFetchMoreIssues}
            onStarRepository={this.onStarRepository}
          />
        ) : (
          <p>No information yet ...</p>
        )}
      </div>
    );
  }
}

const Organization = ({
```

```
  organization,
  errors,
  onFetchMoreIssues,
  onStarRepository,
}) => {
  if (errors) {
    return (
      <p>
        <strong>Something went wrong:</strong>
        {errors.map(error => error.message).join(' ')}
      </p>
    );
  }

  return (
    <div>
      <p>
        <strong>Issues from Organization:</strong>
        <a href={organization.url}>{organization.name}</a>
      </p>
      <Repository
        repository={organization.repository}
        onFetchMoreIssues={onFetchMoreIssues}
        onStarRepository={onStarRepository}
      />
    </div>
  );
};

const Repository = ({
  repository,
  onFetchMoreIssues,
  onStarRepository,
}) => (
  <div>
    <p>
      <strong>In Repository:</strong>
      <a href={repository.url}>{repository.name}</a>
    </p>

    <button
      type="button"
      onClick={() =>
        onStarRepository(repository.id, repository.viewerHasStarred)
      }
    >
      {repository.stargazers.totalCount}
      {repository.viewerHasStarred ? ' Unstar' : ' Star'}
    </button>

    <ul>
      {repository.issues.edges.map(issue => (
        <li key={issue.node.id}>
          <a href={issue.node.url}>{issue.node.title}</a>

          <ul>
            {issue.node.reactions.edges.map(reaction => (
              <li key={reaction.node.id}>{reaction.node.content}</li>
            ))}
          </ul>
        </li>
      ))}
```

```
      </ul>

      <hr />

      {repository.issues.pageInfo.hasNextPage && (
        <button onClick={onFetchMoreIssues}>More</button>
      )}
    </div>
);

export default App;
```

In the next lesson, we will learn the shortcomings of using GraphQL in React without a GraphQL Client Library.