GraphQL Nested Objects in React

Learn to request a nested object for the Organization

In this lesson, we will request a nested object for the organization. Since the application will eventually show the issues in a repository, we will fetch a repository of an organization as the next step.

> Remember, a query reaches into the GraphQL graph. So we can nest the `repository` field in the `organization` when the schema defines the relationship between these two entities.

Environment Variables

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

```
const GET_REPOSITORY_OF_ORGANIZATION = `
  {
    organization(login: "the-road-to-learn-react") {
      name
      url
      repository(name: "the-road-to-learn-react") {
        name
        url
      }
    }
  }
`;

class App extends Component {
  ...
  onFetchFromGitHub = () => {
    axiosGitHubGraphQL
      .post('', { query: GET_REPOSITORY_OF_ORGANIZATION })
      .then(result =>
          ...
      );
  };
  ...
}
```

In this case, the repository name is identical to the organization. That's okay for now. Later on, we will define an organization and repository on our own dynamically.

You can extend the `Organization` component with another `Repository` component as a child component. The result for the query should now have a nested repository object in the organization object.

Environment Variables

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

```
const Organization = ({ organization, errors }) => {
  if (errors) {
    ...
  }

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

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

The GraphQL query structure aligns perfectly with the component tree of the application. We can continue extending the query structure by nesting other objects into the query and extending the component tree along the structure of the GraphQL query. Since the application is an issue tracker, we need to add a "list" of issues to the query.

To be more thorough while making query structure, you can open the "Docs"

sidebar in GraphQL and learn about the types of `Organizations`, `Repositories` and `Issues`. The paginated issues list field can be found there as well.

In the code below, we extend the query with the list field for the issues. These issues are a paginated list. To fetch the last item of the list, nest it in the `repository` field with a `"last"` argument.

Environment Variables

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

```
const GET_ISSUES_OF_REPOSITORY = `
  {
    organization(login: "the-road-to-learn-react") {
      name
      url
      repository(name: "the-road-to-learn-react") {
        name
        url
        issues(last: 5) {
          edges {
            node {
              id
              title
              url
            }
          }
        }
      }
    }
  }
`;
```

You can request id for each issue using the `id` field on the issue's `node` field. Some of the best practices in React can be viewed here.

Remember to adjust the name of the query variable when its used to perform the request.

Environment Variables

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

```
class App extends Component {
  ...
```

```
  onFetchFromGitHub = () => {
    axiosGitHubGraphQL
      .post('', { query: GET_ISSUES_OF_REPOSITORY })

      .then(result =>
          ...
      );
  };
  ...
}
```

The component structure follows the query structure. We will add a list of rendered issues to the Repository component as per needed. It is up to you to extract it to its own component as a refactoring to keep your components concise, readable, and maintainable.

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

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

That's it for the nested objects, fields, and list fields in a query. Once you run your application again, you should see the last issues of the specified repository rendered.

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',
  headers: {
    Authorization: `bearer ${
```

```
      process.env.REACT_APP_GITHUB_PERSONAL_ACCESS_TOKEN
    }`,
  },

});

const TITLE = 'React GraphQL GitHub Client';
const GET_ISSUES_OF_REPOSITORY = `
  {
    organization(login: "the-road-to-learn-react") {
      name
      url
                        repository(name: "the-road-to-learn-react") {
        name
        url
                                issues(last: 5) {
          edges {
            node {
              id
              title
              url
            }
          }
        }
      }
    }
  }
`;
class App extends Component {

   state = {
    path: 'the-road-to-learn-react/the-road-to-learn-react',
  };

   componentDidMount() {
    // fetch data
      this.onFetchFromGitHub();
  }

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

  onSubmit = event => {
    // fetch data

    event.preventDefault();
  };

        onFetchFromGitHub = () => {
    axiosGitHubGraphQL
       .post('', { query: GET_ISSUES_OF_REPOSITORY })
      .then(result =>
            this.setState(() => ({
          organization: result.data.data.organization,
          errors: result.data.errors,
        })),
        );
  };
  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} />
        ) : (
          <p>No information yet ...</p>
        )}
      </div>
    );
  }
}

const Organization = ({ organization, errors }) => {
  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} />
  </div>
);
};

const Repository = ({ repository }) => (
  <div>
    <p>
      <strong>In Repository:</strong>
      <a href={repository.url}>{repository.name}</a>
    </p>
     <ul>
      {repository.issues.edges.map(issue => (
        <li key={issue.node.id}>
          <a href={issue.node.url}>{issue.node.title}</a>
        </li>
      ))}
    </ul>
```

```
  </div>
);


export default App;
```

In the next lesson, we will learn about GraphQL Variables and Arguments.