GraphQL Variables and Arguments in React

Let's learn how to query using GraphQL variables and arguments!

WE'LL COVER THE FOLLOWING

- Exercises
- Reading Task

Next, we'll make use of the form and input elements. They should be used to request the data from GitHub's GraphQL API when a user fills in content and submits it. The content is also used for the initial request in `componentDidMount()` of the `App` component. So far, the organization `login` and repository `name` were inlined arguments in the query. Now, you should be able to pass in the `path` from the local state to the query to define dynamically an organization and repository. That's where variables in a GraphQL query came into play, do you remember?

First, let's use a naive approach by performing string interpolation with JavaScript rather than using GraphQL variables. To do this, we will refactor the query from a template literal variable to a function that returns a template literal variable. By using the function, you should be able to pass in an organization and repository.

Environment Variables

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

```
const getIssuesOfRepositoryQuery = (organization, repository) => `
  {
    organization(login: "${organization}") {
      name
      url
      repository(name: "${repository}") {
        name
        url
        issues(last: 5) {
          edges {
            node {
              id
              title
              url
            }
          }
        }
      }
    }
  }
`;
```

Next, we'll call the `onFetchFromGitHub()` class method in the submit handler, but also when the component mounts in `componentDidMount()` with the initial local state of the `path` property. These are the two essential places to fetch the data from the GraphQL API on initial render, and on every other manual submission from a button click.

Environment Variables

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

```
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 = () => {
    ...
  }

  render() {
    ...
  }
}
```

Lastly, we'll call the function that returns the query instead of passing the query string directly as payload. Use the JavaScript's split method on a string to retrieve the prefix and suffix of the `/` character from the path variable where the prefix is the organization and the suffix is the repository.

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

```
class App extends Component {
  ...
  onFetchFromGitHub = path => {
    const [organization, repository] = path.split('/');

    axiosGitHubGraphQL
      .post('', {
        query: getIssuesOfRepositoryQuery(organization, repository),
      })
      .then(result =>
        this.setState(() => ({
          organization: result.data.data.organization,
          errors: result.data.errors,
        })),
      );
  };
  ...
}
```

Since the split returns an array of values and it is assumed that there is only one slash in the path, the array should consist of two values:

- the organization
- the repository

That's why it is convenient to use a JavaScript array destructuring to pull out both values from an array in the same line.

Note that the application is not built for to be robust, but is intended only as a learning experience. It is unlikely anyone will ask a user to input the organization and repository with a different pattern than organization/repository, so there is no validation included yet. Still, it is a good foundation for you to gain experience with the concepts.

If you want to go further, you can extract the first part of the class method to its own function, which uses axios to send a request with the query and return a promise. The promise can be used to resolve the result into the local state, using `this.setState()` in the `then()` resolver block of the promise.

Environment Variables

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

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

  return axiosGitHubGraphQL.post('', {
    query: getIssuesOfRepositoryQuery(organization, repository),
  });
};

class App extends Component {
  ...
  onFetchFromGitHub = path => {
    getIssuesOfRepository(path).then(result =>
      this.setState(() => ({
        organization: result.data.data.organization,
        errors: result.data.errors,
      })),
    );
  };
  ...
}
```

You can always split your applications into parts, whether they are functions or components, to make them concise, readable, reusable and testable. The function that is passed to `this.setState()` can be extracted as higher-order function. It has to be a higher-order function, because you need to pass the result of the promise, but also provide a function for the `this.setState()` method.

Environment Variables

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

```
const resolveIssuesQuery = queryResult => () => ({
  organization: queryResult.data.data.organization,

  errors: queryResult.data.errors,
});

class App extends Component {
  ...
  onFetchFromGitHub = path => {
    getIssuesOfRepository(path).then(queryResult =>
      this.setState(resolveIssuesQuery(queryResult)),
    );
  };
  ...
}
```

Now we've made our query flexible by providing dynamic arguments to your query. Try it by starting your application on the command line and by filling in a different organization with a specific repository (e.g. facebook/create-react-app).

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 resolveIssuesQuery = queryResult => () => ({
  organization: queryResult.data.data.organization,
  errors: queryResult.data.errors,
});


const TITLE = 'React GraphQL GitHub Client';
const getIssuesOfRepository = path => {
  const [organization, repository] = path.split('/');

  return axiosGitHubGraphQL.post('', {
    query: getIssuesOfRepositoryQuery(organization, repository),
  });
};

const getIssuesOfRepositoryQuery = (organization, repository) => `
const getIssuesOfRepositoryQuery = (organization, repository) =>
  {
    organization(login: "${organization}") {
      name
      url
      repository(name: "${repository}") {
        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(this.state.path);
  }

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

  onSubmit = event => {
    // fetch data
         this.onFetchFromGitHub(this.state.path);
    event.preventDefault();
  };

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

It’s a decent setup, but there was nothing to see about variables yet. You simply passed the arguments to the query using a function and string interpolation with template literals. Now we’ll use GraphQL variables instead, to refactor the query variable again to a template literal that defines inline variables.

Environment Variables

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

```
const GET_ISSUES_OF_REPOSITORY = `
  query ($organization: String!, $repository: String!) {
    organization(login: $organization) {
      name
      url
      repository(name: $repository) {
        name
        url
        issues(last: 5) {
          edges {
            node {
              id
              title
              url
            }
          }
        }
      }
    }
  }
`;
```

Now you can pass those variables as arguments next to the query for the HTTP POST request:

Environment Variables

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

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

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

Finally, the query takes variables into account without detouring into a function with string interpolation. I strongly suggest practicing with the exercises below before continuing to the next lesson. We've yet to discuss

features like fragments or operation names, but we'll soon cover them using Apollo instead of plain HTTP with axios.

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 resolveIssuesQuery = queryResult => () => ({
  organization: queryResult.data.data.organization,
  errors: queryResult.data.errors,
});


const TITLE = 'React GraphQL GitHub Client';
const getIssuesOfRepository = path => {
  const [organization, repository] = path.split('/');

  return axiosGitHubGraphQL.post('', {
    query: GET_ISSUES_OF_REPOSITORY,
    variables: { organization, repository },
  });
};
const GET_ISSUES_OF_REPOSITORY = `
  query ($organization: String!, $repository: String!) {
    organization(login: $organization) {
      name
      url
      repository(name: $repository) {
        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(this.state.path);
  }

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

  onSubmit = event => {
    // fetch data
         this.onFetchFromGitHub(this.state.path);
    event.preventDefault();
  };

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

Exercises

1. Confirm your source code for the last section
2. Explore and add fields to your organization, repository, and issues
   - Extend your components to display the additional information

Reading Task

1. Read more about serving a GraphQL API over HTTP