Apollo Client Prefetching in React

In this lesson, you will be introduced to another UI technique which we will implement with Apollo Client in React.

WE'LL COVER THE FOLLOWING

This lesson is all about prefetching data. It is another UX technique that can be deployed to the optimistic UI technique you used earlier. We will implement the prefetching data feature for the list of issues, but feel free to implement it for other data fetching later as your exercise.

Data Prefetching

When your application renders for the first time, there are no issues fetched, so no issues are rendered. The user has to toggle the filter button to fetch open issues and do it again to fetch closed issues. The third click will hide the list of issues again. The goal of this lesson is to prefetch the next bulk of issues when the user hovers the filter button. For instance, when the issues are still hidden and the user hovers the filter button, the issues with the open state are prefetched in the background. When the user clicks the button, there is no waiting time, because the issues with the open state are already there. The same scenario applies to the transition from open to closed issues.

To prepare this behavior, we'll split out the filter button as its own component in the *src/Issue/IssueList/index.js* file:

Environment Variables

Key: Value:

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

```
const Issues = ({
  repositoryOwner,
  repositoryName,
  issueState,
  onChangeIssueState,
}) => (
  <div className="Issues">
    <IssueFilter
      issueState={issueState}
      onChangeIssueState={onChangeIssueState}
    />

    {isShow(issueState) && (
      ...
    )}
  </div>
);

const IssueFilter = ({ issueState, onChangeIssueState }) => (
  <ButtonUnobtrusive
    onClick={() => onChangeIssueState(TRANSITION_STATE[issueState])}
  >
    {TRANSITION_LABELS[issueState]}
  </ButtonUnobtrusive>
);
```

src/Issue/IssueList/index.js

Now it is easier to focus on the `IssueFilter` component where most of the logic for data prefetching is implemented. Like before, the prefetching should happen when the user hovers over the button. There needs to be a prop for it, and a callback function which is executed when the user hovers over it. There is such a prop (attribute) for a button (element). We are dealing with HTML elements here.

Environment Variables

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

```
const prefetchIssues = () => {};

...

const IssueFilter = ({ issueState, onChangeIssueState }) => (
  <ButtonUnobtrusive
    onClick={() => onChangeIssueState(TRANSITION_STATE[issueState])}
    onMouseOver={prefetchIssues}
```

```
    >
      {TRANSITION_LABELS[issueState]}


    </ButtonUnobtrusive>
);
```

src/Issue/IssueList/index.js

The `prefetchIssue()` function has to execute the identical GraphQL query executed by the `Query` component in the `Issues` component, but this time it is done in an imperative way instead of declarative. Rather than using the `Query` component for it, we use the Apollo Client instance directly to execute a query. Remember, the Apollo Client instance is hidden in the component tree because you used React's Context API to provide the Apollo Client instance the component tree's top level.

The `Query` and `Mutation` components have access to the Apollo Client, even though we have never used it yourself directly. However, this time we use it to query the prefetched data. Hence, we use the `ApolloConsumer` component from the React Apollo package to expose the Apollo Client instance in our component tree. We have used the `ApolloProvider` somewhere to provide the client instance, and we can use the `ApolloConsumer` to retrieve it now. In the *src/Issue/IssueList/index.js* file, we import the `ApolloConsumer` component and use it in the `IssueFilter` component. It gives us access to the Apollo Client instance via its render props child function.

Environment Variables

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

```
import React from 'react';
import { Query, ApolloConsumer } from 'react-apollo';
import gql from 'graphql-tag';
import { withState } from 'recompose';

...

const IssueFilter = ({ issueState, onChangeIssueState }) => (
  <ApolloConsumer>
    {client => (
      <ButtonUnobtrusive
        onClick={() =>
          onChangeIssueState(TRANSITION_STATE[issueState])
        }
```

```
        onMouseOver={() => prefetchIssues(client)}
      >
        {TRANSITION_LABELS[issueState]}

      </ButtonUnobtrusive>
    )}
  </ApolloConsumer>
);
```

src/Issue/IssueList/index.js

Now we have access to the Apollo Client instance to perform queries and mutations, which will enable you to query GitHub's GraphQL API imperatively. The variables needed to perform the prefetching of issues are the same ones used in the `Query` component. We need to pass those to the `IssueFilter` component, and then to the `prefetchIssues()` function.

Environment Variables

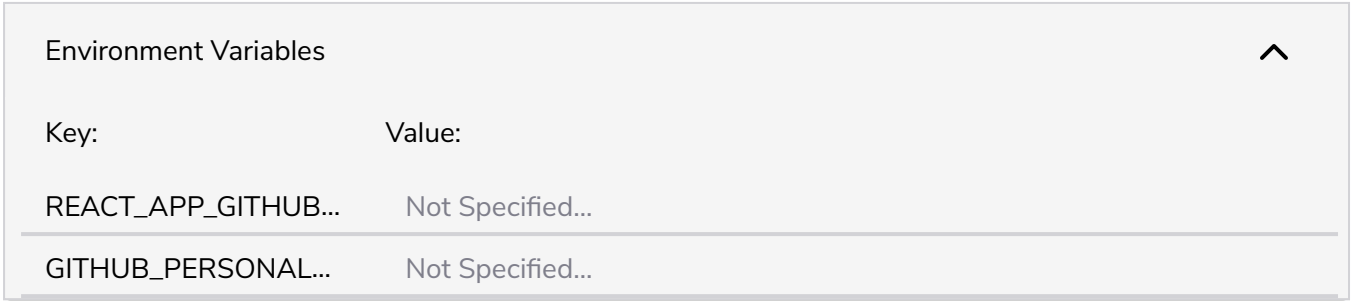
Key:	Value:
REACT_APP_GITHUB...	Not Specified...
GITHUB_PERSONAL...	Not Specified...

```
...

const Issues = ({
  repositoryOwner,
  repositoryName,
  issueState,
  onChangeIssueState,
}) => (
  <div className="Issues">
    <IssueFilter
      repositoryOwner={repositoryOwner}
      repositoryName={repositoryName}
      issueState={issueState}
      onChangeIssueState={onChangeIssueState}
    />

    {isShow(issueState) && (
      ...
    )}
  </div>
);

const IssueFilter = ({
  repositoryOwner,
  repositoryName,
  issueState,
  onChangeIssueState,
}) => (
  <ApolloConsumer>
    {client => (
      <ButtonUnobtrusive
        onClick={() =>
```

```
          onChangeIssueState(TRANSITION_STATE[issueState])
        }
        onMouseOver={() =>
          prefetchIssues(
            client,
            repositoryOwner,
            repositoryName,
            issueState,
          )
        }
      >
        {TRANSITION_LABELS[issueState]}
      </ButtonUnobtrusive>
    )}
  </ApolloConsumer>
);

...
```

src/Issue/IssueList/index.js

We use this information to perform the prefetching data query. The Apollo Client instance exposes a `query()` method for this. We make sure to retrieve the next `issueState` because when prefetching open issues, the current `issueState` should be `NONE`.



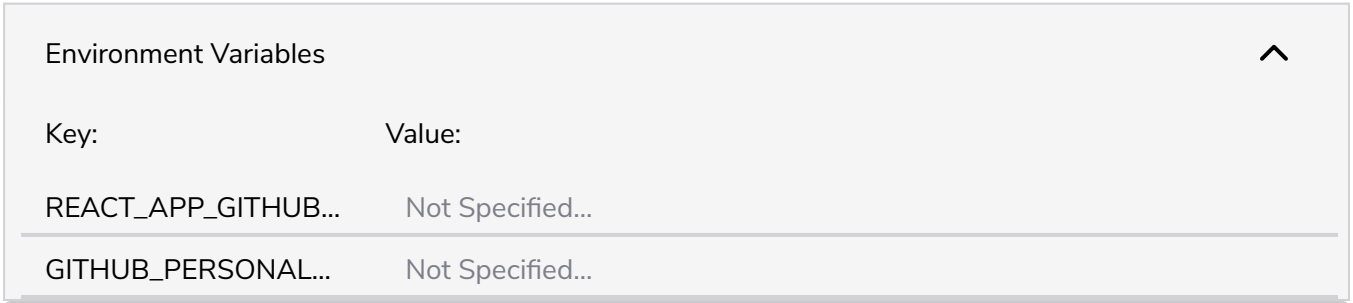

```
const prefetchIssues = (
  client,
  repositoryOwner,
  repositoryName,
  issueState,
) => {
  const nextIssueState = TRANSITION_STATE[issueState];

  if (isShow(nextIssueState)) {
    client.query({
      query: GET_ISSUES_OF_REPOSITORY,
      variables: {
        repositoryOwner,
        repositoryName,
        issueState: nextIssueState,
      },
    });
  }
};
```

src/Issue/IssueList/index.js

That's it. Once the button is hovered, it should prefetch the issues for the next `issueState`. The Apollo Client makes sure that the new data is updated in the cache as it would do for the `Query` component. There shouldn't be any visible loading indicator in between except when the network request takes too long and you click the button right after hovering it. You can verify that the request is happening in your network tab in the developer development tools of your browser.

In the end, you have learned about two UX improvements that can be achieved with ease when using Apollo Client: optimistic UI and prefetching data.

Check out everything below:




```
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;
```

Exercise

1. Confirm your source code for the last section

Reading Task

1. Read more about Apollo Prefetching and Query Splitting in React