Implementing the Issues Feature: Client-Side Filter

Here we learn how to enhance the Issue feature with client-side filtering.

WE'LL COVER THE FOLLOWING

- Exercises

First, let's introduce our three states as enumeration next to the Issues component. The `NONE` state is used to show no issues; otherwise, the other states are used to show open or closed issues.

Environment Variables

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

```
const ISSUE_STATES = {
  NONE: 'NONE',
  OPEN: 'OPEN',
  CLOSED: 'CLOSED',
};
```

src/Issue/IssueList/index.js

Second, let's implement a short function that decides whether it is a state to show the issues or not. This function can be defined in the same file.

Environment Variables

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

```
const isShow = issueState => issueState !== ISSUE_STATES.NONE;
```

src/Issue/IssueList/index.js

Third, the function can be used for conditional rendering, to either query the issues and show the `IssueList`, or to do nothing. It's not clear yet where the `issueState` property comes from.

Environment Variables

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

```
const Issues = ({ repositoryOwner, repositoryName }) => (
  <div className="Issues">
    {isShow(issueState) && (
      <Query ... >
        ...
      </Query>
    )}
  </div>
);
```

src/Issue/IssueList/index.js

The `issueState` property must come from the local state to toggle it via a button in the component, so the `Issues` component must be refactored to a class component to manage this state.

Environment Variables

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

```
class Issues extends React.Component {
  state = {
    issueState: ISSUE_STATES.NONE,
  };

  render() {
    const { issueState } = this.state;
    const { repositoryOwner, repositoryName } = this.props;

    return (
      <div className="Issues">
        {isShow(issueState) && (
          <Query ... >
            ...
```

```
        )}
      </div>
    );
  }
}
```

src/Issue/IssueList/index.js

The application should be error-free now, because the initial state is set to `NONE` and the conditional rendering prevents the query and the rendering of a result. However, the client-side filtering is not done yet, as we still need to toggle the `issueState` property with React's local state. The `ButtonUnobtrusive` component has the appropriate style, so we can reuse it to implement this toggling behavior to transition between the three available states.

Environment Variables

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

```
...

import IssueItem from '../IssueItem';
import Loading from '../../Loading';
import ErrorMessage from '../../Error';
import { ButtonUnobtrusive } from '../../Button';

class Issues extends React.Component {
  state = {
    issueState: ISSUE_STATES.NONE,
  };

  onChangeIssueState = nextIssueState => {
    this.setState({ issueState: nextIssueState });
  };

  render() {
    const { issueState } = this.state;
    const { repositoryOwner, repositoryName } = this.props;

    return (
      <div className="Issues">
        <ButtonUnobtrusive
          onClick={() =>
            this.onChangeIssueState(TRANSITION_STATE[issueState])
          }
        >
          {TRANSITION_LABELS[issueState]}
        </ButtonUnobtrusive>

        {isShow(issueState) && (
          <Query ... >
```

```
            ...
          </Query>
        )}

      </div>
    );
  }
}
```

src/Issue/IssueList/index.js

In the last step, we introduced the button to toggle between the three states. We used two enumerations, `TRANSITION_LABELS` and `TRANSITION_STATE`, to show an appropriate button label and to define the next state after a state transition. These enumerations can be defined next to the `ISSUE_STATES` enumeration.

Environment Variables

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

```
const TRANSITION_LABELS = {
  [ISSUE_STATES.NONE]: 'Show Open Issues',
  [ISSUE_STATES.OPEN]: 'Show Closed Issues',
  [ISSUE_STATES.CLOSED]: 'Hide Issues',
};

const TRANSITION_STATE = {
  [ISSUE_STATES.NONE]: ISSUE_STATES.OPEN,
  [ISSUE_STATES.OPEN]: ISSUE_STATES.CLOSED,
  [ISSUE_STATES.CLOSED]: ISSUE_STATES.NONE,
};
```

src/Issue/IssueList/index.js

As you can see, the former enumeration only matches a label to a given state while the latter enumeration matches the next state to a given state. That's how toggling to a next state can be made simple. Last but not least, the `issueState` from the local state has to be used to filter the list of issues after they have been queried and should be rendered.

Environment Variables

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

GITHUB_PERSONAL... Not Specified...

```
class Issues extends React.Component {
  ...

  render() {
    ...

    return (
      <div className="Issues">
        ...

        {isShow(issueState) && (
          <Query ... >
            {({ data, loading, error }) => {
              if (error) {
                return <ErrorMessage error={error} />;
              }

              const { repository } = data;

              if (loading && !repository) {
                return <Loading />;
              }

              const filteredRepository = {
                issues: {
                  edges: repository.issues.edges.filter(
                    issue => issue.node.state === issueState,
                  ),
                },
              };

              if (!filteredRepository.issues.edges.length) {
                return <div className="IssueList">No issues ...</div>;
              }

              return (
                <IssueList issues={filteredRepository.issues} />
              );
            }}
          </Query>
        )}
      </div>
    );
  }
}
```

src/Issue/IssueList/index.js

We have implemented client-side filtering! The button is used to toggle between the three states managed in the local state of the component. Run the application below:

Environment Variables

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

```
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;
```

The issues are only queried in filtered and rendered states. In the next step, the existing client-side filtering should be advanced to server-side filtering, which means the filtered issues are already requested from the server and not filtered afterward on the client.

Exercises

1. Confirm your source code for the last section

2. Install the recompose library which implements many higher-order components
3. Refactor the `Issues` component from class component to functional stateless component
4. Use the `withState` HOC for the `Issues` component to manage the `issueState`