Error Handling

Let's catch some errors in your application and debug them!

Now we've taken care of interactions with the Hacker News API. We've introduced an elegant way to cache results from the API and make use of its paginated list functionality to fetch an endless list of sublists of stories from the API. But no application is complete without error handling.

In this chapter, we introduce an efficient solution to add error handling for the application that we are building in case of an erroneous API request. We have learned the necessary building block to introduce error handling in React: local state and conditional rendering. The error is just another state, which we store in the local state and display with conditional rendering in the component.

Now, we'll implement it in the App component, since that's where we fetch data from the Hacker News API. First, introduce the error in the local state. It is initialized as null but will be set to the error object in case of an error.

```
class App extends Component {
  constructor(props) {
    super(props);

    this.state = {
      results: null,
      searchKey: '',
      searchTerm: DEFAULT_QUERY,
      error: null,
    };

    ...
  }

...

}
```

Second, use the catch block in your native fetch to store the error object in the local state by using `setState()`. Every time the API request isn't successful, the

catch block is executed.

```
class App extends Component {

  ...

  fetchSearchTopStories(searchTerm, page = 0) {
    // https://github.com/the-road-to-learn-react/the-road-to-learn-react/issues/43
    fetch(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}${page}&${PAR
      .then(response => response.json())
      .then(result => this.setSearchTopStories(result))
      .catch(error => this.setState({ error }));
  }

  ...

}
```

Third, retrieve the error object from your local state in the `render()` method, and display a message in case of an error using React's conditional rendering.

```
class App extends Component {

  ...

  render() {
    const {
      searchTerm,
      results,
      searchKey,
      error
    } = this.state;

    ...

    if (error) {
      return <p>Something went wrong.</p>;
    }

    return (
      <div className="page">
        ...
      </div>
    );
  }
}
```

If you want to test that your error handling is working, change the API URL to something non-existent to force an error.

```
const PATH_BASE = 'https://hn.mydomain.com/api/v1';
```

Now you should see an error message instead of your application. It's up to you where you want to place the conditional rendering for the error message. In this case, the app is overridden by the error message, but that wouldn't be the best user experience. The remaining application would still be visible so the user knows it's still running:

```
class App extends Component {

  ...

  render() {
    const {
      searchTerm,
      results,
      searchKey,
      error
    } = this.state;

    const page = (
      results &&
      results[searchKey] &&
      results[searchKey].page
    ) || 0;

    const list = (
      results &&
      results[searchKey] &&
      results[searchKey].hits
    ) || [];

    return (
      <div className="page">
        <div className="interactions">
          ...
        </div>
        { error
          ? <div className="interactions">
            <p>Something went wrong.</p>
          </div>
          : <Table
            list={list}
            onDismiss={this.onDismiss}
          />
        }
        ...
      </div>
    );
  }
}
```

Remember to revert the URL for the API to the existent one:

```
const PATH_BASE = 'https://hn.algolia.com/api/v1';
```

Our application now has error handling in case the API request fails.

Further Reading:

- Read about React's Error Handling for Components