Component Interface with PropTypes

In this lesson, we introduce a way to make React components type-safe using PropTypes.

TypeScript and Flow are used to introduce a type system to JavaScript. A typed language is less error prone because the code gets validated based on its program text. Editors and other utilities can catch these errors before the program runs.

In the course, you will not introduce Flow or TypeScript, but another useful way to check your types in components: React comes with a built-in type checker to prevent bugs. You can use 'PropTypes' to describe your component interface. All the props passed from a parent component to a child get validated based on the PropTypes interface assigned to the child.

This section will show you to make components type safe with PropTypes. I will omit the changes for the following chapters, since they add unnecessary code refactorings, but you should update them along the way to keep your components interface type safe.

First, install a separate package for React:

```
npm install prop-types
```

Now, you can import the PropTypes.

```
import React, { Component } from 'react';
import axios from 'axios';
import PropTypes from 'prop-types';
```

Let's start to assign a props interface to the components:

```
const Button = ({
  onClick,
  className = '',
  children,
}) =>
  <button
    onClick={onClick}
    className={className}
    type="button"
  >
    {children}
  </button>

Button.propTypes = {
  onClick: PropTypes.func,
  className: PropTypes.string,
  children: PropTypes.node,
};
```

Essentially, we want to take every argument from the function signature and assign a PropType to it. The basic PropTypes for primitives and complex objects are:

- PropTypes.array
- PropTypes.bool
- PropTypes.func
- PropTypes.number
- PropTypes.object
- PropTypes.string

There are two more PropTypes that can define a renderable fragment (node), e.g. a string, and a React element:

- PropTypes.node
- PropTypes.element

You already used the `node` PropType for the Button component. There are more PropType definitions in the official React documentation.

At the moment, all the defined PropTypes for the Button are optional. The parameters can be `null` or `undefined`. But for several props you want to enforce that they be defined. To do this, we make it a requirement that these props are passed to the component.

```
Button.propTypes = {
  onClick: PropTypes.func.isRequired,
  className: PropTypes.string,
  children: PropTypes.node.isRequired,
};
```

The `className` is not required, because it can default to an empty string. Next you will define a PropType interface for the Table component:

```
Table.propTypes = {
  list: PropTypes.array.isRequired,
  onDismiss: PropTypes.func.isRequired,
};
```

You can define the content of an array PropType more explicitly:

```
Table.propTypes = {
  list: PropTypes.arrayOf(
    PropTypes.shape({
      objectID: PropTypes.string.isRequired,
      author: PropTypes.string,
      url: PropTypes.string,
      num_comments: PropTypes.number,
      points: PropTypes.number,
    })
  ).isRequired,
  onDismiss: PropTypes.func.isRequired,
};
```

Only the `objectID` is required, because some of the code depends on it. The other properties are only displayed, so they are not required. Moreover you cannot be sure the Hacker News API always has a defined property for each object in the array.

You can also define default props in your component. The `className` property has an ES6 default parameter in the component signature:

```
const Button = ({
  onClick,
  className = '',
  children
}) =>
  ...
```

Replace it with the internal React default prop:

```
const Button = ({
  onClick,
  className,
  children
}) =>
  <button
    onClick={onClick}

    className={className}
    type="button"
  >
    {children}
  </button>

Button.defaultProps = {
  className: '',
};
```

Like the ES6 default parameter, the default prop ensures the property is set to a default value when the parent component doesn't specify it. The PropType type check happens after the default prop is evaluated.

If you run your tests again, you might see PropType errors for your components on your command line. It happens when we don't define all props for components in the tests that are required in the PropType definition. The tests themselves all pass correctly though. Make sure to pass all required props to the components in your tests to avoid these errors.

Exercises

- Define the PropType interface for the Search component
- Add and update the PropType interfaces when you add and update components in the next chapters

Further Reading:

- Read about React PropTypes

Q

Which of the following is an example of complex object PropType?

COMPLETED 0%

1 of 1