Garrett Hopper

Dynamically Generating React-Bootstrap Components

Garrett Hopper

Jun 26, 2020 • 5 Min read • 449 Views

Jun 26, 2020 • 5 Min read • 449 Views

Web Development Front End Web Dev... Client-side Framew... React

Introduction

4

Introduction

Imagine you're working on creating a new React-Bootstrap page and realize you need to use a component that's slightly different from an existing component. There are a few different ways this could be done.

Depending on the context, the existing component could be parameterized to support the new functionality. This works in some

scenarios, but it can often lead to components with many mismatched features that are difficult to maintain. Another option is to simply copy and paste the existing component. If the shared functionality is only going to be duplicated in two components, this might make sense. This isn't a great solution if the functionality is going to be duplicated in three or more components though.

This is where dynamically generated components come in. Since React allows you to use simple functions as components and JavaScript supports first-class functions, a function can be created which dynamically generates new components.

In this example, the React-Bootstrap `<Toast>` element will be used.

jsx

```jsx
import React from 'react';
import Toast from 'react-bootstrap/Toast';
import Container from 'react-bootstrap/Container';

function ErrorToast() {
  return (
    <Toast>
      <Toast.Header>Error</Toast.Header>
      <Toast.Body>Something went wrong.</Toast.Body>
    </Toast>
  );
}

export default function App() {
  return <Container>
```

```
      <ErrorToast />
    </Container>
  }
```

This new `<ErrorToast>` element will create a new React-Bootstrap `<Toast>` element with a preset header and body. What if you wanted to create a `<WarningToast>` or a `<NotifyToast>`? Ideally you wouldn't need to duplicate the entire component just to modify the preset text.

Creating a Dynamic Component Generator

The first step is to create a function that returns function components.

jsx

```jsx
import React from 'react';
import Toast from 'react-bootstrap/Toast';

function createToast() {
  return () => (
    <Toast>
      <Toast.Header>Error</Toast.Header>
      <Toast.Body>Something went wrong.</Toast.Body>
    </Toast>
  );
}
```

This function will allow you to create an `ErrorToast` component.

jsx

```jsx
const ErrorToast = createToast();
```

Obviously this doesn't provide much benefit until the `createToast` function is updated to accept some arguments.

jsx

```jsx
import React from 'react';
import Toast from 'react-bootstrap/Toast';

function createToast({ header }) {
  return () => (
    <Toast>
      <Toast.Header>{ header }</Toast.Header>
      <Toast.Body>Something went wrong.</Toast.Body>
    </Toast>
  );
}
```

This is great if all you wanted to do was create components with different headers, but what about the `<Toast.Body>` component? You could just pass a value in the same way as `header`, but what if you want the body to be different for each instance of the toast? This could be done one of two ways. The toast element could

accept a attribute, or it could use the `children` attribute to use any elements or text nested under the element.

jsx

```jsx
import React from 'react';
import Toast from 'react-bootstrap/Toast';

function createToast({ header }) {
  return ({ children }) => (
    <Toast>
      <Toast.Header>{ header }</Toast.Header>
      <Toast.Body>{ children }</Toast.Body>
    </Toast>
  );
}
```

Now any children of any elements created with this function will be nested under the `<Toast.Body>` element.

One last thing to add to the component is some local state. The React-Bootstrap `<Toast>` component has a `show` attribute which allows it to be hidden as well as an `onClose` attribute which gets called when the close button is clicked.

jsx

```jsx
import React, { useState } from 'react';
import Toast from 'react-bootstrap/Toast';
```

```jsx
  function createToast({ header }) {
    const [show, changeShow] = useState(true);

    return ({ children }) => (
      <Toast show={show} onClose={() => changeShow(false)}>
        <Toast.Header>{ header }</Toast.Header>
        <Toast.Body>{ children }</Toast.Body>
      </Toast>
    );
  }
```

This will allow the toasts to be dismissed.

Using the Dynamic Component Generator

Now that you've created the `createToast` function, it can be used to create new components.

jsx

```jsx
const ErrorToast = createToast({ header: "Error" });
const WarningToast = createToast({ header: "Warning" });
```

Using the Toast Components

With these components created, here's how they would be used in the UI.

jsx

```jsx
import React from 'react';
import Container from 'react-bootstrap/Container';

export default function App() {
  return <Container>
    <ErrorToast>Something went wrong.</ErrorToast>
    <WarningToast>You shouldn't do that.</WarningToast>
  </Container>
}
```

Conclusion

Using dynamic functions to create components is a useful pattern for avoiding duplication. Whenever you find yourself repeating the same pattern across multiple components or thinking of duplicating an existing component to make small modifications, you should consider creating a function to dynamically generate those components for you.


4