The Pragmatic Programmers

Test-Driven React 2

Find Problems Early, Fix Them Quickly, Code with Confidence

Trevor Burnham
edited by Jacquelyn Carter

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Testing Nested Markup

So far, we've used React to encapsulate the functionality of a single DOM element (<button>) in a component (CarouselButton). But React components are capable of doing more than that.

We're going to build a component called CarouselSlide, which will be responsible for rendering several distinct DOM elements:

- An <img> to display the actual image
- A <figcaption> to associate caption text with the image
- Text, some of which will be wrapped in for emphasis
- A <figure> to wrap it all up

We'll take a TDD approach to building this tree while ensuring that the props we provide to CarouselSlide are routed correctly. Start by creating a "stub" of the component, a minimal implementation you can add functionality to later:

```
// src/CarouselSlide.tsx
const CarouselSlide = () => <figure />;

export default CarouselSlide;
```

Now for the tests! A good way to start is to check that the right type of DOM element is rendered:

```
// src/CarouselSlide.test.tsx
import { render, screen } from "@testing-library/react";
import CarouselSlide from "./CarouselSlide";

describe("CarouselSlide", () => {
  it("renders a <figure>", () => {
    render(<CarouselSlide />);
    expect(screen.getByRole("figure")).toBeInTheDocument();
  });
});
```

This test should be green. So let's add more requirements. We want the <figure> to contain two children: an <img> and a <figcaption>. To express that in a test, you'll need to write Testing Library queries for those elements. Testing Library encourages the use of ARIA roles when possible; that way, tests are aligned with best practices for writing accessible markup. The <img> tag has an associated ARIA role: "img". The <figcaption> tag, on the other hand, does not. So we'll use the data-testid attribute again to make it easy to select:

```
// src/CarouselSlide.test.tsx
//...
➤ it("renders an <img> and a <figcaption>", () => {
➤   render(<CarouselSlide />);
```

```
  const figure = screen.getByRole("figure");
  const img = screen.getByRole("img");
  const figcaption = screen.getByTestId("caption");
  expect(figure).toContainElement(img);
  expect(figure).toContainElement(figcaption);
});
//...
```

The new test will be red, since <img> and <figcaption> don't yet exist. Add them to the CarouselSlide render tree:

```
// src/CarouselSlide.tsx
const CarouselSlide = () => (
  <figure>
    <img />
    <figcaption data-testid="caption" />
  </figure>
);

export default CarouselSlide;
```

That should put you in the green. Next, we need to add content. For that, we'll supply three props:

1. imgUrl, a URL for the image displayed in the slide
2. description, a short piece of caption text
3. attribution, the name of image's author

The imgUrl will be used as the src for the <img> tag. Add a test:

```
// src/CarouselSlide.test.tsx
...
it("passes `imgUrl` through to the <img>", () => {
  const imgUrl = "https://example.com/image.png";
  render(<CarouselSlide imgUrl={imgUrl} />);
  expect(screen.getByRole("img")).toHaveAttribute("src", imgUrl);
});
...
```

Modify CarouselSlide so that the imgUrl test turns green:

```
// src/CarouselSlide.tsx
const CarouselSlide = ({ imgUrl }: { imgUrl?: string }) => (
  <figure>
    <img src={imgUrl} />
    <figcaption data-testid="caption" />
  </figure>
);

export default CarouselSlide;
```

Now let's add another requirement. We want to add props called description and attribution, and we want both to be rendered in <figcaption>, with the description bolded by a tag:

```
// src/CarouselSlide.test.tsx
...
it("uses `description` and `attribution` as the caption", () => {
  const props = {
    description: "A jaw-droppingly spectacular image",
    attribution: "Trevor Burnham",
  };
  render(<CarouselSlide {...props} />);
  const figcaption = screen.getByTestId("caption");
  expect(figcaption).toHaveTextContent(
    `${props.description} ${props.attribution}`
  );
});
...
```

Try making all tests pass. When you're done, your implementation should look something like this:

```
// src/CarouselSlide.tsx
import { ReactNode } from "react";

const CarouselSlide = ({
  imgUrl,
  description,
  attribution,
}: {
  imgUrl?: string;
  description?: ReactNode;
  attribution?: ReactNode;
}) => (
  <figure>
    <img src={imgUrl} />
    <figcaption>
      <strong>{description}</strong> {attribution}
    </figcaption>
  </figure>
);

export default CarouselSlide;
```

There's one feature still missing from the component: in order to support styling, we should pass the className and style props through to the <figure>. In fact, for maximum flexibility, we should allow event handlers, data- attributes, etc. In short: we should pass every prop *except* the three we're explicitly using through to the <figure>.

Add a test that sets an arbitrary assortment of props as the finishing touch on CarouselSlide.test.tsx for this chapter:

ch3/src/CarouselSlide.test.tsx

```
import { render, screen } from "@testing-library/react";
import CarouselSlide from "./CarouselSlide";

describe("CarouselSlide", () => {
  it("renders a <figure>", () => {
    render(<CarouselSlide />);
    expect(screen.getByRole("figure")).toBeInTheDocument();
  });

  it("renders an <img> and a <figcaption>", () => {
    render(<CarouselSlide />);
    const figure = screen.getByRole("figure");
    const img = screen.getByRole("img");
    const figcaption = screen.getByTestId("caption");
    expect(figure).toContainElement(img);
    expect(figure).toContainElement(figcaption);
  });

  it("passes `imgUrl` through to the <img>", () => {
    const imgUrl = "https://example.com/image.png";
    render(<CarouselSlide imgUrl={imgUrl} />);
    expect(screen.getByRole("img")).toHaveAttribute("src", imgUrl);
  });

  it("uses `description` and `attribution` as the caption", () => {
    const props = {
      description: "A jaw-droppingly spectacular image",
      attribution: "Trevor Burnham",
    };
    render(<CarouselSlide {...props} />);
    const figcaption = screen.getByTestId("caption");
    expect(figcaption).toHaveTextContent(
      `${props.description} ${props.attribution}`
    );
  });

➤ it("passes other props through to the <figure>", () => {
➤   const props = {
➤     className: "my-carousel-slide",
➤     "data-test-name": "My slide",
➤   };
➤   render(<CarouselSlide {...props} />);
➤   const figure = screen.getByRole("figure");
➤   expect(figure).toHaveClass(props.className);
➤   expect(figure).toHaveAttribute("data-test-name", props["data-test-name"]);
➤ });
});
```

The most common way to implement this functionality is with the *object rest syntax*. Here's what it looks like:

ch3/src/CarouselSlide.tsx

```
import { ComponentPropsWithRef, ReactNode } from "react";

const CarouselSlide = ({
  imgUrl,
  description,
  attribution,
  ...rest
}: {
  imgUrl?: string;
  description?: ReactNode;
  attribution?: ReactNode;
} & ComponentPropsWithRef<"figure">) => (
  <figure {...rest}>
    <img src={imgUrl} />
    <figcaption data-testid="caption">
      <strong>{description}</strong> {attribution}
    </figcaption>
  </figure>
);

export default CarouselSlide;
```

As before, the function only takes a single argument that's destructured into individual variables. However, now there's an object named `rest` that collects all values from the props object that haven't been explicitly destructured.

Conversely, the JSX spread `{...rest}` takes the key-value pairs from the `rest` object and converts them into props. Since `rest` was originally created from the leftover props given to `CarouselSlide`, the effect is to pass those props—everything but `imgUrl`, `description`, and `attribution`—through to the `<figure>`.

You may be familiar with the rest/spread syntax from argument lists and arrays, where it's been supported since ES6. The object rest/spread syntax is newer, and was added to the language as part of the ES2018 specification.

`CarouselSlide` and its tests should be looking ship-shape now! Make a commit:

```
:sparkles: Initial implementation of CarouselSlide
```

Just one component to go: `Carousel` itself.