Mae Capozzi

711 Followers · About Follow

How to Use React.lazy

Mae Capozzi Oct 29, 2018 · 4 min read ★

React 16.6.0 introduced `React.lazy`, which allows you to code-split using the new Suspense API.

Siddharth Kshetrapal came out with a great video showing how this works in 60 seconds. I'd recommend watching it — it's really well-done.

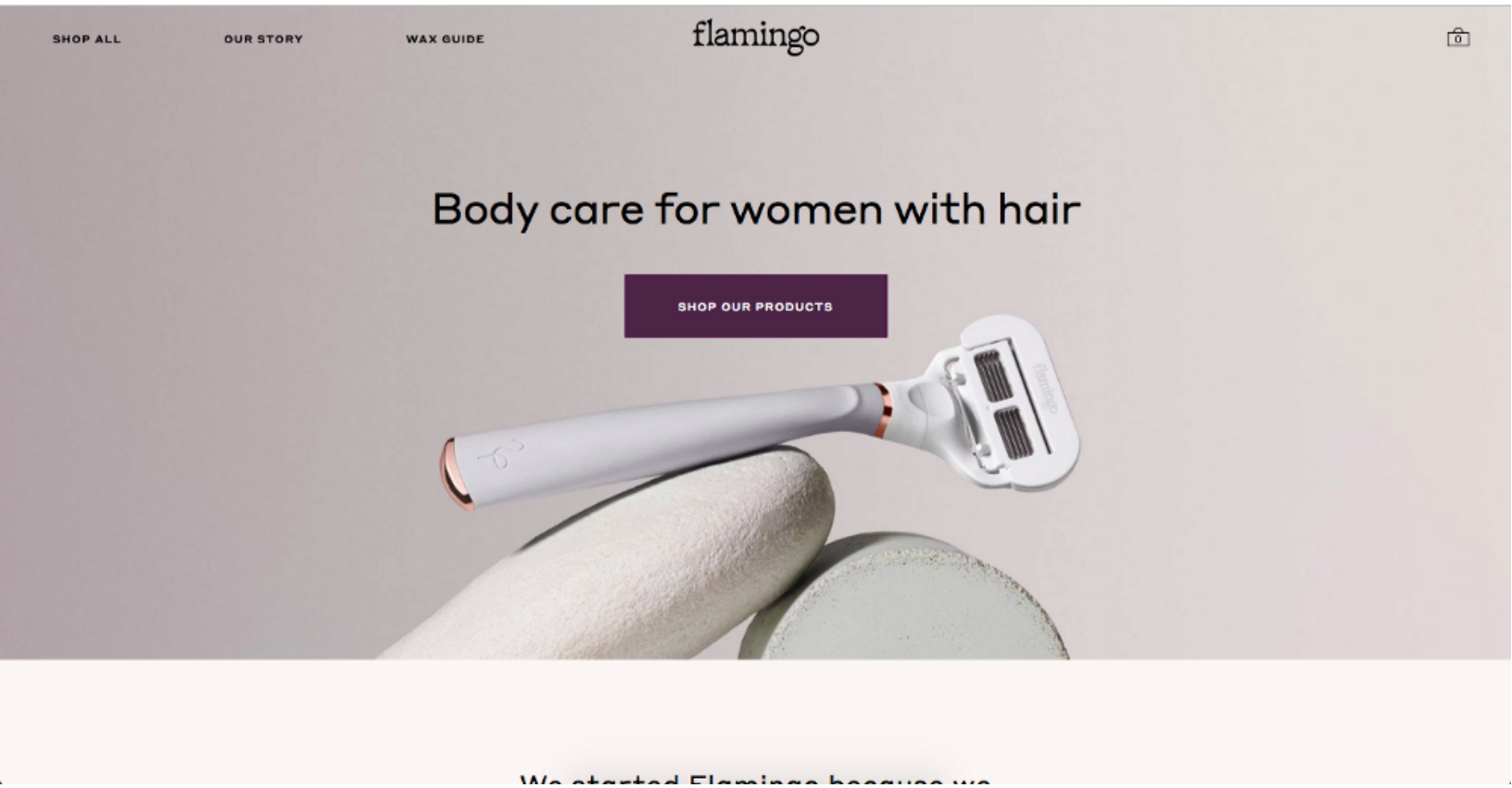
I've built a Github clone based off of Siddharth's example. Hopefully this will help people who prefer reading to watching videos, or who to be walked through it step by step!

Why would I use React.lazy?

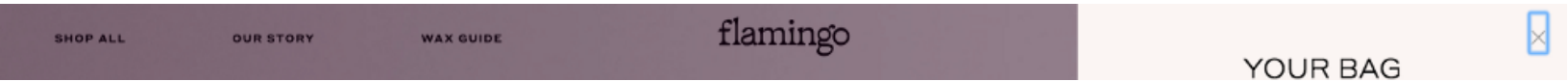
`React.lazy` allows developers to block UI from rendering until a pre-determined condition is met. For example, maybe you don't want your component to render until you get a response back from an endpoint.

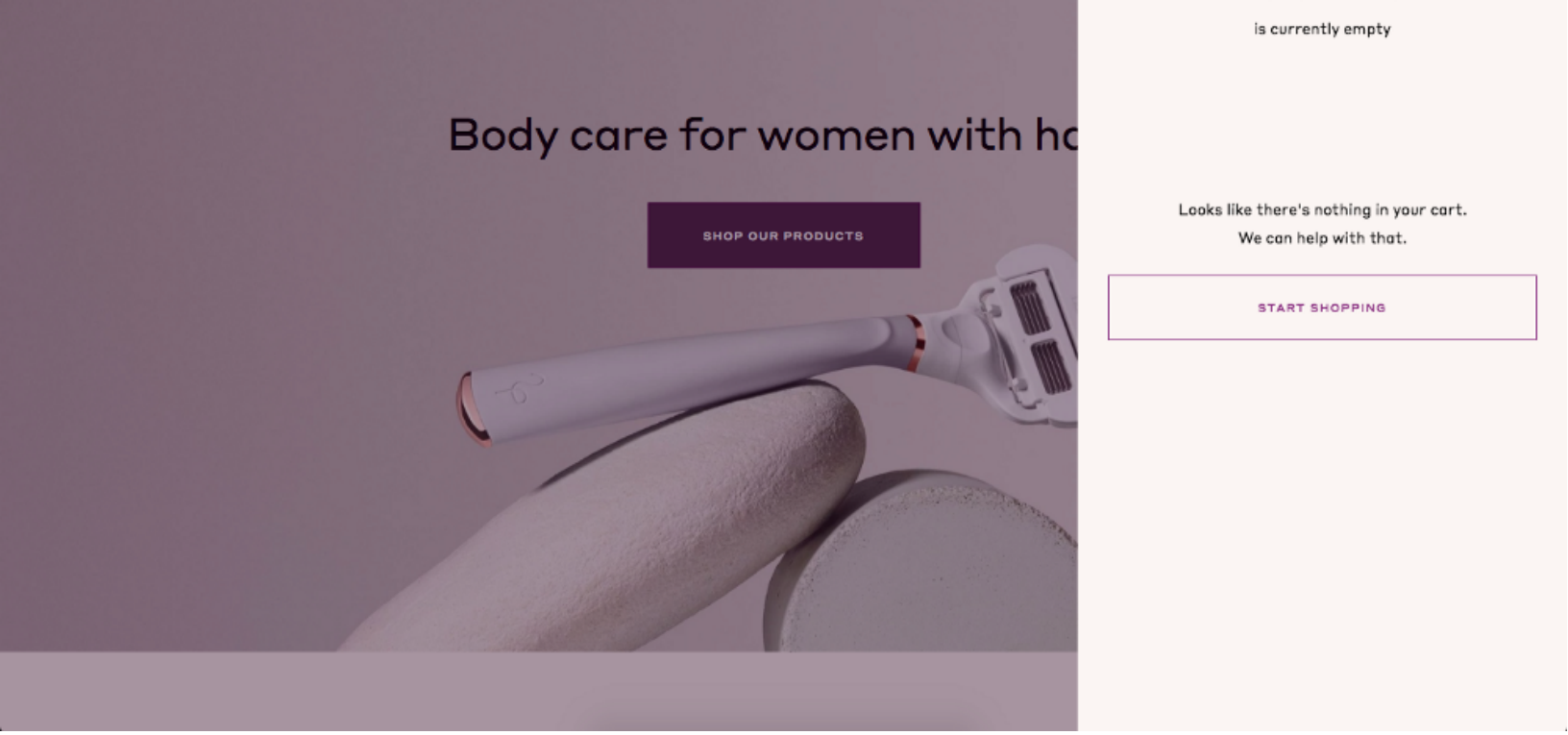
Before `React.lazy`, you had to load all of the files that were ever going to be visible in that view, even if some of them were hidden.

For example, imagine you built an e-commerce site like I'm working on at Harry's: www.shopflamingo.com.



In the upper-right hand corner, we have a cart icon. When you click it, a `PanelOverlay` component slides out from the right-hand side.





Right now, we load the `PanelOverlay` component when we load `/`. But with `React.lazy`, we could dynamically import that component so that it's only being loaded after it has been clicked! That should allow us to load the homepage even more quickly.

. . .

You might be wondering how `lazy()` is different from `react-loadable`, for example. Sean T. Larkin explains it well in this tweet:

`lazy()` lets you code-split without adding another dependency to your codebase. It makes a best practice (code-splitting) easier, which will encourage developer to use it, and will make writing performant React apps more seamless.

Let's Take a Look at an Example

I built a demo of code-splitting with `React.lazy` using Github's API to show you exactly how to implement it in your own application.

. . .

Below is a .gif of how code-splitting would look on a low-end mobile device. I slowed down so it's easy to see every distinct step of the process.

Let's walk through what's happening here:

1. I load the name of the application and a search bar.
2. I search for a Github username.
3. My application fetches data from Github's API and renders the username.
4. The application dynamically imports the `Repos` component.

5. While the application waits for the `Repos` component to load, it displays a blurred out “fallback” UI.
6. The `Repos` component loads.

Let’s also take a look at the network tab so you can see the code-splitting in action.

When we don’t code-split, two chunks are loaded immediately.

On the other hand, when we implement code-splitting, we load two chunks initially, and then a third once we actually search for a Github username.

How Does It Work?

The best part about this React feature is how easy it is to implement.

Let's start with a simple implementation example. @ggrgur provided an awesome example on twitter:

There are three important pieces to code-splitting in React:

1. Importing `Suspense` and `lazy` from React.

```
import React, { Suspense, lazy } from 'React';
```

2. Dynamically importing the file you want to defer loading

```
const Deferred = lazy(() => import("./Deferred"))
```

3. Wrapping that component in `Suspense` and setting a fallback UI.

```
<Suspense fallback={<div>Fallback Component</div>}>
  <Deferred {...props} />
</Suspense>
```

And now I'll share a gist from my actual demo:

JavaScript React Software Development Web Development Coding