Gatsby + loadable-components

This repo is meant to illustrate the pattern of adding loadable-components to a Gatsby project. In this case, the Header component is being code-split out while maintaining SSR.

```
// src/components/layout.jsx
import * as React from "react"

const Header = loadable(() => import("./header"))
export default Header
```

Why?

Why do you need to do this when Gatsby already has excellent out-of-the-box code-splitting configuration?

There is one main scenario: *conditional rendering*. This can come in two forms:

- flex layouts from CMS content
- logic in a component that may or may not render an import

These are essentially the same thing, but at different scales.

Flex Layouts

When you build a page with a headless CMS, parts of your site may have a structured layout that is defined in code by components and can only be changed with a code change. Other pages, due to business cases or user preferences, need to have flexible layouts where a content editor can mix and reorder components in the CMS, and have that layout reflected in the code.

The most common pattern is a switch case or series of if statements checking the component's type, rendering the correct one and passing in its props.

```
// src/components/flex-page.jsx
import * as React from "react"

const FlexPage = ({ components }) => {
  return (
    <>
      {components.map(component => {
        switch (component.type) {
          case "header":
            return <Header {...component} />
          case "text":
            return <Text {...component} />
          case "carousel":
            return <Carousel {...component} />
          default:
            return <div>empty</div>
        }
      })}
    </>
  )
}

export default FlexPage
```

Logical Checks in Render

This is basically the above pattern but on a smaller scale.

```
import * as React from "react"

const FlexPage = ({ hasCarousel }) => {
  return (
    <div>
      <Heading />
      <Text />
      {hasCarousel && <Carousel />}
    </div>
  )
}

export default FlexPage
```

The Conditional Drawback

So what's the problem with the patterns above? A content editor can build a page with a Heading, Text or a Carousel. Technically, the page will display as expected, but imagine a page where the editor only choose a Heading and Text. In that case, the Carousel and its dependencies will still be on the page!

This is because Gatsby's default code splitting configuration we mentioned at the beginning splits bundles by page-level. That means any import in a page template will end up in its bundle whether it is actually rendered, used, or not.

How to Configure

To use loadable-components with SSR, we use a Gatsby plugin, gatsby-plugin-loadable-components-ssr. This makes the webpack configuration we need to use loadable-components while *maintaining SSR*. This is crucial, because if you only code-split, yes the bundle can be smaller, but the client-side rendering will drastically hurt performance more than the bundle reduction could make up. To configure the plugin, follow the instructions here.

Verification

Once you follow this pattern, how can you test if a component is being code-split and SSR'd? The easiest way is through DevTools.

After building the site and serving locally, refresh the page with DevTools open.

First, verify the code-splitting by looking for a JS bundle in the Network tab with the naming pattern: `component--[my-component]-[hash].js` .

Then, to verify SSR is still working, click the 'All' option in the Network tab and look for the first item `localhost` . This is the initial HTML sent by Gatsby. Verify that you can see the code-split component in this initial HTML.

That's it, happy building! 🎉