Debugging HTML Builds

Errors while building static HTML files (the build-time React SSR process) or while using `getServerData` (the runtime SSR process) generally happen for one of the following reasons:

1. Some of your code references "browser globals" like `window` or `document` that aren't available in Node.js. If this is your problem you should see an error above like "window is not defined". To fix this, find the offending code and either a) check before calling the code if window is defined so the code doesn't run while Gatsby is building (see code sample below) or b) if the code is in the render function of a React.js component, move that code into a `componentDidMount` lifecycle or into a `useEffect` hook, which ensures the code doesn't run unless it's in the browser.
2. Check that each of your JS files listed in your `pages` directory (and any sub-directories) are exporting either a React component or string. Gatsby treats any JS file listed under the `pages` dir as a page component, so it must have a default export that's a component or string.
3. You mix up `import` and `require` calls in the same file. This might lead to "WebpackError: Invariant Violation: Minified React error #130" since webpack 4 is stricter than v3. The solution is to only use `import` and this also extends to `gatsby-ssr` and `gatsby-browser` files.
4. Your app doesn't correctly hydrate in the client, which results in gatsby develop and gatsby build being inconsistent. It's possible that a change in a file like `gatsby-ssr` or `gatsby-browser` has a structure that is not reflected in the other file, meaning that there is a mismatch between client and server output.
5. Some other reason :-) #1 is the most common reason building static files fail. If it's another reason, you have to be a bit more creative in figuring out the problem.

How to check if `window` is defined

When referencing `window` in a React component.

```
import * as React from "react"
```

```
// Check if window is defined (so if in the browser or in node.js).
const isBrowser = typeof window !== "undefined"

export default function MyComponent() {
  let loggedIn = false
  if (isBrowser) {
    window.localstorage.getItem("isLoggedIn") === "true"
  }

  return <div>Am I logged in? {loggedIn}</div>
}
```

When requiring a module:

```
// Requiring a function causes an error during builds
// as the code tries to reference window
const module = require("module") // Error

// Wrap the require in check for window
if (typeof window !== `undefined`) {
  const module = require("module")
}
```

In case the module needs to be defined for the code to run, you can use a ternary operator

```
const module = typeof window !== `undefined` ? require("module") : null
```

Fixing third-party modules

So, the worst has happened and you're using an npm module that expects `window` to be defined. You may be able to file an issue and get the module patched, but what to do in the meantime?

One solution is to customize your webpack configuration to replace the offending module with a dummy module during server rendering.

`gatsby-node.js` in the project root:

```
exports.onCreateWebpackConfig = ({ stage, loaders, actions }) => {
  if (stage === "build-html" || stage === "develop-html") {
    actions.setWebpackConfig({
      module: {
        rules: [
          {
            test: /bad-module/,
            use: loaders.null(),
          },
        ],
```

```
      },
    })
  }
}
```

Another solution is to use a package like loadable-components. The module that tries to use `window` will be dynamically loaded only on the client side (and not during SSR).