In general, every website needs assets: images, stylesheets, scripts, etc. When using Gatsby, we recommend Importing Assets Directly in JavaScript files, because of the benefits it provides:

- Scripts and stylesheets are minified and bundled together to avoid extra network requests.
- Missing files cause compilation errors instead of 404 errors for your users.
- Result filenames include content hashes so you don't need to worry about browsers caching their old versions.

However, there is an **escape hatch** that you can use to add an asset outside of the module system.

Adding assets outside of the module system

You can create a folder named `static` at the root of your project. Every file you put into that folder will be copied into the `public` folder. E.g. if you add a file named `sun.jpg` to the static folder, it'll be copied to `public/sun.jpg`

Referencing your static asset

You can reference assets from the `static` folder in your code without anything special required:

```
render() {
  // Note: this is an escape hatch and should be used sparingly!
  // Normally we recommend using `import` for getting asset URLs
  // as described in the "Importing Assets Directly Into Files" page.
  return <img src={'/logo.png'} alt="Logo" />;
}
```

Downsides

Keep in mind the downsides of this approach:

- None of the files in the `static` folder will be post-processed or minified.
- Missing files will not be called at compilation time, and will cause 404 errors for your users.
- Result filenames won't include content hashes, so you'll need to add query arguments or rename them every time they change.

When to use the `static` folder

Normally we recommend importing stylesheets, images, and font assets from JavaScript. The `static` folder is useful as a workaround for a number of less common cases:

- You need a file with a specific name in the build output, such as `manifest.webmanifest`.
- You have thousands of images and need to dynamically reference their paths.
- You want to include a small script like `pace.js` outside of the bundled code.
- Some libraries may be incompatible with webpack and you have no other option but to include it as a `<script>` tag.
- You need to import JSON file that doesn't have a consistent schema, like TopoJSON files, which is difficult to handle with GraphQL. Note that importing JSON files directly inside a page, a template, or a component using `import` syntax results in adding that file to the app bundle and increasing the size of all site's pages. Instead, it's better to place your JSON file inside the `static` folder and use the dynamic import

syntax (`import('/static/myjson.json')`) within the `componentDidMount` lifecycle or the `useEffect` hook.