Importing Assets Directly into Files

There are two major ways to import assets, such as images, fonts, and files, into a Gatsby site. The default path is to import the file directly into a Gatsby template, page, or component. The alternative path, which makes sense for some edge cases, is to use the static folder.

Importing assets with webpack

With webpack you can **`import` a file right in a JavaScript module**. This tells webpack to include that file in the bundle. Unlike CSS imports, importing a file gives you a string value. This imported file's value is the final path you can reference in your code, e.g. as the `src` attribute of an image or the `href` of a link to a PDF.

To reduce the number of requests to the server, importing images that are less than 10,000 bytes returns a data URI instead of a path. This applies to the following file extensions: `svg`, `jpg`, `jpeg`, `png`, `gif`, `mp4`, `webm`, `wav`, `mp3`, `m4a`, `aac`, and `oga`.

Here's an example:

```
import React from "react"
import logo from "./logo.png" // Tell webpack this JS file uses this image

console.log(logo) // /logo.84287d09.png

function Header() {
  // Import result is the URL of your image
  return <img src={logo} alt="Logo" />
}

export default Header
```

This ensures that when the project is built, webpack will correctly move the images into the public folder, and provide us with correct paths.

You can reference files in CSS to import them, too:

```
.Logo {
  background-image: url(./logo.png);
}
```

webpack finds all relative module references in CSS (they start with `./`) and replaces them with the final paths from the compiled bundle. If you make a typo or accidentally delete an important file, you will see a compilation error, just like when you import a non-existent JavaScript module. The final filenames in the compiled bundle are generated by webpack from content hashes. If the file content changes in the future, webpack will give it a different name in production so you don't need to worry about long-term caching of assets.

If you're using SCSS the imports are relative to the entry SCSS file.

Please be advised that this is also a custom feature of webpack.

Additional resources

- More on using an imported font.

Querying for a `File` in GraphQL using gatsby-source-filesystem

You can also import files using GraphQL by querying for them in your data layer, which will trigger copying of those files to the public directory. Querying for the `publicURL` field of `File` nodes will provide URLs you can use in your JavaScript components, pages and templates.

Examples

1. Copy all `.pdf` files you have in your data layer to your build directory and return URLs to them:

```
{
  allFile(filter: { extension: { eq: "pdf" } }) {
    edges {
      node {
        publicURL
      }
    }
  }
}
```

Reference those PDF files in a page with useStaticQuery:

```
import React from "react"
import { useStaticQuery, graphql } from "gatsby"

import Layout from "../components/layout"

const DownloadsPage = () => {
  const data = useStaticQuery(graphql`
    {
      allFile(filter: { extension: { eq: "pdf" } }) {
        edges {
          node {
            publicURL
            name
          }
        }
      }
    }
  `)
  return (
    <Layout>
      <h1>All PDF Downloads</h1>
      <ul>
        {data.allFile.edges.map((file, index) => {
          return (
            <li key={`pdf-${index}`}>
              <a href={file.node.publicURL} download>
                {file.node.name}
              </a>
            </li>
          )
        })}
      </ul>
    </Layout>
  )
}
export default DownloadsPage
```

2. Link to attachments in the frontmatter of your Markdown files:

```
---
title: "Title of article"
attachments:
  - "./assets.zip"
  - "./presentation.pdf"
---

Hi, this is a great article.
```

In an article template component file, you can then query for the attachments:

```
query ($slug: String!) {
  markdownRemark(fields: { slug: { eq: $slug } }) {
    html
    frontmatter {
      title
      attachments {
        publicURL
      }
    }
  }
}
```

Read more about querying for files in `gatsby-source-filesystem`.