Part of what makes Gatsby sites so fast is that a lot of the work is done at build time and the running site is static content served from a CDN. During the build process, Gatsby creates paths to access pages, handling routing for you. Creating navigation for a Gatsby app requires an understanding of what those paths are and how they're generated.

Often your application will include routes that are not known at build time. This includes routes that need authentication or for dynamic content. To handle those pages, you can make use of client-only routes using `@reach/router` which is built into Gatsby.

Creating routes

Routes can be created in three ways:

- By creating React components in `src/pages`
- By using the File System Route API to programmatically create pages from GraphQL and to create client-only routes.
- By implementing the API `createPages` in your site's `gatsby-node.js`. (Plugins can also implement `createPages` and create pages for you.)

Define routes in `src/pages`

Gatsby generates pages for each `.js` file inside `src/pages`. The browser path is generated from the file path.

Add this component to `src/pages/index.js` to create a home page for your site.

```
import * as React from "react"

export default function Index() {
  return <div>Hello world</div>
}
```

For example, `src/pages/contact.js` creates the page `yoursite.com/contact`, and `src/pages/home.js` creates the page `yoursite.com/home`. This works at whatever level the file is created at. If `contact.js` is moved to a directory called `information`, located inside `src/pages`, the page will now be created at `yoursite.com/information/contact`.

The exception to this rule is any file named `index.js`. Files with this name are matched to the root directory they're found in. That means `index.js` in the root `src/pages` directory is accessed via `yoursite.com`. However, if there is an `index.js` inside the `information` directory, it is created at `yoursite.com/information`.

Path	Route
`src/pages/contact.js`	`yoursite.com/contact`
`src/pages/information/contact.js`	`yoursite.com/information/contact`

Note that if a particular directory does not have an `index.js` file, then that root page is not created, and attempts to navigate to it will land you on a 404 page. For example, `yoursite.com/information/contact` may exist, but that does not guarantee `yoursite.com/information` exists.

Using the File System Route API

Other than creating single-page routes in `src/pages` you can also create multiple pages from a model based on the collection of nodes within it. To do that, use curly braces (`{ }`) in the file path to signify dynamic URL segments that relate to a field within the node.

Use the File System Route API when you want to programmatically create pages from your GraphQL data, e.g. to create individual blog post pages for your blog. With this API you can control the file path and queried data by adding some extra notation to the names of your files without touching or creating `gatsby-node.js` whatsoever.

For example, assuming you have a model called `Product` :

- `src/pages/products/{Product.name}.js => /products/burger`

See the File System Route API documentation for more detail.

Using `gatsby-node.js`

The File System Route API should be enough to get you through most use cases but if you need extra control, e.g. for passing data via `pageContext` or modifying the `path` , you can use Gatsby Node APIs, including the `createPages` function, inside your `gatsby-node.js` file. This function will give you access to the `createPage` action, which is at the core of programmatically creating a page. Here's an example for creating pages from Markdown files sourced by Gatsby's data layer:

```
exports.createPages = async function ({ actions, graphql }) {
  const { data } = await graphql(`
    query {
      allMarkdownRemark {
        nodes {
          fields {
            slug
          }
        }
      }
    }
  `)
  // highlight-start
  data.allMarkdownRemark.forEach(node => {
    const slug = node.fields.slug
    actions.createPage({
      path: slug,
      component: require.resolve(`./src/templates/blog-post.js`),
      context: { slug: slug },
    })
  })
  // highlight-end
}
```

The data for creating these pages doesn't necessarily have to come from Gatsby's internal GraphQL data layer. For example, you can source local files or make async calls to remote APIs. For more information, please see Creating and Modifying Pages.

Conflicting Routes

Since there are multiple ways to create a page, different plugins, themes, or sections of code in your `gatsby-node` file may accidentally create multiple pages that are meant to be accessed by the same path. When this happens, Gatsby will show a warning at build time, but the site will still build successfully. In this situation, the page that was built last will be accessible and any other conflicting pages will not be. Changing any conflicting paths to produce unique URLs should clear up the problem.

Nested Routes

If your goal is to define paths that are multiple levels deep, such as `/portfolio/art/item1`, that can be done directly when creating pages as mentioned in Routes defined in `src/pages`.

Alternatively, if you want to create pages that will display different subcomponents depending on the URL path (such as a specific sidebar widget), Gatsby can handle that at the page level using layouts.

Linking between routes

In order to link between pages, you can use `gatsby-link`. Using `gatsby-link` gives you built in performance benefits.

Alternatively, you can navigate between pages using standard `<a>` tags, but you won't get the benefit of prefetching in this case.

Gatsby will handle scroll restoration for you in most cases. To track and restore scroll position in additional containers, you can use the `useScrollRestoration` hook.

Creating authentication-gated routes

For pages dealing with sensitive information, or other dynamic behavior, you may want to handle that information server-side. Gatsby lets you create client-only routes that live behind an authentication gate, ensuring that the information is only available to authorized users.

Performance and Prefetching

In order to improve performance, Gatsby looks for links that appear on the current page to perform prefetching. Before a user has even clicked on a link, Gatsby has started to fetch the page it points to. Learn more about prefetching.