Use the File System Route API when you want to create dynamic pages e.g. to create individual blog post pages for your blog.

You should be able to accomplish most common tasks with this file-based API. If you want more control over the page creation you should use the `createPages` API.

Dynamic pages can be created from collections in Gatsby's GraphQL data layer and to create client-only routes.

A complete example showcasing all options can be found in Gatsby's examples folder.

Collection routes

Imagine a Gatsby project that sources a `product.yaml` file and multiple Markdown blog posts. At build time, Gatsby will automatically infer the fields and create multiple nodes for both types (`Product` and `MarkdownRemark`).

To create collection routes, use curly braces (`{ }`) in your filenames to signify dynamic URL segments that relate to a field within the node. Here are a few examples:

- `src/pages/products/{Product.name}.js` will generate a route like `/products/burger`
- `src/pages/products/{Product.fields__sku}.js` will generate a route like `/products/001923`
- `src/pages/blog/{MarkdownRemark.parent__(File)__name}.js` will generate a route like `/blog/learning-gatsby`

Gatsby creates a page for each node in a collection route. So if you have three markdown files that are blog posts, Gatsby will create the three pages from a collection route. As you add and remove markdown files, Gatsby will add and remove pages.

Collection routes can be created for any GraphQL data type. Creating new collection routes in Gatsby is a process of adding a source plugin, use GraphiQL to identify the type and field to construct the route file name, and then code the route component.

Syntax (collection routes)

There are some general syntax requirements when using collection routes:

- Dynamic segments of file paths must start and end with curly braces (`{ }`).
- Types are case-sensitive (e.g. `MarkdownRemark` or `contentfulMyContentType`). Check GraphiQL for the correct names.
- Dynamic segments must include both a type and a field e.g. `{Type.field}` or `{BlogPost.slug}` .

Nested routes

You can use dynamic segments multiple times in a path. For example, you might want to nest product names within its product category. For example:

- `src/pages/products/{Product.category}/{Product.name}.js` will generate a route like `/products/toys/fidget-spinner`
- `src/pages/products/{Product.category}/{Product.name}/{Product.color}.js` will generate a route like `/products/toys/fidget-spinner/red`

Field syntax

Dot notation

Using `.` you signify that you want to access a field on a node of a type.

`src/pages/products/{Product.name}.js` generates the following query:

```
allProduct {
  nodes {
    id # Gatsby always queries for id
    name
  }
}
```

Underscore notation

Using `__` (double underscore) you signify that you want to access a nested field on a node.

`src/pages/products/{Product.fields__sku}.js` generates the following query:

```
allProduct {
  nodes {
    id # Gatsby always queries for id
    fields {
      sku
    }
  }
}
```

You can nest as deep as necessary, e.g. `src/pages/products/{Product.fields__date__createdAt}.js` generates the following query:

```
allProduct {
  nodes {
    id # Gatsby always queries for id
    fields {
      date {
        createdAt
      }
    }
  }
}
```

Parentheses notation

Using `( )` you signify that you want to access a [GraphQL union type](). This is often possible with types that Gatsby creates for you. For example, `MarkdownRemark` always has `File` as a parent type, and thus you can also access fields from the `File` node. You can use this multiple levels deep, too, e.g. `src/pages/blog/{Post.parent__(MarkdownRemark)__parent__(File)__name}.js` .

`src/pages/blog/{MarkdownRemark.parent__(File)__name}.js` generates the following query:

```
allMarkdownRemark {
  nodes {
    id # Gatsby always queries for id
    parent {
      … on File {
        name
      }
    }
  }
}
```

Collection Route Components

Collection route components are passed two dynamic variables. The `id` of each page's node and the URL path as `params`. The params is passed to the component as `props.params` and the id as `props.pageContext.id`.

Both are also passed as variables to the component's GraphQL query so you can query fields from the node. Page querying, including the use of variables, is explained in more depth in querying data in pages with GraphQL.

For example:

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

export default function Component(props) {
  return props.data.fields.sku + props.params.name // highlight-line
}

// This is the page query that connects the data to the actual component. Here you
can query for any and all fields
// you need access to within your code. Again, since Gatsby always queries for `id`
in the collection, you can use that
// to connect to this GraphQL query.

export const query = graphql`
  query($id: String) {
    product(id: { eq: $id }) {
      fields {
        sku
      }
    }
  }
`
```

For the page `src/pages/{Product.name}/{Product.coupon}.js` you'd have `props.params.name` and `props.params.coupon` available inside `{Product.coupon}.js`.

If you need to want to create pages for only some nodes in a collection (e.g. filtering out any product of type `"Food"`) or customize the variables passed to the query, you should use the `createPages` API instead as File System Route API doesn't support this at the moment.

Routing and linking

Gatsby "slugifies" every route that gets created from collection pages (by using [`sindresorhus/slugify`]). Or in other words: If you have a route called `src/pages/wholesome/{Animal.slogan}.js` where `slogan` is `I ♥ Dogs` the final URL will be `/wholesome/i-love-dogs`. Gatsby will convert the field into a human-readable URL format while stripping it of invalid characters.

When you want to link to a collection route page, it may not always be clear how to construct the URL from scratch.

To address this issue, Gatsby automatically includes a `gatsbyPath` field on every type used by collection pages. The `gatsbyPath` field must take an argument of the `filePath` it is trying to resolve. This is necessary because it’s possible that one type is used in multiple collection pages.

There are some general syntax requirements when using the `filePath` argument:

- The path must be an absolute path (starting with a `/`).
- You must omit the file extension.
- You must omit the `src/pages` prefix.
- Your path must not include `index`.

`gatsbyPath` example

Assume that a `Product` type is used in two pages:

- `src/pages/products/{Product.name}.js`
- `src/pages/discounts/{Product.name}.js`

If you wanted to link to the `products/{Product.name}` and `discounts/{Product.name}` routes from your home page, you would have a component like this:

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

export default function HomePage(props) {
  return (
    <ul>
      {props.data.allProduct.map(product => (
        <li key={product.name}>
          <Link to={product.productPath}>{product.name}</Link> (
          <Link to={product.discountPath}>Discount</Link>) // highlight-line
        </li>
      ))}
    </ul>
  )
}

export const query = graphql`
  query {
    allProduct {
      name
      productPath: gatsbyPath(filePath: "/products/{Product.name}") // highlight-
line
      discountPath: gatsbyPath(filePath: "/discounts/{Product.name}") // highlight-
line
    }
  }
`
```

By using aliasing you can use `gatsbyPath` multiple times.

Creating client-only routes

Use client-only routes if you have dynamic data that does not live in Gatsby. This might be something like a user settings page, or some other dynamic content that isn't known to Gatsby at build time. In these situations, you will usually create a route with one or more dynamic segments to query data from a server in order to render your page.

Syntax (client-only routes)

You can use square brackets (`[ ]`) in the file path to mark any dynamic segments of the URL. For example, in order to edit a user, you might want a route like `/user/:id` to fetch the data for whatever `id` is passed into the URL.

- `src/pages/users/[id].js` will generate a route like `/users/:id`
- `src/pages/users/[id]/group/[groupId].js` will generate a route like `/users/:id/group/:groupId`

Splat routes

Gatsby also supports *splat* (or wildcard) routes, which are routes that will match *anything* after the splat. These are less common, but still have use cases. Use three periods in square brackets (`[...]`) in a file path to mark a page as a splat route. You can also name the parameter your page receives by adding a name after the three periods (`[...myNameKey]`).

As an example, suppose that you are rendering images from S3 and the URL is actually the key to the asset in AWS. Here is how you might create your file:

- `src/pages/image/[...].js` will generate a route like `/image/*`. `*` is accessible in your page's received properties with the key name `*`.
- `src/pages/image/[...awsKey].js` will generate a route like `/image/*awsKey`. `*awsKey` is accessible in your page's received properties with the key name `awsKey`.

```
export default function ImagePage({ params }) {
  const param = params[`*`]

  // When visiting a route like `image/hello/world`,
  // the value of `param` is `hello/world`.
}
```

```
export default function ImagePage({ params }) {
  const param = params[`awsKey`]

  // When visiting a route like `image/hello/world`,
  // the value of `param` is `hello/world`.
}
```

Splat routes may not live in the same directory as regular client only routes.

Examples

The dynamic segment of the file name (the part between the square brackets) will be filled in and provided to your components on a `props.params` object. For example:

```
function UserPage(props) {
  const name = props.params.name
}
```

```
function ProductsPage(props) {
  const splat = props.params.awsKey
}
```

```
function AppPage(props) {
  const splat = props.params['*']
}
```

`config` function

Inside a File System Route template you can export an async function called `config`. You can use this function to:

- Mark the page as deferred or not (see Deferred Static Generation API reference)

Inside your template:

```
import { graphql } from "gatsby"

// The rest of your page, including imports, page component & page query etc.

export async function config() {
  const { data } = graphql`
    {
      # Your GraphQL query
    }
  `

  return ({ params }) => {
    return {
      defer: params.name === data.someValue.name,
    }
  }
}
```

When you export an async `config` function Gatsby will evaluate the returned object and optionally run any GraphQL queries defined inside the outer function. You can't run GraphQL queries inside the inner function.

The `params` parameter is an object that contains the URL path, see explanation above.

The inner function of `config` can return an object with one key:

- `defer` : Boolean of whether the page should be marked as deferred or not

Read the Deferred Static Generation guide to see a real-world example.

Example use cases

Have a look at the route-api example for more detail.

Collection route + fallback

By using a combination of a collection route with a client-only route, you can create a seamless experience when a user tries to visit a URL from the collection route that doesn't exist (yet) for the collection item. Consider these two file paths:

- `src/pages/products/{Product.name}.js` (collection route)
- `src/pages/products/[name].js` (client-only route, fallback)

The collection route will create all available product pages at the time of the build. If you're adding a new product you want to link to but only periodically building your site, you'll need a fallback. By using a client-only route as a fallback you then can load the necessary information for the product on the client until you re-built your site.

Similarly, the fallback page could also be used for when a product doesn't exist and you want to show some helpful information (like a 404 page).

Using one template for multiple routes

By placing the template/view for your routes into a reusable component you can display the same information under different routes. Take this example:

You want to display product information which is both accessible by name and SKU but has the same design. Create two file paths first:

- `src/pages/products/{Product.name}.js`
- `src/pages/products/{Product.meta__sku}.js`

Create a view component at `src/view/product-view.js` that takes in a `product` prop. Use that component in both collection routes, e.g.:

```
import React from "react"
import { graphql } from "gatsby"
import ProductView from "../../views/product-view"

function Product(props) {
  const { product } = props.data
  return <ProductView product={product} />
}

export default Product

export const query = graphql`
  query($id: String!) {
    product(id: { eq: $id }) {
      name
      description
      appearance
      meta {
        createdAt
        id
        sku
      }
    }
  }
`
```

You can copy the same code to the `src/pages/products/{Product.meta__sku}.js` file.

Purely client-only app

If you want your Gatsby app to be 100% client-only, you can create a file at `src/pages/[...].js` to catch all requests. See the [client-only-paths example]() for more detail.