Creating Dynamic Navigation in Gatsby

At times you will want to be able to edit your website's navigation in response to a *change in requirements*. To achieve this, you can use Gatsby to dynamically generate your navigation. Where you store the data for your navigation can be anywhere - a backend API, CMS, headless CMS or even the filesystem.

What this section will cover:

- Adding data to your site's configuration
- Querying the data using GraphQL
- Pulling the data into a component using StaticQuery
- Dynamically rendering your navigation

You will be using `gatsby-config.js` to store the data for your links. `gatsby-config.js` is a file used for configuring Gatsby, located in the root path of every Gatsby project. A plain old JavaScript object is exported from this file; this object contains the `siteMetadata` object which you can query through GraphQL when generating your static pages.

This guide will use the Gatsby starter project `gatsby-starter-default`, which can be downloaded through the Gatsby command line interface tool using the command `gatsby new [project-name] https://github.com/gatsbyjs/gatsby-starter-default`.

Creating the link data

First, locate the `gatsby-config.js` file in the root directory of your project. Inside the `siteMetadata` object, add an array of menu link objects. These objects should contain two properties: `name` and `link`. `name` is the name of your navigation item and `link` is the page which will be navigated to when a menu item is clicked.

```
module.exports = {
  siteMetadata: {
    title: 'Gatsby Default Starter',
+    menuLinks:[
+      {
+         name:'home',
+         link:'/'
+      },
```

```
+      {
+         name:'page2',
+         link:'/page-2'
+      }
+    ]
  },
  plugins: []
}
```

Viewing the `siteMetadata` in GraphQL

GraphQL can be used to query for information contained in the `siteMetadata` object located in your project's `gatsby-config.js`. In order to test this out, you can start the `default-starter-project` in development mode by running `npm run develop`.

Navigate to `http://localhost:8000/___graphql` in your browser to view the GraphiQL editor, which enables you to test GraphQL queries on the underlying APIs. Using the documentation explorer you can view the current GraphQL schema for your project, which is an invaluable resource during development.

Examining the available types in GraphQL you will notice that you can query `site`. This GraphQL type further returns the `siteMetadata` which needs to be accessed to create the dynamic navigation. At this point, it is useful if you know a little GraphQL in order to extract the menu links. If you are unfamiliar with GraphQL, there is some excellent documentation available at GraphQL's official website found here that you can use to brush up on your skills! The query below will return the menu links.

```
query SiteQuery {
  site {
    siteMetadata {
      title
      menuLinks {
        name
        link
      }
    }
  }
}
```

When executing this query within the GraphiQL editor you see output that looks similar to the following:

```
{
  "data": {
    "site": {
      "siteMetadata": {
```

```
        "title": "Gatsby Default Starter",
        "menuLinks": [
          {
            "name": "home",
            "link": "/"
          },
          {
            "name": "page2",
            "link": "/page-2"
          }
        ]
      }
    }
  }
}
```

Perfect! You now have a way of obtaining data from the `gatsby-config.js` file. Continue by pulling this data into the layout using the query you just formed.

Pulling data inside the layout component

Inside your project, locate the `src/components` folder and navigate to the `layout.js` file. Within this layout component, you should notice a component named `StaticQuery`.

StaticQuery is a new component introduced in Gatsby V2, which allows you to run GraphQL queries within your components, not just pages. It allows developers to collocate data with their components.

Extend the query within this component to include the menu links, so it looks like so:

```
const Layout = ({ children }) => (
  <StaticQuery
    query={graphql`
      query SiteTitleQuery {
        site {
          siteMetadata {
            title
+            menuLinks {
+              name
+              link
+            }
          }
        }
      }
    `}
    render={data => (
```

```
      <React.Fragment>
        <Helmet
          title={'title'}
          meta={[
            { name: 'description', content: 'Sample' },
            { name: 'keywords', content: 'sample, something' },
          ]}
        >
        </Helmet>
        <Header siteTitle={data.site.siteMetadata.title} />
        <div
          style={{
            margin: '0 auto',
            maxWidth: 960,
            padding: '0px 1.0875rem 1.45rem',
            paddingTop: 0,
          }}
        >
          {children}
        </div>
      </React.Fragment>
    )}
  />
)
```

With the above changes to your `StaticQuery` component, the `render` property, which accepts a function that takes one argument, now has access to the menu links for use inside the function (as the argument). The last thing that is left to do is to display the site's navigation.

To do this, the header component that is already available in the project seems like it might be a good starting place to display the navigation. Pass the `menuLinks` object to this header component like so:

```
const Layout = ({ children }) => (
  <StaticQuery
    query={graphql`
      query SiteTitleQuery {
        site {
          siteMetadata {
            title
+            menuLinks {
+              name
+              link
+            }
          }
        }
```

```
      }
    `}
    render={data => (
      <React.Fragment>
        <Helmet
          title={'title'}
          meta={[
            { name: 'description', content: 'Sample' },
            { name: 'keywords', content: 'sample, something' },
          ]}
        >
        </Helmet>
-       <Header siteTitle={data.site.siteMetadata.title} />
+       <Header menuLinks={data.site.siteMetadata.menuLinks} siteTitle={data.site.siteMetadat
        <div
          style={{
            margin: '0 auto',
            maxWidth: 960,
            padding: '0px 1.0875rem 1.45rem',
            paddingTop: 0,
          }}
        >
          {children}
        </div>
      </React.Fragment>
    )}
  />
)
```

Using the header component to display the navigation

Locate the `header.js` file inside `src/components` and remove everything so only the functional component definition is left (everything else is boilerplate code given to you when generating your project):

```
import React from "react"
import { Link } from "gatsby"
import PropTypes from "prop-types"

const Header = ({ siteTitle, menuLinks }) => (
  <header
    style={{
      background: "rebeccapurple",
      marginBottom: "1.45rem",
    }}
  >
```

```
<div
  style={{
    background: "rebeccapurple",
    marginBottom: "1.45rem",
  }}
>
  <div
    style={{
      margin: "0 auto",
      maxWidth: 960,
      padding: "1.45rem 1.0875rem",
      display: "flex",
      justifyItems: "space-between",
      alignItems: "center",
    }}
  >
    <h1 style={{ margin: 0, flex: 1 }}>
      <Link
        to="/"
        style={{
          color: "white",
          textDecoration: "none",
        }}
      >
        {siteTitle}
      </Link>
    </h1>
    // highlight-start
    <div>
      <nav>
        <ul style={{ display: "flex", flex: 1 }}>
          {menuLinks.map(link => (
            <li
              key={link.name}
              style={{
                listStyleType: `none`,
                padding: `1rem`,
              }}
            >
              <Link style={{ color: `white` }} to={link.link}>
                {link.name}
              </Link>
            </li>
          ))}
        </ul>
      </nav>
```

```
        </div>
        // highlight-end
      </div>
    </div>
  </header>
)

Header.propTypes = {
  siteTitle: PropTypes.string,
}

Header.defaultProps = {
  siteTitle: ``,
}

export default Header
```

Starting the development server by running `npm run develop` and navigating to `http://localhost:8000` you should now see some dynamically generated menu links on your page.

If you have made it this far, good job! You can now add new site links to your website dynamically by adding entries to the `gatsby-config.js` file.

Where to next?

Be sure to check out more documentation for further in-depth examples and guides on achieving tasks using Gatsby.

- Authentication in Gatsby
- E-commerce in Gatsby