Creating Tags Pages for Blog Posts

Creating tag pages for your blog post is a way to let visitors browse related content.

To add tags to your blog posts, you will first want to have your site set up to turn your markdown pages into blog posts. To get your blog pages set up, see the tutorial on Gatsby's data layer and Adding Markdown Pages.

The process will essentially look like this:

1. Add tags to your `markdown` files
2. Write a query to get all tags for your posts
3. Make a tags page template (for `/tags/{tag}`)
4. Modify `gatsby-node.js` to render pages using that template
5. Make a tags index page (`/tags`) that renders a list of all tags
6. *(optional)* Render tags inline with your blog posts

Add tags to your `markdown` files

You add tags by defining them in the `frontmatter` of your Markdown file. The `frontmatter` is the area at the top surrounded by dashes that includes post data like the title and date.

```
---
title: "A Trip To the Zoo"
---

I went to the zoo today. It was terrible.
```

Fields can be strings, numbers, or arrays. Since a post can usually have many tags, it makes sense to define it as an array. Here you add your new tags field:

```
---
title: "A Trip To the Zoo"
tags: ["animals", "Chicago", "zoos"]
---

I went to the zoo today. It was terrible.
```

If `gatsby develop` is running, restart it so Gatsby can pick up the new fields.

Write a query to get all tags for your posts

Now, these fields are available in the data layer. To use field data, query it using `graphql`. All fields are available to query inside `frontmatter`

Try running the following query in GraphiQL (`http://localhost:8000/___graphql`):

```
{
  allMarkdownRemark {
    group(field: frontmatter___tags) {
      tag: fieldValue
      totalCount
    }
  }
}
```

The above query groups posts by `tags`, and returns each `tag` with the number of posts as `totalCount`. As an addition, you could extract some post data in each group if you need to. To keep this tutorial small, you're only using the tag name in your tag pages. Make the tag page template now:

Make a tags page template (for `/tags/{tag}`)

If you followed the How-To Guide for Adding Markdown Pages, then this process should sound familiar: Make a tag page template, then use it in `createPages` in `gatsby-node.js` to generate individual pages for the tags in your posts.

First, you'll add a tags template at `src/templates/tags.js`:

```
import React from "react"
import PropTypes from "prop-types"

// Components
import { Link, graphql } from "gatsby"

const Tags = ({ pageContext, data }) => {
  const { tag } = pageContext
  const { edges, totalCount } = data.allMarkdownRemark
  const tagHeader = `${totalCount} post${
    totalCount === 1 ? "" : "s"
  } tagged with "${tag}"`

  return (
    <div>
      <h1>{tagHeader}</h1>
      <ul>
        {edges.map(({ node }) => {
          const { slug } = node.fields
          const { title } = node.frontmatter
```

```
          return (
            <li key={slug}>
              <Link to={slug}>{title}</Link>
            </li>
          )
        })}
      </ul>
      {/*
              This links to a page that does not yet exist.
              You'll come back to it!
            */}
      <Link to="/tags">All tags</Link>
    </div>
  )
}

Tags.propTypes = {
  pageContext: PropTypes.shape({
    tag: PropTypes.string.isRequired,
  }),
  data: PropTypes.shape({
    allMarkdownRemark: PropTypes.shape({
      totalCount: PropTypes.number.isRequired,
      edges: PropTypes.arrayOf(
        PropTypes.shape({
          node: PropTypes.shape({
            frontmatter: PropTypes.shape({
              title: PropTypes.string.isRequired,
            }),
            fields: PropTypes.shape({
              slug: PropTypes.string.isRequired,
            }),
          }),
        }).isRequired
      ),
    }),
  }),
}

export default Tags

export const pageQuery = graphql`
  query($tag: String) {
    allMarkdownRemark(
      limit: 2000
      sort: { fields: [frontmatter___date], order: DESC }
```

```
      filter: { frontmatter: { tags: { in: [$tag] } } }
    ) {
      totalCount
      edges {
        node {
          fields {
            slug
          }
          frontmatter {
            title
          }
        }
      }
    }
  }
`
```

Note: `propTypes` are included in this example to help you ensure you're getting all the data you need in the component, and to help serve as a guide while destructuring / using those props.

Modify `gatsby-node.js` to render pages using that template

Now you've got a template. Great! Assuming you followed the How-To Guide for Adding Markdown Pages and provide a sample `createPages` that generates post pages as well as tag pages. In the site's `gatsby-node.js` file, include `lodash` (`const _ = require('lodash')`) and then make sure your `createPages` looks something like this:

```
const path = require("path")
const _ = require("lodash")

exports.createPages = async ({ actions, graphql, reporter }) => {
  const { createPage } = actions

  const blogPostTemplate = path.resolve("src/templates/blog.js")
  const tagTemplate = path.resolve("src/templates/tags.js")

  const result = await graphql(`
    {
      postsRemark: allMarkdownRemark(
        sort: { order: DESC, fields: [frontmatter___date] }
        limit: 2000
      ) {
        edges {
          node {
            fields {
```

```
            slug
          }
          frontmatter {
            tags
          }
        }
      }
    }
    tagsGroup: allMarkdownRemark(limit: 2000) {
      group(field: frontmatter___tags) {
        fieldValue
      }
    }
  }
`)

// handle errors
if (result.errors) {
  reporter.panicOnBuild(`Error while running GraphQL query.`)
  return
}

const posts = result.data.postsRemark.edges

// Create post detail pages
posts.forEach(({ node }) => {
  createPage({
    path: node.fields.slug,
    component: blogPostTemplate,
  })
})

// Extract tag data from query
const tags = result.data.tagsGroup.group

// Make tag pages
tags.forEach(tag => {
  createPage({
    path: `/tags/${_.kebabCase(tag.fieldValue)}/`,
    component: tagTemplate,
    context: {
      tag: tag.fieldValue,
    },
  })
})
}
```

Some notes:

- Your GraphQL query only looks for data you need to generate these pages. Anything else can be queried again later (and, if you notice, you do this above in the tags template for the post title).
- You have referenced two `allMarkdownRemark` fields in your query. To avoid naming collisions you must alias one of them. You alias both to make your code more human-readable.
- While making the tag pages, note that you pass `tag.name` through in the `context`. This is the value that gets used in the `TagPage` query to limit your search to only posts tagged with the tag in the URL.

Make a tags index page (/tags) that renders a list of all tags

Your `/tags` page will list out all tags, followed by the number of posts with that tag. You can get the data with the first query you wrote earlier, that groups posts by tags:

```
import React from "react"
import PropTypes from "prop-types"

// Utilities
import kebabCase from "lodash/kebabCase"

// Components
import { Helmet } from "react-helmet"
import { Link, graphql } from "gatsby"

const TagsPage = ({
  data: {
    allMarkdownRemark: { group },
    site: {
      siteMetadata: { title },
    },
  },
}) => (
  <div>
    <Helmet title={title} />
    <div>
      <h1>Tags</h1>
      <ul>
        {group.map(tag => (
          <li key={tag.fieldValue}>
            <Link to={`/tags/${kebabCase(tag.fieldValue)}/`}>
              {tag.fieldValue} ({tag.totalCount})
            </Link>
```

```
          </li>
        ))}
      </ul>
    </div>
  </div>
)

TagsPage.propTypes = {
  data: PropTypes.shape({
    allMarkdownRemark: PropTypes.shape({
      group: PropTypes.arrayOf(
        PropTypes.shape({
          fieldValue: PropTypes.string.isRequired,
          totalCount: PropTypes.number.isRequired,
        }).isRequired
      ),
    }),
    site: PropTypes.shape({
      siteMetadata: PropTypes.shape({
        title: PropTypes.string.isRequired,
      }),
    }),
  }),
}

export default TagsPage

export const pageQuery = graphql`
  query {
    site {
      siteMetadata {
        title
      }
    }
    allMarkdownRemark(limit: 2000) {
      group(field: frontmatter___tags) {
        fieldValue
        totalCount
      }
    }
  }
`
```

(optional) Render tags inline with your blog posts

The home stretch! Anywhere else you'd like to render your tags, add them to the `frontmatter` section of your `graphql` query and access them in your component like any other prop.