Using Prismic with GraphQL Source Plugin

Prismic + Gatsby features

Prismic is a headless CMS that provides a web application that allows content writers to focus on creating and updating the content of their website. Essentially, it differentiates itself thanks to its UX that allows a lot of flexibility without complicating it. The Slices feature plays a big role in this, they essentially correspond to components in the web application that non-technical people can use to assemble their page. Used alongside Gatsby it lets you create fast, editable sites.

Note that the Gatsby Prismic source plugin you will use during this tutorial, as a main feature, allows for **real previews** of your pages so you can know how they will look like before publishing or building them. You can generate **shareable links** that can even be seen by non-Prismic users.

What's contained in this tutorial?

By the end of this tutorial, you'll have done the following:

- Learned how to configure the latest version of the Prismic source plugin
- Fetch and render data from a Prismic repository
- Create pages programmatically from a Prismic source
- Configure previews for unpublished and edited documents

Prerequisites

- Familiarity with Gatsby
- A personal Prismic repository. You can follow this 5 minute guide for instructions to set up a barebones repository for a minimalistic blog

Preparing your environment

Start a new Gatsby project using the default starter.

```
gatsby new gatsby-prismic-blog
cd gatsby-prismic-blog
```

Now, you will have to install missing packages. This command includes the Gatsby Prismic source plugin, and the Prismic kits for facilitating processes in React:

```
npm install gatsby-source-prismic-graphql prismic-javascript prismic-reactjs
```

If you want to focus on learning how to fetch and render data from a Prismic repository, it's safe to just use the sample repository `gatsby-blog-scratch` but you should create your own so you can modify the content and test out previews. Otherwise, you can follow this article to create your own Prismic repository. You will want to write several blog posts to have some nice content to show off.

Configuring the plugin

You will set up the Gatsby source plugin so it can fetch data from the Prismic repository. For this the only option you have to specify is the repository's name.

```
{
  resolve: `gatsby-source-prismic-graphql`,
  options: {
    repositoryName: 'gatsby-blog-scratch'
  }
},
```

If you run `gatsby develop` now, you should be able to access the data from Prismic through the GraphiQL interface in `http://localhost:8000/___graphql`. You can check this by trying a query that uses the Prismic source.

```
{
  prismic {
    _allDocuments {
      edges {
        node {
          __typename
        }
      }
    }
  }
}
```

If your Prismic documents are being retrieved, then the configuration was successful!

Fetching and rendering data

Once you have verified on GraphiQL that you can retrieve the data from your previously created Prismic content repository, you can start on fetching and rendering that content.

In GraphiQL, experiment with the data and how it's structured. You can use the autocomplete to help you navigate around how Gatsby interprets the Prismic repository. You will need a query that gets the Home information, as well as an array of blog posts sorted in descending order.

```
import React from "react"
import { Link, graphql } from "gatsby" //highlight-line

import Layout from "../components/layout"
import { RichText } from "prismic-reactjs" //highlight-line

//highlight-start
export const query = graphql`
  {
    prismic {
      allBlog_homes {
        edges {
          node {
            headline
            description
            image
          }
        }
      }
      allPosts(sortBy: date_DESC) {
        edges {
          node {
            _meta {
              id
              uid
              type
            }
            title
            date
          }
        }
      }
    }
  }
`
// highlight-end
```

In order to render this data, replace the `export default IndexPage` line in the original `index.js` file with the following:

```
export default function Home({ data }) {
  const doc = data.prismic.allBlog_homes.edges.slice(0, 1).pop()
```

```
  const posts = data.prismic.allPosts.edges

  if (!doc) return null

  return (
    <Layout>
      <div>
        <img src={doc.node.image.url} alt={doc.node.image.alt} /> // Make sure
        to add an accessible alt attribute when adding images in Prismic:
        https://user-guides.prismic.io/articles/768849-add-metadata-to-an-asset
        <h1>{RichText.asText(doc.node.headline)}</h1>
        <p>{RichText.asText(doc.node.description)}</p>
      </div>
    </Layout>
  )
}
```

Save the file and check on your site running at `http://localhost:8000`

You can use the helper function `RichText` to render formatted text and generally, this is the process you will use to query your Prismic repository and then render it. You can clean this up and include a function that will render the array of queried blog posts.

```
//highlight-start
const BlogPosts = ({ posts }) => {
  if (!posts) return null
  return (
    <div>
      {posts.map(post => {
        return (
          <div key={post.node._meta.id}>
            <h2>{RichText.asText(post.node.title)}</h2>
            <p>
              <time>{post.node.date}</time>
            </p>
          </div>
        )
      })}
    </div>
  )
}
//highlight-end

export default function Home({ data }) {
  const doc = data.prismic.allBlog_homes.edges.slice(0, 1).pop()
  const posts = data.prismic.allPosts.edges
```

```
  if (!doc) return null

  return (
    <Layout>
      <div>
        <img src={doc.node.image.url} alt={doc.node.image.alt} />
        <h1>{RichText.asText(doc.node.headline)}</h1>
        <p>{RichText.asText(doc.node.description)}</p>
      </div>
      <BlogPosts posts={posts} /> //highlight-line
    </Layout>
  )
}
```

Building links to your documents

Now things are really taking shape. You will turn these blog post titles into links by building a link resolver function, which will build the correct route for your posts.

```
exports.linkResolver = function linkResolver(doc) {
  // Route for blog posts
  if (doc.type === "post") {
    return "/blog/" + doc.uid
  }
  // Homepage route fallback
  return "/"
}
```

This function's goal is to generate a navigation url that depends on the document type. In this case, it will be `/blog/{unique slug}` for documents of the type `post` and the root folder `/` for every other document type, including `blog_home`. As you can see, the data required to generate a proper url with this helper function is the document's type and unique identifier. These fields are always present as part of the `_meta` field, so make sure to retrieve it in your query.

You will have to set up your `gatsby-browser.js` file to use the Link Resolver as well.

```
const { registerLinkResolver } = require("gatsby-source-prismic-graphql")
const { linkResolver } = require("./src/utils/linkResolver")

registerLinkResolver(linkResolver)
```

You can now use the proper url generated by the `linkResolver` function to build the `<Link>` component for each blog post, so that post titles are links now.

```
const BlogPosts = ({ posts }) => {
```

```
  if (!posts) return null
  return (
    <ul>
      {posts.map(post => {
        return (
          <li key={post.node._meta.id}>
            // highlight-start
            <Link to={linkResolver(post.node._meta)}>
              {RichText.asText(post.node.title)}
            </Link>
            // highlight-end
            <p>
              <time>{post.node.date}</time>
            </p>
          </li>
        )
      })}
    </ul>
  )
}
```

At this point, if you try to use these links to navigate your site, you will notice that you're faced with a 404 error on every page. That's because even though you're navigating to the pages correctly, there's still the issue of actually creating the pages. For this, you will have to use a template file which will programmatically build a page for each blog post.

Creating pages programmatically with a template

The template you will build is very similar in concept to a regular page, with the addition of introducing a variable which will be used to query the different blog posts.

```
import React from "react"
import { graphql, Link } from "gatsby"
import { RichText } from "prismic-reactjs"

export const query = graphql`
  query BlogPostQuery($uid: String) {
    prismic {
      allPosts(uid: $uid) {
        edges {
          node {
            title
            date
            post_body
          }
```

```
        }
      }
    }
  }
`

export default function Post({ data }) {
  const doc = data.prismic.allPosts.edges.slice(0, 1).pop()
  if (!doc) return null

  return (
    <div>
      <Link to="/">
        <span>go back</span>
      </Link>
      <h1>{RichText.asText(doc.node.title)}</h1>
      <span>
        <em>{doc.node.date}</em>
      </span>
      <div>{RichText.render(doc.node.post_body)}</div>
    </div>
  )
}
```

You're rendering a very minimalistic page for each blog post; consisting of:

- A link back to the homepage.
- The article's title and page.
- The rich text body which will contain formatted text and images.

Having a template is not enough to generate the dynamic pages, you will need to configure the plugin in `gatsby-config.js` so that pages are created following the same pattern defined in the link resolver function.

```
options: {
  repositoryName: 'gatsby-blog-scratch',
  //highlight-start
  pages: [{
    type: 'Post',          // Custom type of the document
    match: '/blog/:uid',   // Pages will be generated in this pattern
    path: '/blog-preview', // Placeholder route for previews
    component: require.resolve('./src/templates/post.js') // Template file
  }]
  //highlight-end
}
```

And with this last step you should be able to see all of your blog posts rendered on your site.

Setting up for Previews

One of the most exciting features that this Gatsby Prismic source plugin provides is the ability to preview changes to your documents without having to publish them or rebuild your Gatsby app. To activate this, you first need to set up an endpoint in your Prismic repository.

In your repository, go to **Settings > Previews > Create a New Preview** and fill in the fields for your setup. For a default local development environment, you should use `http://localhost:8000` as the Domain, with `/preview` as the optional Link Resolver. Don't worry about including the toolbar script, the plugin will take care of it.

Finally, return to your Gatsby configuration file to activate the feature.

```
{
  resolve: `gatsby-source-prismic-graphql`,
  options: {
    repositoryName: 'gatsby-blog-scratch',
    //highlight-start
    previews: true,
    path: '/preview',
    //highlight-end
    pages: [{
      type: 'Post',
      match: '/blog/:uid',
      path: '/blog-preview',
      component: require.resolve('./src/templates/post.js')
    }]
  }
}
```

Your blog is ready to handle previews now. Just edit any of your blog posts in your Prismic repository and preview the changes you've made instead of publishing it. Previews are not limited to the development environment, you can adjust the endpoint configuration to work for the hosted version you deploy. Content writers can use it as a staging environment to preview their content changes without having to rebuild your site, and before publishing.

What did you just build?

After following this tutorial you have a minimalist blog that uses a Prismic repository as a data source. You have learned:

- How to query the relevant data with GraphQL.
- Render that data in your site.
- Create templates to generate pages programmatically.
- Build links to navigate your site.

- And most importantly, how to set up previews so you can view changes to your documents without publishing them.

Where to go from here

If you want to go further, here are some more advanced things you can do using Prismic:

- Rendering slice components for modular page building.
- Using webhooks as a trigger to rebuild your site.
- Using a helper function to change links inside rich text fields to Link components.

You can read more about it in Prismic's Gatsby documentation, build your own full-featured blog where you can try out Slices.