Headless the GraphCMS way

GraphCMS is a GraphQL native Headless Content Management System (Headless CMS) that lets you programmatically deliver content across platforms. With features like an intuitive schema builder, GraphQL mutations API, and out of the box localization, GraphCMS enables you to rapidly build digital projects with your preferred frameworks and languages.

Getting started

In this guide you'll create a Gatsby site capable of querying data from GraphCMS.

Prerequisites

This guide assumes the following:

- You have an active GraphCMS account
- You've created a new GraphCMS project (preferably using the `Blog Starter` template)

All schema and data references in this guide are from the GraphCMS `Blog Starter` template.

Create a new Gatsby site

To begin, let's create a new Gatsby site using the default starter.

```
gatsby new gatsby-site https://github.com/gatsbyjs/gatsby-starter-default
```

Once finished, navigate inside of the project with `cd gatsby-site`.

Add the `gatsby-source-graphcms` plugin

In order to fetch data from your GraphCMS project, you will need to install `gatsby-source-graphcms`.

You can install this package with:

```
npm install gatsby-source-graphcms
```

Configure the plugin

The last step required before you can query your data is to configure `gatsby-source-graphcms`. Inside of `gatsby-config.js`, add a new plugin configuration.

> *We recommend using environment variables with your GraphCMS `token` and `endpoint` values. You can learn more about using environment variables with Gatsby in the How-To Guide about environment variables.*

```
{
  resolve: 'gatsby-source-graphcms',
    options: {
      // Your GraphCMS API endpoint. Available from your project settings.
      endpoint: process.env.GRAPHCMS_ENDPOINT
      // A PAT (Permanent Auth Token) for your project. Required if your project is
not available publicly, or you want to scope access to a specific content stage
(i.e. draft content).
```

```
      token: process.env.GRAPHCMS_TOKEN
    },
},
```

Inspecting the schema

Start the Gatsby development environment with `gatsby develop`. Once running, you will be able to access the GraphiQL explorer in your browser:

`http://localhost:8000/___graphql`

From here, you will be able to browse the generated GraphQL schema of your Gatsby project.

 GraphCMS Schema

The generated Gatsby types for the GraphCMS project models will all be prefixed accordingly. For example, the `Post` model from our GraphCMS project becomes a `GraphCMS_Post` type inside of Gatsby. This prefix can be configured.

Query fields also follow a similar naming convention and are prefixed by the same value as types. Gatsby will generate two root query fields per type. For example, for our `Post` model Gatsby will generate the following root query fields:

- `allGraphCmsPost` for querying multiple `Post` entries
- `graphCmsPost` for querying a single `Post` entry

Querying for content

 GraphCMS Query

Using the generated schema, we can begin to write GraphQL queries for Gatsby data. Consider the query below, which will return a full list of all available `GraphCMS_Post` nodes.

```
{
  allGraphCmsPost {
    nodes {
      id
      content {
        markdown
      }
      coverImage {
        url
      }
      date
      slug
      title
    }
  }
}
```

```
{
  "data": {
    "allGraphCmsPost": {
      "nodes": [
        {
          "id": "Post:ckadrcx4g00pw01525c5d2e56",
          "content": {
            "markdown": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quid ergo? Huius ego nunc auctoritatem sequens idem faciam. Duo Reges: constructio interrete. Sed in rebus apertissimis nimium longi sumus. Itaque his sapiens semper vacabit. Non semper, inquam;\n\n\n\nVerum hoc idem saepe faciamus. Quamquam haec quidem praeposita recte et reiecta dicere licebit.\n\n\n\nAt coluit ipse amicitias. Certe non potest. Bonum incolumis acies: misera caecitas. Quo studio Aristophanem putamus aetatem in litteris duxisse? Idem iste, inquam, de voluptate quid sentit? Facillimum id quidem est, inquam.\n"
          },
          "coverImage": {
            "url": "https://media.graphcms.com/QEg7oQCTEeEjLSEPQJtg"
          },
          "date": "2020-05-05",
          "slug": "technical-seo-with-graphcms",
          "title": "Technical SEO with GraphCMS"
        },
        {
          "id": "Post:ckadrfuu000pe0148kels2b5e",
          "content": {
            "markdown": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Id enim natura desiderat. Falli igitur possumus. Negat enim summo bono afferre incrementum diem. Indicant pueri, in quibus ut in speculis natura cernitur.\n\n\n\n# Lorem Ipsum\n\n\n\nNe amores quidem sanctos a sapiente alienos esse arbitrantur. Summus dolor plures dies manere non potest? Expectoque quid ad id, quod quaerebam, respondeas. Non est ista, inquam, Piso, magna dissensio. Respondeat totidem verbis. Non est igitur summum malum dolor.\n\n\n\nHic ambiguo ludimur. Nam Pyrrho, Aristo, Erillus iam diu abiecti. Si longus, levis dictata sunt. Duo Reges: constructio interrete. Deinde dolorem quem maximum?\n"
          },
          "coverImage": {
            "url": "https://media.graphcms.com/gzYJIkMRRHCq0JLDOqgU"
          },
          "date": "2020-05-01",
          "slug": "union-types-and-sortable-relations",
          "title": "Union Types and Sortable Relations with GraphCMS"
        }

        // ...more results
      ]
    }
  }
}
```

Query implementation

Gatsby offers two means of data querying: page queries and static queries.

Page query

Page queries run at build time and can accept GraphQL variables via page context. As the name suggest, they can only be used on pages and **not** on non-page components.

The resulting data is available via the page props `data` key.

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

function IndexPage({ data: { posts } }) {
  return (
    <ul>
      {posts.nodes.map(post => (
        <li key={post.id}>
          <h3>{post.title}</h3>
        </li>
      ))}
    </ul>
  )
}

export const pageQuery = graphql`
  query IndexPageQuery {
    posts: allGraphCmsPost {
      nodes {
        id
        content {
          markdown
        }
        coverImage {
          url
        }
        date
        slug
        title
      }
    }
  }
`

export default IndexPage
```

Static query hook

Static queries also run at build time, but can be used in all components via the `useStaticQuery` hook or `<StaticQuery />` component. However they do **not** accept GraphQL variables.

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

function IndexPage() {
  const { posts } = useStaticQuery(graphql`
    {
      posts: allGraphCmsPost {
        nodes {
          id
          content {
            markdown
          }
          coverImage {
            url
          }
          date
          slug
          title
        }
      }
    }
  `)

  return (
    <ul>
      {posts.nodes.map(post => (
        <li key={post.id}>
          <h3>{post.title}</h3>
        </li>
      ))}
    </ul>
  )
}

export default IndexPage
```

Learn more

For additional examples of how to query and use GraphCMS data in the context of Gatsby, check out the following references:

- `gatsby-source-graphcms`
- `gatsby-starter-graphcms-blog`
- `graphcms-examples`

Summary

With projects of all sizes transitioning to an API-first approach, creating, enriching, and delivering your content via API to your Gatsby projects is a breeze. Adding a content API in the backend with GraphCMS provides a scalable CMS that you can start using within minutes, and have your project up and running quickly.

Try GraphCMS by getting started with the free-forever community plan