This guide takes you through the steps involved in setting up a Gatsby site that fetches content from Agility CMS.

Check our docs for up-to-date documentation.

What is Agility CMS?

Agility CMS is a flexible headless Content Management System (CMS) with an infinite number of ways to configure and set up your pages and content. Agility lets you define custom content types, relationships and pages. This is called Content Architecture, and you can reuse this content for your websites and apps.

Agility believes that a successful website balances the **User Experience (UX)**, **Editor Experience (EX)**, and **Developer Experience (DX)**.

While Gatsby tends to handle **UX** and **DX** quite well, editors are not always comfortable working directly in a codebase. They prefer to manage their sitemap and see what content is on which pages. Using Gatsby with a headless CMS like Agility allows for this.

Agility aims to empower and improve the **Editor Experience** by providing built-in **Page Management**. This means developers can build UI Components and leave editors to compose their pages.

Learn more about Agility CMS and Gatsby

Getting Started

This guide walks you through the steps involved in setting up a Gatsby site that fetches content from Agility CMS.

Step 1: Create a free Agility account

In order to showcase best practices, evaluate, and help onboard new users when using Agility CMS, this guide uses the pre-configured Blog Template. This template has a few Pages along with some Page Modules and Content Models already set for you to help support a familiar blogging experience.

It's important to note though, that you can customize anything and everything in this template, it's just giving you a head start and serving as an example!

Create an Agility CMS account with the **Free Developer Plan** (this plan is free forever).

Step 2: Get the code

Make sure you have the Gatsby CLI installed:

```
npm install -g gatsby-cli
```

Create a new Gatsby project using the Agility CMS Gatsby Starter repo from GitHub that has all the code you need to get started:

```
gatsby new agilitycms-gatsby-starter https://github.com/agility/agilitycms-gatsby-starter
```

Install the dependencies:

```
npm install
```

Step 3: Authenticate your Gatsby site with Agility

Make a copy of `.env.development.example` file called `.env.development`.

```
cp .env.development.example .env.development
```

Add your `AGILITY_GUID` and `AGILITY_API_KEY` variable values to the `.env.development` file, you can find these in the Content Manager by going to Settings.

```
# Your Instance Id
AGILITY_GUID=

# Your Preview API Key (recommended) - you can get this from the Getting Started
Page in Agility CMS. It starts with defaultpreview.
AGILITY_API_KEY=

# If using your Preview API Key, set this to true
AGILITY_API_ISPREVIEW=true
```

Step 4: Running the site locally

Once your environment variables have been added for development, you can run the site locally:

```
gatsby develop
```

If successful, your site's build should complete and you be able to view the site in your browser on `http://localhost:8000`. You can also run the GraphiQL IDE at `http://localhost:8000/___graphql`. The GraphiQL IDE will help you explore the app's data, including data from Agility.

For production, follow the same steps using the `.env.production` file and your Live API Key.

How it works

Sourcing and Querying Agility Content in Gatsby

You can source content from Agility CMS into your Gatsby site with `gatsby-source-agilitycms` which will synchronize all of your content, page modules, sitemaps and pages, and make them available via GraphQL.

The source plugin uses `@agility/content-sync` which ensures that only content that has changed is updated. It keeps things in-sync for you. This means your first build will download all of your content, while subsequent builds only update what has changed. This enables incremental builds and results in bleeding-edge, fast build times.

To query content from Agility CMS, you would query `allAgility<contentTypeName>`, for example, `allAgilityPost`, then select what fields you want to retrieve for each item. An example can be seen here in the agilitycms-gatsby-starter:

```
query {
  posts: allAgilityPost {
    nodes {
      customFields {
```

```
        title
        date(formatString: "MMMM DD, YYYY")
        image {
          url
          label
        }
        content
      }
    }
  }
}
```

Learn more about Sourcing and Querying Agility Content in Gatsby

How Pages Work

The `gatsby-source-agilitycms` plugin makes it easy to source content, but it also generates your Pages for you based off of your sitemap in Agility CMS. This means that editors in the CMS control what pages are available, what their URLs are, and exactly what UI components (Agility CMS calls these Page Modules) make up each page.

Each page in Agility is composed and generated dynamically at *build-time* using a **masterPageTemplate** that you define in your `gatsby-config.js` plugin options.

```
module.exports = {
  siteMetadata: {
    title: "Agility CMS Gatsby Starter",
  },
  plugins: [
    ...
    {
      //the name of the plugin
      resolve: "@agility/gatsby-source-agilitycms",
      //the options for our plugin
      options: {
        ...
         //the page template that will be used to render Agility CMS pages
        masterPageTemplate: "./src/AgilityPage.jsx"
      },
    },
    ...
  ]
}
```

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

export const query = graphql`
  query($pageID: Int!, $contentID: Int!, $languageCode: String!) {
    agilitypage(languageCode: { eq: $languageCode }, itemID: { eq: $pageID }) {
      pageJson
    }
```

```
    agilityitem(
      languageCode: { eq: $languageCode }
      itemID: { eq: $contentID }
    ) {
      itemJson
    }
  }
`

const AgilityPage = ({ pageContext, data }) => {
  return (
    <>
      <SEO
        title={viewModel.page.title}
        description={viewModel.page.seo.metaDescription}
        keywords={viewModel.page.seo.metaKeywords}
        ogImage={viewModel.dynamicPageItem?.customFields?.image?.url}
      />
      <PreviewBar isPreview={viewModel.isPreview} />
      <div id="site-wrapper" className="flex flex-col min-h-screen">
        <SiteHeader
          languageCode={viewModel.languageCode}
          isMultiLanguage={viewModel.isMultiLanguage}
        />
        <main className="flex-grow">
          <AgilityPageTemplate {...viewModel} />
        </main>
        <SiteFooter />
      </div>
    </>
  )
}

export default AgilityPage
```

Learn more about How Agility Pages Work in Gatsby

How Page Modules Work

Page Modules in Agility CMS are the functional components that make up a page. Editors use these to compose what type of content is on each page and in what order they appear. Developers define what page modules are available in the CMS and what fields they have. Each module defined in Agility CMS should correspond to a React Component in your Gatsby site.

In the agilitycms-gatsby-starter site, the name of the page module is used to find a corresponding React component that matches the same name. If a match is found, that component is dynamically imported and rendered.

For example, if a module has a reference name of RichTextArea in the CMS, then while the page is being rendered by the gatsby-source-agilitycms plugin, it will look for `src/components/agility-pageModules/RichTextArea.jsx` in the `src/components/agility-pageModules` directory.

```
import React from "react"
import { renderHTML } from "../../agility/utils"

const RichTextArea = ({ module }) => {
  // get module fields
  const { customFields } = module

  return (
    <div className="relative px-8">
      <div className="max-w-2xl mx-auto my-12 md:mt-18 lg:mt-20">
        <div
          className="prose max-w-full mx-auto"
          dangerouslySetInnerHTML={renderHTML(customFields.textblob)}
        />
      </div>
    </div>
  )
}

export default RichTextArea
```

Learn more about How Agility Page Modules Work in Gatsby

How Page Templates Work

Page Templates in Agility CMS are how developers can differentiate the styles of certain types of pages, and define where on the page that editors can add Modules (functional components of the page that editors control). Some sites may only have a single page template and this is re-used across the site, others may have other templates to allow for more flexible layouts.

In the agilitycms-gatsby-starter site, the Name of the Page Template is used to find a corresponding React component that matches the same name. If a match is found, that component is dynamically imported and rendered.

For example, if a Page Template has a reference name of MainTemplate in the CMS, then while the page is being rendered by the `gatsby-source-agilitycms` plugin, it will look for `src/components/agility-pageTemplates/MainTemplate.jsx` in the `src/components/agility-pageTemplates` directory.

```
import React from "react"
import ContentZone from "../../agility/components/ContentZone"
import { getModule } from "../../components/agility-pageModules"

const MainTemplate = props => {
  return (
    <div id="main-template">
      <ContentZone name="MainContentZone" {...props} getModule={getModule} />
    </div>
  )
}
export default MainTemplate
```

Learn more about How Agility Page Templates Work in Gatsby

Resolving Linked Content

While you are sourcing and querying content in gatsby, you are likely to come across a scenario where you need to retrieve a content item and its related content. In Agility CMS, Linked Content fields are used to related content to one another in various ways.

When querying a Post, for example, you may want to also retrieve the details for the Category. On your Post GraphQL node, you may notice a category property, however, it will only contain a contentID reference, not the entire node representing the Category. You'll need to resolve this Linked Content when you need it.

In the agilitycms-gatsby-starter, the Linked Content is resolved by using Gatsby Resolvers.

Resolvers are added to your `gatsby-node.js` in your site, and they allow you to add a new field to your content node which will handle resolving your Linked Content reference. This means you are telling GraphQL, when you query a specific property on a node, it will actually run a function to go and get your Linked Content and return it.

An example of this can be found here in the starter site:

```
const agility = require("./src/agility/utils")

exports.createResolvers = args => {
  const {
    createResolvers,
    getNode,
    createNodeId,
    createNode,
    createContentDigest,
    configOptions,
  } = args

  // The data needed for Linked Content is resolved here
  const resolvers = {
    // on the 'agilityPost' node type
    agilityPost: {
      // get the sitemap node that represents this item ( i.e. /blog/my-blog-post )
      sitemapNode: agility.getDynamicPageItemSitemapNode(),

      // get the category
      linkedContent_agilityCategory: agility.getLinkedContentItem({
        type: "agilityCategory",
        linkedContentFieldName: "category",
      }),
    },
  }
  createResolvers(resolvers)
}
```

Now on the `agilityPost` node, you'll be able to query the category:

```
query {
  posts: allAgilityPost {
    nodes {
```

```
      customFields {
        title
        date(formatString: "MMMM DD, YYYY")
        image {
          url
          label
        }
        content
      }
      linkedContent_agilityCategory {
        customFields {
          title
        }
      }
    }
  }
}
```

How Images Work

Agility CMS also provides `gatsby-image-agilitycms` as a npm package. This is a custom image component that takes images stored within Agility CMS and handles all of the hard parts of displaying responsive images that follow best practices for performance on your website or application.

```
import { AgilityImage } from "@agility/gatsby-image-agilitycms"

const Component = ({ image }) => (
  <AgilityImage image={image} layout="fullWidth" />
)
```

Learn more about How Images Work in Gatsby

Resources

- Agility CMS Official Site
- Agility CMS & Gatsby Documentation
- Agility CMS Community Slack