Creating a new site from scratch

What this tutorial covers:

In this tutorial, you will install the `gatsby-source-wordpress` plugin in order to pull blog and image data from a WordPress install into your Gatsby site and render that data. This Gatsby + WordPress starter shows you the source code for an example site similar to what you're going to be building in this tutorial.

Creating a site with the `gatsby-source-wordpress` plugin

Create a new Gatsby project and change directories into the new project you just created:

```
gatsby new wordpress-tutorial-site
cd wordpress-tutorial-site
```

Install the `gatsby-source-wordpress` plugin. For extra reading on the plugin's features and examples of GraphQL queries not included in this tutorial, see the `gatsby-source-wordpress` plugin's README file.

```
npm install gatsby-source-wordpress
```

Add the `gatsby-source-wordpress` plugin to `gatsby-config.js` using the following code, which you can also find in this starter's source code.

```
module.exports = {
  siteMetadata: {
    title: `Gatsby WordPress Tutorial`,
    description: `An example to learn how to source data from WordPress.`,
    author: `@gatsbyjs`,
  },
  plugins: [
    /*
     * Gatsby's data processing layer begins with "source"
     * plugins. Here the site sources its data from WordPress.
     */
    // highlight-start
    {
      resolve: `gatsby-source-wordpress`,
      options: {
        /*
         * The full URL of the WordPress site's GraphQL API.
         * Example : 'https://www.example-site.com/graphql'
         */
        url: `https://wpgatsbydemo.wpengine.com/graphql`,
      },
    },
    // highlight-end
```

```
    /**
     * The following plugins aren't required for gatsby-source-wordpress,
     * but we need them so the default starter we installed above will keep working.
     **/
    `gatsby-plugin-react-helmet`,
    {
      resolve: `gatsby-source-filesystem`,
      options: {
        name: `images`,
        path: `${__dirname}/src/images`,
      },
    },
    `gatsby-transformer-sharp`,
    `gatsby-plugin-sharp`,
    {
      resolve: `gatsby-plugin-manifest`,
      options: {
        name: `gatsby-starter-default`,
        short_name: `starter`,
        start_url: `/`,
        background_color: `#663399`,
        theme_color: `#663399`,
        display: `minimal-ui`,
        icon: `src/images/gatsby-icon.png`, // This path is relative to the root of the site
      },
    },
  ],
}
```

Creating GraphQL queries that pull data from WordPress

Now you are ready to create a GraphQL query to pull in some data from the WordPress site. You will create a query that pulls in the title of the blog posts, date they were posted, and blogpost content.

Run:

```
gatsby develop
```

In your browser, open `http://localhost:8000` to see your site, and `http://localhost:8000/___graphql` to see Graphiql. You can use Graphiql to create your GraphQL queries.

As an exercise, try re-creating the following queries in your GraphiQL explorer.

Tip: If you've never used Graphiql before, try pressing `shift+space` to be given a list of available fields you can query for. Press up and down on your keyboard to select a field and press enter to write it to your query. Alternatively you can

use the explorer pane to the left side of the page. If your explorer pane isn't open, you can open it by clicking "Explorer" at the top of the screen.

This first query will pull in the blogpost content from WordPress:

```
query {
  allWpPost {
    nodes {
      id
      title
      excerpt
      slug
      date(formatString: "MMMM DD, YYYY")
    }
  }
}
```

This next query will pull in a sorted list of the blog posts:

```
{
  allWpPost(sort: { fields: [date] }) {
    nodes {
      title
      excerpt
      slug
    }
  }
}
```

Rendering the blog posts to `index.js`

Now that you've created GraphQL queries that pull in the data you want, you'll use that second query to create a list of sorted blogpost titles on your site's homepage. Here's what your home page component in `src/pages/index.js` should look like:

```
import React from "react"
import { graphql } from "gatsby"
import Layout from "../components/layout"
import SEO from "../components/seo"

export default function Home({ data }) {
  //highlight-line
  return (
    <Layout>
      <SEO title="home" />
      {/* highlight-start */}
      <h1>My WordPress Blog</h1>
```

```
      <h4>Posts</h4>
      {data.allWpPost.nodes.map(node => (
        <div>
          <p>{node.title}</p>
          <div dangerouslySetInnerHTML={{ __html: node.excerpt }} />
        </div>
      ))}
      {/* highlight-end */}
    </Layout>
  )
}

//highlight-start
export const pageQuery = graphql`
  query {
    allWpPost(sort: { fields: [date] }) {
      nodes {
        title
        excerpt
        slug
      }
    }
  }
`
//highlight-end
```

Save these changes and look at `http://localhost:8000` to see your new homepage with a list of sorted blog posts!

Creating pages for each blog post and linking to them

An index page with a post title and excerpt is great, but you should also build pages out for each of the blog posts, and link to them from your `index.js` file.

To do this, you need to:

1. Create pages for each blog post
2. Link up the title on the index page with the post page.

If you haven't already, please read through Part 7 of the foundational tutorial, as it goes through the concept and examples of this process with Markdown instead of WordPress.

Creating pages for each blog post

In Part 7 of the tutorial, the first step in creating pages is creating slugs for the markdown files. Since you are using WordPress and not Markdown files, you can

grab the slugs that get returned from your API call to the WordPress source. You can skip creating slugs, since you already have them.

Open up your `gatsby-node.js` file in the root of your project (it should be blank except for some comments) and add the following:

```
const path = require(`path`)

exports.createPages = ({ graphql, actions }) => {
  const { createPage } = actions
  return graphql(`
    {
      allWpPost(sort: { fields: [date] }) {
        nodes {
          title
          excerpt
          content
          slug
        }
      }
    }
  `).then(result => {
    console.log(JSON.stringify(result, null, 4))
    process.exit()
  })
}
```

Next, stop and restart the `gatsby develop` environment. As you watch the terminal you should see two Post objects log to the terminal.

Excellent! As explained in Part 7 of the tutorial, this `createPages` export is one of the Gatsby "workhorses" and allows us to create your blog posts (or pages, or custom post types, etc.) from your WordPress install.

Before you can create the blog posts, however, you need to specify a template to build the pages.

In your `src` directory, create a directory called `templates` and in the newly created `templates` folder, create a filed named `blog-post.js`. In that new file, paste the following:

```
import React from "react"
import Layout from "../components/layout"
import { graphql } from "gatsby"

export default function BlogPost({ data }) {
  const post = data.allWpPost.nodes[0]
  console.log(post)
  return (
```

```
    <Layout>
      <div>
        <h1>{post.title}</h1>
        <div dangerouslySetInnerHTML={{ __html: post.content }} />
      </div>
    </Layout>
  )
}
export const query = graphql`
  query($slug: String!) {
    allWpPost(filter: { slug: { eq: $slug } }) {
      nodes {
        title
        content
      }
    }
  }
`
```

What is this file doing? After importing your dependencies, it constructs the layout of the post with JSX. It wraps everything in the `Layout` component, so the style is the same throughout the site. Then, it simply adds the post title and the post content. You can add anything you want and can query for here (e.g. feature image, post meta, custom fields, etc.).

Below that, you can see the GraphQL query calling the specific post based on the `$slug`. This variable is passed to the `blog-post.js` template when the page is created in `gatsby-node.js`. To accomplish this, add the following code to the `gatsby-node.js` file:

```
const path = require(`path`)

exports.createPages = ({ graphql, actions }) => {
  const { createPage } = actions
  return graphql(`
    {
      allWpPost(sort: { fields: [date] }) {
        nodes {
          title
          excerpt
          content
          slug
        }
      }
    }
  `).then(result => {
    //highlight-start
```

```
    result.data.allWpPost.nodes.forEach(node => {
      createPage({
        path: node.slug,
        component: path.resolve(`./src/templates/blog-post.js`),
        context: {
          // This is the $slug variable
          // passed to blog-post.js
          slug: node.slug,
        },
      })
    })
    //highlight-end
  })
}
```

You will need to stop and start your environment again using `gatsby develop`. When you do, you will not see a change on the index page of the site, but if you navigate to a 404 page, like `http://localhost:8000/asdf`, you should see the two sample posts created and be able to click on them to go to the sample posts.

But nobody likes to go to a 404 page to find a blog post! So, let's link these up from the home page.

Linking to posts from the homepage

Since you already have your structure and query done for the `index.js` page, all you need to do is use the `Link` component to wrap your titles and you should be good to go.

Open up `src/pages/index.js` again and add the following:

```
import React from "react"
import { Link, graphql } from "gatsby" //highlight-line
import Layout from "../components/layout"
import SEO from "../components/seo"

export default function Home({ data }) {
  return (
    <Layout>
      <SEO title="home" />
      <h1>My WordPress Blog</h1>
      <h4>Posts</h4>
      {data.allWpPost.nodes.map(node => (
        <div key={node.slug}>
          {/* highlight-start */}
          <Link to={node.slug}>
            <p>{node.title}</p>
          </Link>
```

```
          {/* highlight-end */}
          <div dangerouslySetInnerHTML={{ __html: node.excerpt }} />
        </div>
      ))}
    </Layout>
  )
}

export const pageQuery = graphql`
  query {
    allWpPost(sort: { fields: [date] }) {
      nodes {
        title
        excerpt
        slug
      }
    }
  }
`
```

And that's it! When you wrap the title in the `Link` component and reference the slug of the post, Gatsby will add some magic to the link, preload it, and make the transition between pages incredibly fast

Wrapping up

You can apply the same procedure to calling and creating pages, custom post types, custom fields, taxonomies, and all the fun and flexible content WordPress is known for. This can be as simple or as complex as you would like it to be, so explore and have fun with it!

Up Next :point_right:

- :boat: Migrating from other WP source plugins
- :house: Hosting WordPress
- :athletic_shoe: Themes, Starters, and Examples
- :medal_sports: Usage with popular WPGraphQL extensions
- :hammer_and_wrench: Debugging and troubleshooting
- :national_park: Community and Support
- :point_left: Back to README.md