Most examples in the Gatsby docs and on the web at large focus on leveraging source plugins to manage your data in Gatsby sites. However, source plugins (or even Gatsby nodes) aren't strictly necessary to pull data into a Gatsby site! It's also possible to use an "unstructured data" approach in Gatsby sites, no GraphQL required.

> *Note: For our purposes here, "unstructured data" means data "handled outside of Gatsby's data layer" (we're using the data directly, and not transforming the data into Gatsby nodes).*

The approach: fetch data and use Gatsby's `createPages` API

> *Note: This example is drawn from an example repo built specifically to model how to use this "unstructured data" approach. View the full repo on GitHub.*

In your Gatsby project's `gatsby-node.js` file, fetch the needed data, and supply it to the `createPage` action within the `createPages` API:

```
exports.createPages = async ({ actions: { createPage } }) => {
  // `getPokemonData` is a function that fetches our data
  const allPokemon = await getPokemonData(["pikachu", "charizard", "squirtle"])

  // Create a page that lists all Pokémon.
  createPage({
    path: `/`,
    component: require.resolve("./src/templates/all-pokemon.js"),
    context: { allPokemon }, // highlight-line
  })

  // Create a page for each Pokémon.
  allPokemon.forEach(pokemon => {
    createPage({
      path: `/pokemon/${pokemon.name}/`, // highlight-line
      component: require.resolve("./src/templates/pokemon.js"),
      context: { pokemon }, // highlight-line
    })
  })
}
```

- `createPages` is a Gatsby Node API. It hooks into a certain point in Gatsby's bootstrap sequence.
- The `createPage` action is what actually creates the page.

On the highlighted lines, the data is being supplied to the page template, where it can be accessed as props:

```
// highlight-next-line
export default function Pokemon({ pageContext: { pokemon } }) {
  return (
    <div style={{ width: 960, margin: "4rem auto" }}>
      {/* highlight-start */}
      <h1>{pokemon.name}</h1>
      <img src={pokemon.sprites.front_default} alt={pokemon.name} />
      {/* highlight-end */}
      <h2>Abilities</h2>
      <ul>
```

```
        {/* highlight-start */}
        {pokemon.abilities.map(ability => (
          <li key={ability.name}>
            <Link to={`./pokemon/${pokemon.name}/ability/${ability.name}`}>
              {ability.name}
              {/* highlight-end */}
            </Link>
          </li>
        ))}
      </ul>
      <Link to="/">Back to all Pokémon</Link>
    </div>
  )
}
```

When might using "unstructured data" make sense?

You may find this approach useful when using Gatsby's data layer feels a bit too heavy-handed for your project scope.

The pros of using unstructured data

- The approach is familiar and comfortable, especially if you're new to GraphQL
- There's no intermediate step: you fetch some data, then build pages with it

The tradeoffs of foregoing Gatsby's data layer

Using Gatsby's data layer provides the following benefits:

- Enables you to declaratively specify what data a page component needs, alongside the page component
- Eliminates frontend data boilerplate — no need to worry about requesting & waiting for data. Ask for the data you need with a GraphQL query and it'll show up when you need it
- Pushes frontend complexity into queries — many data transformations can be done at build-time within your GraphQL queries
- It's the perfect data querying language for the often complex/nested data dependencies of modern applications
- Improves performance by removing data bloat — GraphQL is a big part of why Gatsby is so fast as it enables lazy-loading the exact data in the exact form each view needs
- Enables you to take advantage of hot reloading when developing; For example, in this post's example "Pokémon" site, if you wanted to add a "see other Pokémon" section to the Pokémon detail view, you would need to change your `gatsby-node.js` to pass all Pokémon to the page, and restart the dev server. In contrast, when using queries, you can add a query and it will hot reload.

> *Learn more about GraphQL in Gatsby.*

Working outside of the data layer also means foregoing the optimizations provided by transformer plugins, like:

- `gatsby-plugin-image` (speedy optimized images),
- `gatsby-transformer-sharp` (provides queryable fields for processing your images in a variety of ways including resizing, cropping, and creating responsive images),
- ... the whole Gatsby ecosystem of official and community-created transformer plugins.

Another difficulty added when working with unstructured data is that your data fetching code becomes increasingly hairy when you source directly from multiple locations.

The Gatsby recommendation

If you're building a small site, one efficient way to build it is to pull in unstructured data as outlined in this guide, using `createPages` API, and then if the site becomes more complex later on, you move on to building more complex sites, or you'd like to transform your data, follow these steps:

1. Check out the Plugin Library to see if the source plugins and/or transformer plugins you'd like to use already exist
2. If they don't exist, read the Plugin Authoring guide and consider building your own!

Further reading

- Amberley Romo's guide to using Gatsby without GraphQL
- Why Gatsby Uses GraphQL