Contents

[Urls 2](#_Toc6065537)

[12/04/2019 [Level:expert]{react 16.8.6, Gatsby 2.3.17} 2](#_Toc6065538)

[**Core concepts** 2](#_Toc6065539)

[**Tips and tricks** 3](#_Toc6065540)

[**CLI** 10](#_Toc6065541)

[Brief 11](#_Toc6065542)

[**GraphQl** 11](#_Toc6065543)

[GraphQl Plaground 12](#_Toc6065544)

[Three syntaxes of using GraphQl 12](#_Toc6065545)

[Image processing 14](#_Toc6065546)

# Urls

**Example with explanation**

<https://dev.to/saigowthamr/build-a-blog-using-gatsbyjs--react--2lo>

**Good places for hosting static sites**

http://www.render.com

<https://www.netlify.com/>

**Tips and tricks**

<https://www.lekoarts.de/en/blog/tips-and-tricks-for-gatsby>

**GraphQL terminology understanding**

<https://blog.apollographql.com/explaining-graphql-connections-c48b7c3d6976>

**GraphQL tutorial and complete knowhow**

<https://www.howtographql.com/>

**Sample example for using image plugins in gastby**

<https://codebushi.com/using-gatsby-image/>

**Example for using .mdx or video files in Gatsby**

<https://scotch.io/tutorials/building-a-video-blog-with-gatsby-and-markdown-mdx>

**Executing multiple GQL in same query. In this blog you can use async /await in place of promise.**

<https://swas.io/blog/using-multiple-queries-on-gatsbyjs-createpages-node-api/>

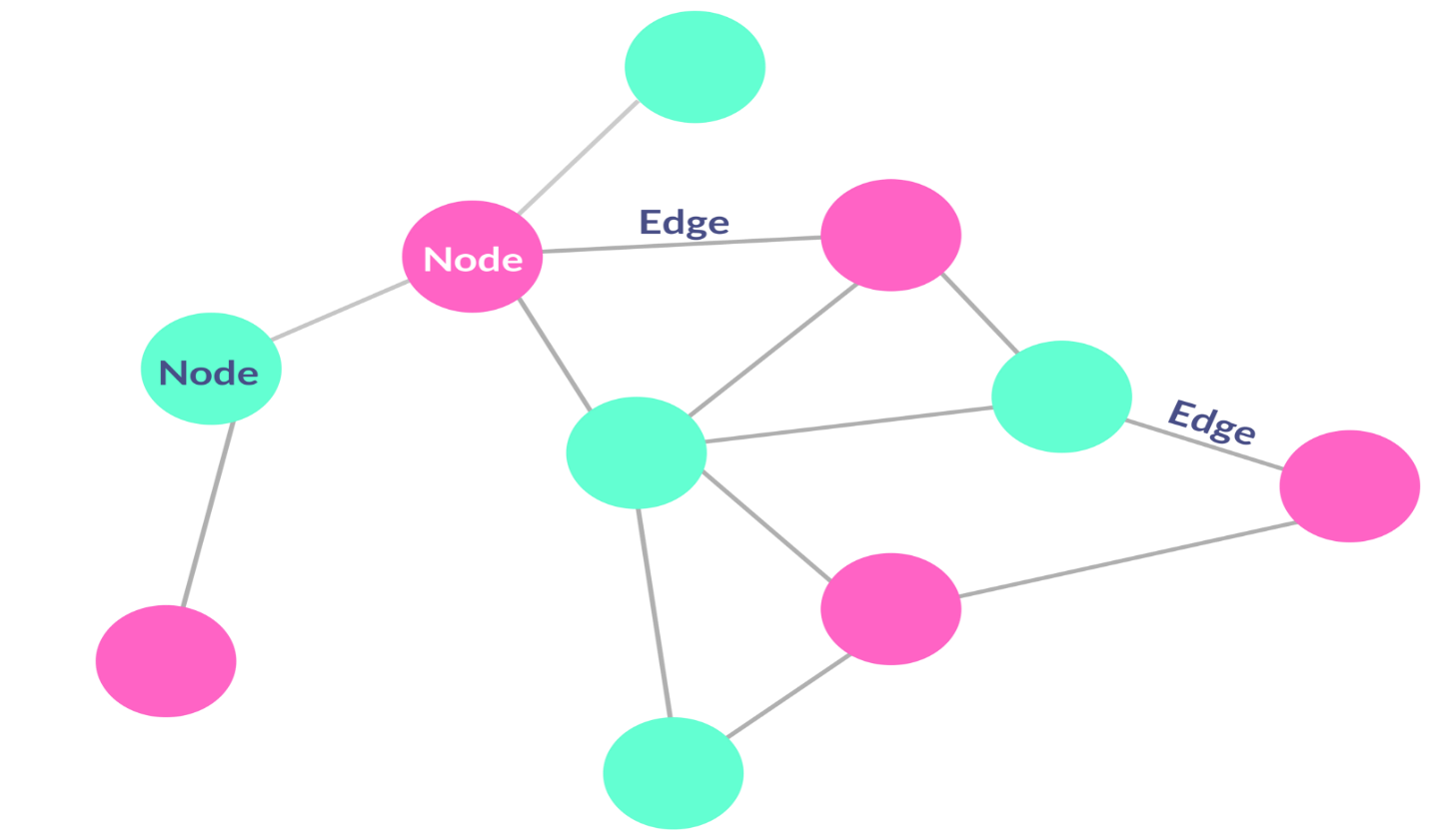
**Site analysis tools**

<https://www.webgnomes.org/blog/10-seo-analysis-tools/>

**Cheat sheet for .md files**

<http://agea.github.io/tutorial.md/>

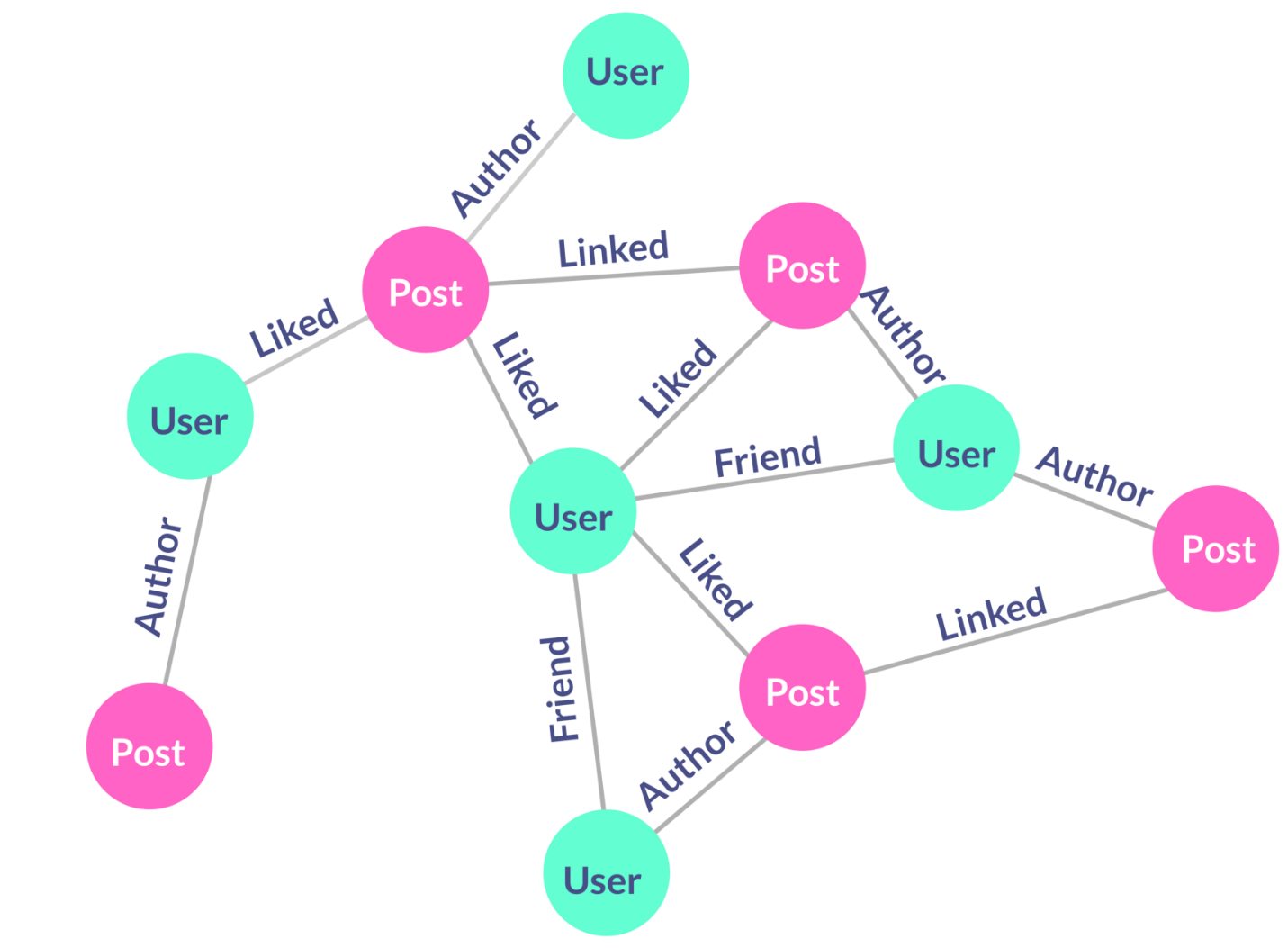
### GraphQL Nomenclature

[GraphQL](https://graphql.org/learn/) uses graphs to represent data. You can picture it to look something like this:

**Edges:** The lines that connect different nodes in the graph and represent a relationship.

**Nodes:** Records or data.

Let’s take the example of a blogging website like Medium. Nodes, in this use case, can be thought of as blog posts or users.



# 20/04/2019[level:expert]

Gatsby provides Gatsby-image library to deal with images. This library provides two formats 1) fluid, 2) fixed. I saw that in fluid format the image comes out from the surrounding container. If you want to constrain the image within the limits of outside container do the following:

* For styling use module system of react. For a component say myComp create myComp.module.scss in the same folder. Import styles from ‘./mycomp.module.scss’. Within code use className = {styles.myClass}. The module system does not allow classes to be placed in another class; this is single nesting level and no multiple nesting of classes are allowed.

Make use of fixed format in GraphQL with only giving the height and not giving width and height both. If you give the height only then width will be auto determined through aspect ratio. Sample graphQL is as follows:

graphql`{

offers:

allFile(filter:{sourceInstanceName: {eq:"offers"}, extension:{ne:"json"}}) {

edges {

node {

name

childImageSharp {

fixed(height:150) {

...GatsbyImageSharpFixed

}

}

}

}

}}`

* There is no need to convert your physical image to any other size. Browser will auto size.
* In code say the image(Img tag) resides in a div. You can size the surrounding div as the same height say 150px and say 200px width. Things will work fine.

# 12/04/2019 [Level:expert]{react 16.8.6, Gatsby 2.3.17}

## **Core concepts**

* Gatsby is used to create all pages in pages folder at build time. It keeps everything in public folder after build.
* At start of build process the Gatsby-config.js file is traversed and a GraphQL database is created through the plugins used in the config file. Then all pages are traversed and if const export query is there, which is a GQL query, it is executed, and results are passed to page function. The data portion of executed query result is passed to the page function in form of {data} destructured object parameter. You can use this data to make use of results of gql.
* Then the exported methods of Gatsby-node.js files are executed. This file follows syntax of node.js and not ES6. The ES6 commands such as export and import do not work in node.js hence they do not work in this file also. For auto creation of blogs pages from a list of .md files in a folder I used onCreateNode and createPages function of this file. OnCreateNode was used to create a field ‘slug’ in page which was used to create the page with that url. The createPages function uses createPage function to create an individual page. The createPage function tages two arguments a) path which is obtained from slug and b) component which is obtained from template for the page. The createPage is iteratively run to create several pages as obtained from gql results.
* In page you cannot use staticQuery or useStaticQuery hook, they can only be used in components. I found useStaticQuery hook very useful.
* The Gatsby-image library is used for auto image processing for very fast loading of images. It provides responsive images, blurred effect at page download time and several other effects. It uses fixed and fluid formats and is expected to do good work while loading in mobile devices.
* The <Img> tag of above library can be used in only <StaticQuery> tag or in useStaticQuery hook. These can only be used inside component and not in page.

## **Tips and tricks**

In Gatsby-config.js the siteMetaData must have a siteUrl property. This property is used in many plugins like robots and sitemap. In absence of siteUrl these plugins will result in build error of project.

To me it appears that page code is run at design time and code in components is run at run time. It will be good practice to keep maximum logic in components and not in pages. In github Gatsby -> adam1 is the demo project which depicts auto creation of blogs and image gallery concept.

One component can have only one StaticQuery or GraphQl query.

**GQL**

* The repetitive GQL code can be defined as fragment. The ...GatsbyImageSharpFluid

is provided as fragment in Gatsby-image library.

* At localhost:8000/\_\_\_graphql you get tool to test gql. Necessarily the gql results have same signature as gql query. Its good practice to test your gql here before using it in code.
* The GQL is used in Gatsby in two ways.
  + GQL function: It is used in Gatsby-node.js file. Example is

const result = await **graphql**(`{allMarkdownRemark(sort: { order: DESC, fields: [frontmatter\_\_\_date] }

limit: 1000){edges{node{fields{slug} frontmatter{title}}}}}`);

* GQL query: It is used in page as export const query = export const query = **graphql**`

query {allMarkdownRemark{edges{node{fields{slug}} node{frontmatter{description}}}}}

`

* GQL static query or useStaticQuery hookas: I used it for Images component for image gallary

const gql = **graphql**`

query {

allFile(filter: {sourceInstanceName: {eq: "images"}}) {

edges {

node {

childImageSharp {

fluid(maxWidth: 2000) {

...GatsbyImageSharpFluid

}

}

}

}

}

}`;

const data = **useStaticQuery**(gql);

const Temp = data.allFile.edges.**map**((x: any, index: number) => {

return <**Img** *key*={index} *fluid*={x.node.childImageSharp.fluid}></**Img**>

});

return Temp;

**SEO**

SEO component is auto created in default project. Use this to create meta tags in your project. It uses Helmet plugin to create meta tags. Use SEO in all the pages

**Dynamic blogs:** Creates one blog list page having links of all blogs sorted desc by date.

Idea is you put several blog articles in a folder say blogs in .md format. At build the blog pages are auto created from .md files with name of file as slug / part of url. This uses blog-template.tsx in templates folder to draw up the blog page.

In brief you do: a) Register some plugins, b) create blogs folder and keep .md files over there, c) Create templates folder and keep blog-template.tsx there d) Create blogs-listing file in pages folder to create a listing page wherein you click and go to desired blog page e) Provide a link to the blogs-listing page f) Make some changes in Gatsby-node.js file to create blog pages and associate slug with them and then restart the server.

**Step 1**: Create folders and files

Create folder /src/blogs. Put in all .md blog files here. In all blog files put in the frontmatter at the top as

---

title: My first Blog post

description: This post is related to the gatsbyjs

date: '2018-12-2'

image: ''

tags: ['javascript','react','gatsby']

---

Create a Templates folder as /src/templates to keep the blog template explained later.

**Step 2**: Do some settings in Gatsby.config file for plugins etc. This is for the purpose of reading the .md files from blogs folder and passing them through the transformer remark which provides the HTML against those .md files.

`gatsby-transformer-remark`,

{

resolve: `gatsby-source-filesystem`,

options: {

name: `blogs`,

path: `${\_\_dirname}/src/blogs`,

},

}

**Step 3**: createPages and onCreateNode in Gatsby.node.js

exports.**createPages** = async ({ actions, graphql }) => {

const { createPage } = actions;

const blogTemplate = path.**resolve**('./src/templates/blog-template.tsx');

const result = await **graphql**(`{allMarkdownRemark(sort: { order: DESC, fields: [frontmatter\_\_\_date] }

limit: 1000){edges{node{fields{slug} frontmatter{title}}}}}`);

result.data.allMarkdownRemark.edges.**forEach**(({ node }) => {

**createPage**({

path: node.fields.slug,

component: blogTemplate,

context: {

slug: node.fields.slug,

}, *// additional data can be passed via context*

});

})

}

exports.**onCreateNode** = ({ node, getNode, actions }) => {

const { createNodeField } = actions;

if (node.internal.type === `MarkdownRemark`) {

const slug = **createFilePath**({ node, getNode, basePath: `pages` });

**createNodeField**({

node,

name: `slug`,

value: slug,

});

}

}

**Step 4**: Create a template file in templates folder which has the page structure of all the blogs. The slug is passed to this template file as $slug through the context.

function **BlogTemplate**(props) {

const post = props.data.markdownRemark;

const { title,date } = post.frontmatter;

return (

<**Layout**>

<div>

<h1>{title}</h1>

<div *dangerouslySetInnerHTML*={{ \_\_html: post.html }} />

</div>

</**Layout**>

)

}

export default BlogTemplate;

export const query = **graphql**`

query PostQuery($slug: String!) {

markdownRemark(fields: { slug: { eq: $slug } }) {

html

excerpt

frontmatter {

title

}

}

}`

**Step 5**: Create the blog listing page in pages folder as blogs-listing.tsx

const **BlogsListing** = ({ data }) => (

<**Layout**>

<**SEO** *title*="Page two" />

<h1>This is blogs listing page</h1>

<p>Welcome to my blogs</p>

<**Link** *to*="/">Go back to the homepage</**Link**>

{data.allMarkdownRemark.edges.**map**((x: any, index: number) => {

return <div *key*={index}><**Link** *to*={x.node.fields.slug} >{x.node.frontmatter.description}</**Link**></div>

})}

</**Layout**>

)

export default BlogsListing

export const query = **graphql**`

query {allMarkdownRemark{edges{node{fields{slug}} node{frontmatter{description}}}}}

`

**Step 6**: Provide a link in header or wherever else to point to the blogs-listing page

<**Link** *style*={{color:'white'}} *to*="/blogs-listing/">Blogs listing</**Link**>

Now restart the server.

**Image gallery: You can create multiple image galleries**

Idea is you put several images in images folder. All those images are to be shown in image gallery. I created an Images component for the image gallery in the adam1 project. You can use fluid or fixed image. It uses Gatsby-image library and some other image processing plugins. It creates many images depending on the screen size and resolution required. The result is very fast download of images in responsive manner in multiple devices.

**Step 1**: Create images folder

Create folder as /src/images or anything else named as required.

**Step 2**: Changes in Gatsby.config

{

resolve: `gatsby-source-filesystem`,

options: {

name: `images`,

path: `${\_\_dirname}/src/images`,

},

},

`gatsby-transformer-sharp`,

`gatsby-plugin-sharp`,

**Step 3**: Create the images component to be used by pages. Here I created component ImagesLeft since I used ImagesLeft and ImagesRight as two components to be used in left and right of page in my adam1 project. For each gallery you need to create different component because in present version of Gatsby you can use a single GQL query in a page or component.

import React from "react"

import { graphql, useStaticQuery } from "gatsby"

import Img from "gatsby-image"

function **ImagesLeft**() {

const gql = **graphql**`

query {

allFile(filter: {sourceInstanceName: {eq: "images"}}) {

edges {

node {

childImageSharp {

fluid(maxWidth: 2000) {

...GatsbyImageSharpFluid

}

}

}

}

}

}`;

const data = **useStaticQuery**(gql);

const Temp = data.allFile.edges.**map**((x: any, index: number) => {

return <**Img** *key*={index} *fluid*={x.node.childImageSharp.fluid}></**Img**>

});

return Temp;

}

export default ImagesLeft;

In above in order to create two galleries we created two components. This was because a single GQL is only allowed per page. But One GQL can have multiple queries and by using multiple queries you can create single component images for two image galleries. In following code I used single GQL for two queries. Based on value of source appropriate gallery is invoked,

function AllImages({ source }) {

const gql = graphql`{

qr1:

allFile(filter: {sourceInstanceName: {eq: "images1"}}) {

edges {

node {

childImageSharp {

fluid(maxWidth: 2000) {

...GatsbyImageSharpFluid

}

}

}

}

}

qr2:

allFile(filter: {sourceInstanceName: {eq: "images"}}) {

edges {

node {

childImageSharp {

fluid(maxWidth: 2000) {

...GatsbyImageSharpFluid

}

}

}

}

}

} `;

const data = useStaticQuery(gql);

const sourceObject = {

images: data.qr1.edges,

images1: data.qr2.edges

}

const Temp = sourceObject[source].map((x: any, index: number) => {

return <Img key={index} fluid={x.node.childImageSharp.fluid}></Img>

});

return Temp;

}

export default AllImages;

**Css / scss at component and global level**

You can use sass in typescript in Gatsby projects

`gatsby-plugin-typescript`,

`gatsby-plugin-sass`,

In following manner, you can provide styling to Gatsby projects:

**Method 1**: Global style in layout.js. In layout.js component file use layout.scss and use the layout component in many places. This is default way.

**Method 2**: Global css

Create a global.scss file and import it in Gatsby.browser.js

import "./src/styles/global.css"

**Method 3**: module

For each component create a <component name>.module.scss file and import it in the component. Put a container class in the module file and do nesting of all items in container class.

import React from "react"

import styles from "./custom1.module.scss"

function **Custom1**() {

return <div *className*= {styles.container}>

<input *type*="text" ></input>

<button>Click me1</button>

</div>

}

export default Custom1;

custom1.module.scss file

.container {

input {

width: 500px;

color:red;

height:20px;

line-height: 100%;

}

}

You can also create a folder for each component and keep the index.js and the .module.scss file in the named container. It will work with file name as index.tsx / js

**Start a project capital-chowringhee**

1. **Plugins: In Gatsby.config.js and npm install**

`gatsby-plugin-typescript`,

`gatsby-plugin-sass`,

`gatsby-transformer-remark`,

`gatsby-transformer-sharp`,

`gatsby-plugin-sharp`,

`gatsby-plugin-robots-txt`,

1. I saw that Gatsby introduces a style body{margin:8px;}. To remove that put body{margin:0;} in layout.scss file.

# **CLI**

Npm install Gatsby -g

gatsby new projectname

npm start

npm run build

Gatsby is static page generator. When you build the project, physical pages are created in the public folder. You just copy the files of public folder in your website. It uses Design time and not run time GraphQL to generate static pages.

# Brief

You can write components in components folder. All .js files you write in pages folder create static pages when you build the project. All build files are in public folder.

* Gatsby uses plugins to do various jobs. You need to register the plugin in .config file.
* The purpose of GraphQl is to query nodes available to the application through various plugins and render them. Various plugins create nodes which you can also query at localhost:8000/\_\_\_graphql which is a tool called GraphIql.
* Gatsby-node.js file is used to create new nodes for GraphQl query which you can use in your pages and components.
* gatsby-source-filesystem

This plugin creates nodes for file system files

gatsby-plugin-sass is used to apply .scss files instead of .css. you need to install node-sass also

This is GraphQl query

{

site {

siteMetadata {

title

}

}

}

Which returns this:

{

"site": {

"siteMetadata": {

"title": "A Gatsby site!"

}

}

}

Query signature matches the return data.

# **GraphQl**

Localhost:8000/\_\_\_graphql

There are three underscores.

## GraphQl Plaground

Playground is better tool for querying GraphQL. To start it use following command in package.json file for “develop” property

“develop” : “SET GATSBY\_GRAPHQL\_IDE=playground&gatsby develop”

Then query at <http://localhost:8000/___graphql>

Sometimes it fails. In that case use command line and type SET GATSBY\_GRAPHQL\_IDE=playground and then give command gatsby develop

For GraphQL several ways are there but latest and easiest is useStaticQuery hook. It straightaway returns data from the query. Example is:

import React from "react"

import { useStaticQuery, graphql } from "gatsby"

export default () => {

const data = useStaticQuery(graphql`

query HeaderQuery {

site {

siteMetadata {

title

}

}

}

`)

return (

<header>

<h1>{data.site.siteMetadata.title}</h1>

</header>

)

}

Limitation of useStaticQuery is Only one per page is allowed and you cannot use context to pass data to component while using useStaticQuery.

## Three syntaxes of using GraphQl

**Syntax 1: Page query**

1. Return a const as query from the page
2. In Component function of the page get the destructured data object

Remember that the query is to be exported in page in exactly this syntax as: export const query =…

You cannot do like this: const query = …; export {query}

In fact at compile time Gatsby removes all exported queries from the page and executes and provided the result to the page component.

import React from "react"

import { graphql } from "gatsby"

export default ({ data }) => (

<div>

<h1>About {data.site.siteMetadata.title}</h1>

<p>We're a very cool website you should return to often.</p>

</div>

)

export const query = graphql`

query {

site {

siteMetadata {

title

}

}

}

`

**Syntax 2: Static query**

This syntax is useful at components. StaticQuery has two parts a) query and b) render. Result of query is provided to render function as data.

import React from "react"

import { StaticQuery, graphql } from "gatsby"

export default () => (

<StaticQuery

query={graphql`

query HeadingQuery {

site {

siteMetadata {

title

}

}

}

`}

render={data => (

<header>

<h1>{data.site.siteMetadata.title}</h1>

</header>

)}

/>

)

**Syntax 3: useStaticQuery hook**

Same as static query but in hooks format. Can be reused as custom hooks.

## Image processing

* Install plugins and Gatsby-image library.

npm install --save gatsby-image gatsby-transformer-sharp gatsby-plugin-sharp

set Gatsby-config file accordingly

provide `gatsby-plugin-sharp`, `gatsby-transformer-sharp` in Gatsby-config file.

* Set the source file plugin to point to image files

{

resolve: `gatsby-source-filesystem`,

options: {

name: `images`,

path: `${\_\_dirname}/src/images`,

},

}

Gatsby-image provides a tag <Img>. This tag can be used with Static query only. The Staticquery cannot be used with pages and can be used with components only. So You need to implement Gastby images at the component level and include the component in page.

The best id useStaticQuery hook. It facilitates in reusing the code. It can only be used at component and not in pages. A tested sample is as below to show all images in /src/images folder.

function **Images**() {

const gql = **graphql**`

query {

allFile(filter: {sourceInstanceName: {eq: "images"}}) {

edges {

node {

childImageSharp {

fluid(maxWidth: 2000) {

...GatsbyImageSharpFluid

}

}

}

}

}

}`;

const data = **useStaticQuery**(gql);

const Temp = data.allFile.edges.**map**((x: any, index: number) => {

return <**Img** *key*={index} *fluid*={x.node.childImageSharp.fluid}></**Img**>

});

return Temp;

}

export default Images;

In above code the ...GatsbyImageSharpFluid is called graphql fragment. A graphql fragment is a collection of several lines of graphql code. It is used for repetitive code. ...GatsbyImageSharpFluid is provided out of the box by Gatsby-image.

In any page you can include “return <Images></Images>” to have all the images.