Every site on the web has basic *meta-tags* like the title, favicon or description of the page in their `<head>` element. This information gets displayed in the browser and is used when someone shares your website, e.g. on Twitter. You can give your users and these websites additional data to embed your website with more data — and that's where this guide for a SEO component comes in. At the end you'll have a component you can place in your layout file and have rich previews for other clients, smartphone users, and search engines.

Note: This component will use `useStaticQuery`. If you're unfamiliar with that, have a look at the useStaticQuery documentation. You also have to have `react-helmet` installed for which you can have a look at this document.

gatsby-config.js

Gatsby automatically exposes the `siteMetadata` section of the `gatsby-config` file in the GraphQL datalayer. It's considered best practice to place your site meta information there.

```
module.exports = {
  siteMetadata: {
    title: "Severus Snape",
    titleTemplate: "%s · The Real Hero",
    description:
      "Hogwarts Potions master, Head of Slytherin house and former Death Eater.",
    url: "https://www.doe.com", // No trailing slash allowed!
    image: "/snape.jpg", // Path to the image placed in the 'static' folder, in the
project's root directory.
    twitterUsername: "@occlumency",
  },
}
```

SEO component

First create a new component with this initial boilerplate.

```
import React from "react"
import PropTypes from "prop-types"
import { Helmet } from "react-helmet"
import { useLocation } from "@reach/router"
import { useStaticQuery, graphql } from "gatsby"

const SEO = ({ title, description, image, article }) => ()

export default SEO

SEO.propTypes = {
  title: PropTypes.string,
  description: PropTypes.string,
  image: PropTypes.string,
  article: PropTypes.bool,
}

SEO.defaultProps = {
  title: null,
```

```
  description: null,
  image: null,
  article: false,
}
```

Note: `propTypes` are included in this example to help you ensure you're getting all the data you need in the component, and to help serve as a guide while destructuring / using those props.

As the SEO component should also be usable in other files, e.g. a template file, the component also accepts properties for which you set sensible defaults in the `SEO.defaultProps` section. This way the information you put into `siteMetadata` gets used every time unless you define the property explicitly.

Now define the query and pass it to `useStaticQuery`. You can also alias query items, so `title` gets renamed to `defaultTitle`.

```
const SEO = ({ title, description, image, article }) => {
  const { pathname } = useLocation()
  const { site } = useStaticQuery(query)

  return null
}

export default SEO

const query = graphql`
  query SEO {
    site {
      siteMetadata {
        defaultTitle: title
        titleTemplate
        defaultDescription: description
        siteUrl: url
        defaultImage: image
        twitterUsername
      }
    }
  }
`
```

The next step is to destructure the data from the query and create an object that checks if the props were used. If not, the default values are applied. Aliasing the properties comes in handy here to avoid name collisions.

```
const SEO = ({ title, description, image, article }) => {
  const { pathname } = useLocation()
  const { site } = useStaticQuery(query)

  const {
    defaultTitle,
    titleTemplate,
    defaultDescription,
    siteUrl,
```

```
    defaultImage,
    twitterUsername,
  } = site.siteMetadata

  const seo = {
    title: title || defaultTitle,
    description: description || defaultDescription,
    image: `${siteUrl}${image || defaultImage}`,
    url: `${siteUrl}${pathname}`,
  }

  return null
}

export default SEO
```

The last step is to return this data with the help of `Helmet`. Your complete SEO component should look like this.

```
import React from "react"
import PropTypes from "prop-types"
import { Helmet } from "react-helmet"
import { useLocation } from "@reach/router"
import { useStaticQuery, graphql } from "gatsby"

const SEO = ({ title, description, image, article }) => {
  const { pathname } = useLocation()
  const { site } = useStaticQuery(query)

  const {
    defaultTitle,
    titleTemplate,
    defaultDescription,
    siteUrl,
    defaultImage,
    twitterUsername,
  } = site.siteMetadata

  const seo = {
    title: title || defaultTitle,
    description: description || defaultDescription,
    image: `${siteUrl}${image || defaultImage}`,
    url: `${siteUrl}${pathname}`,
  }

  return (
    <Helmet title={seo.title} titleTemplate={titleTemplate}>
      <meta name="description" content={seo.description} />
      <meta name="image" content={seo.image} />

      {seo.url && <meta property="og:url" content={seo.url} />}
```

```
      {(article ? true : null) && <meta property="og:type" content="article" />}

      {seo.title && <meta property="og:title" content={seo.title} />}

      {seo.description && (
        <meta property="og:description" content={seo.description} />
      )}

      {seo.image && <meta property="og:image" content={seo.image} />}

      <meta name="twitter:card" content="summary_large_image" />

      {twitterUsername && (
        <meta name="twitter:creator" content={twitterUsername} />
      )}

      {seo.title && <meta name="twitter:title" content={seo.title} />}

      {seo.description && (
        <meta name="twitter:description" content={seo.description} />
      )}

      {seo.image && <meta name="twitter:image" content={seo.image} />}
    </Helmet>
  )
}

export default SEO

SEO.propTypes = {
  title: PropTypes.string,
  description: PropTypes.string,
  image: PropTypes.string,
  article: PropTypes.bool,
}

SEO.defaultProps = {
  title: null,
  description: null,
  image: null,
  article: false,
}

const query = graphql`
  query SEO {
    site {
      siteMetadata {
        defaultTitle: title
        titleTemplate
        defaultDescription: description
        siteUrl: url
        defaultImage: image
```

```
        twitterUsername
      }
    }
  }
`
```

Examples

You could also put the Facebook and Twitter meta-tags into their own components, add custom favicons you placed in your `static` folder, and add schema.org data (Google will use that for their Structured Data). To see how that works you can have a look at these two examples:

- marisamorby.com
- gatsby-starter-prismic

As mentioned at the beginning you are also able to use the component in templates, like in this example.