**Proyecto 15**

**1.** Debemos instalar globalmente el cliente de Gatsby: **npm i -g gatsby-cli**

**2.** Creamos el proyecto con: **gatsby new NombreProyecto**

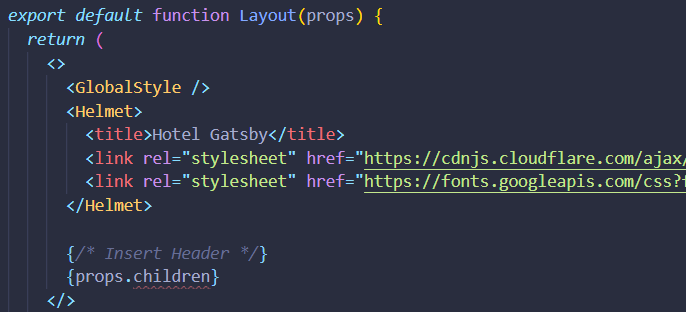
**3.** Levantamos el servidor con: **gatsby develop**

**4.** Al levantar el servidor tendremos **2 URL**: **una** para **ver la página** y **otra** para **GraphQL** donde construiremos nuestras **query** hacia una determinada **API**.

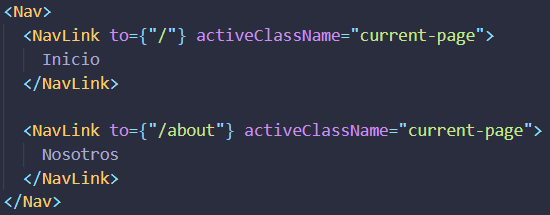
**5.** **Eliminamos** todos los **archivos** dentro de la carpeta “**components**” e “**images**” (menos el png). En “**pages**” **dejamos** únicamente el **index.js y 404.js** (sacándole cualquier import de archivos eliminados)

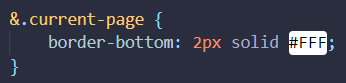
**6**. **Configuramos** **styled-components** siguiendo los pasos de [esta página](https://www.gatsbyjs.com/docs/how-to/styling/styled-components/).

**7**. **globalStyle.js**: creamos el archivo e incluimos todos los estilos globales de la página.

**8**. **Layout.js**: creamos el componente Layout.js que **será** el **contenedor** **padre** de **todas** las “**pages**”. Importamos los **estilos globales** y lo utilizamos al inicio del componente, luego requerimos **Helmet** e incluimos unas **fonts de Google** y **normalize**. Por último, le **pasamos** **props.children** para que **renderize** a todos sus **hijos**.

**HEADER DE LA PÁGINA**

**9**. **NavBar.js**: creamos el componente y sus styled-components (una cosa muy importante es que en Gatsby podemos crearle un estilo a <Link>). Le **damos** la **propiedad** **activeClassName** al <**NavLink**> con el valor “**current-page**”, esto **hará** que el <Link> **tome** **esa clase** únicamente si la **URL actual** **coincide** con su **ruta**. De esta forma, podemos marcar en la barra de navegación en qué lugar estamos.

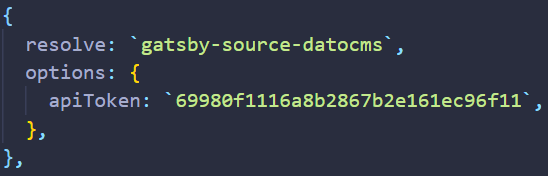


**10. Header.js**: creamos el componente y sus styled-components (GoToHome es un styled de <Link>). **Importamos e incluimos** el **NavBar.js** después del título. Vamos a **Layout.js** y **colocamos el Header** para que este sea **visible** en **todas las vistas** que tengan a Layouts.js como padre.



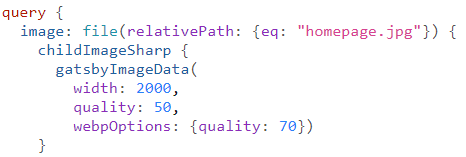
**CONECTANDO DATO CMS CON GATSBY**

**11.** Instalamos: **npm i gatsby-source-datocms**

**12. gatsby-config.js**: al final del array agregamos un objeto con la **configuración** de **Dato CMS**. Esto nos **permitirá** utilizar **métodos** de **Dato CMS** **en los query**.

**HOME DEL SITIO**

**Accediendo a la imagen local** – Hero Image **(Graphql)**

**13.** Dentro de la carpeta “**images**” **ponemos** la imagen de bienvenida del sitio (**homepage.jpg**) y luego entramos en la **interfaz gráfica de Graphql** en <http://localhost:8000/___graphql> para, mediante una **consulta**, **acceder** a la **imagen** **local** y ver el resultado que nos trae.

- **query**: así es como se inicia cualquier consulta de Graphql.

- **Image**: es el **nombre** que queremos que tenga el **objeto** **devuelto** por la **consulta**, es algo **opcional**, puede no estar.

- **file**: indica que se debe buscar de forma interna en el proyecto aquel archivo que coincida con “homepage.jpg”.

- **childImageSharp**: lugar donde se deben buscar todas las consultas de imágenes.

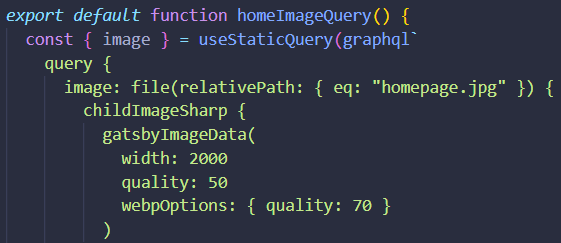
- **gatsbyImageData**: es lo que hay que usar para recolectar la imagen. Le **damos** un par de **propiedades** como el **ancho**, la **calidad** y la **calidad** **webpOptions**, [en este video](https://youtu.be/d5i00ZN2Tuw?t=453) se explica cada una.

**Accediendo a la imagen local** – Hero Image **(VScode)**

Como queremos una **imagen de fondo** con texto arriba, debemos **instalar** una **librería** para que la **etiqueta de imagen admita hijos**. Todos los **hijos** se **pondrán** por **encima de la imagen**.

**14.** Instalamos **npm install --save gatsby-background-image** **gbimage-bridge**. Esto es la librería de imagen de fondo y el bridge es del cual extraeremos el componente que se utilizara como imagen.

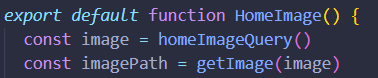
**15**. **homeImageQuery.js**: **creamos** el **archivo** e importamos **graphql** & **useStaticQuery** para poder hacer las **consultas.** **Ejecutamos** la misma **consulta** que en el paso (11), **desestructurando** y **retornando** el **objeto** “**image**” que nos devuelve.



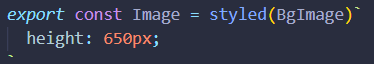


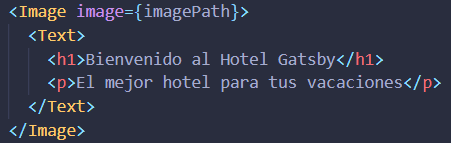
**16. HomeImage.js:** creamos el componente y realizamos lo siguiente

**a.** Importamos el archivo **homeImageQuery.js** y “**getImage**” que **sirve** para **obtener** la ruta de la **imagen** justo cuando empieza el **objeto childImageSharp.**

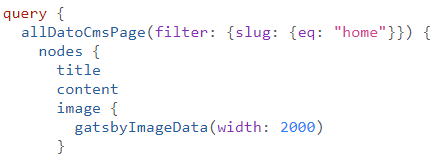
**b. Guardamos** la **información** de la **consulta** en una constante y luego **obtenemos** **su path** con **getImage**.

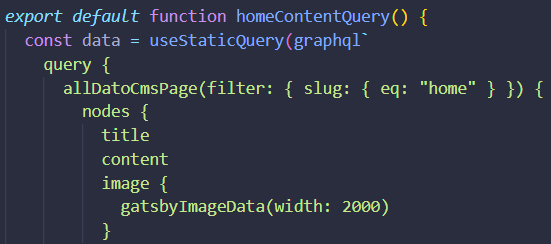
**c.** importamos “**BgImage**”, le creamos un **styled-component** y lo **utilizamos** **como** una **etiqueta** de **imagen** pasándole a la **propiedad** **image** el **path** de la **imagen** obtenido en el paso anterior.





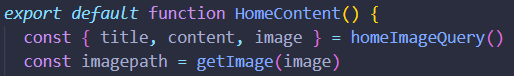
**Contenido que esta debajo de la imagen Hero**

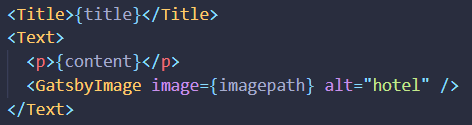
**17**. Vamos a la interfaz gráfica de **graphql** y hacemos la **consulta** del **registro** de “**home**” que cargamos en el **modelo** “**Page**”. La **lógica** del query es la **siguiente**: en **allDatoCmsPage** (refiere al modelo de “Page”) **filtramos** **por el slug** (el identificador único del registro) que sea “**home**”. Dentro de ese registro vamos a **nodes** y **tomamos** su **título**, **contenido** e **imagen**.

**18**. **homeContentQuery.js:** ejecutamos el query del paso anterior y retornamos el objeto que nos devuelve.

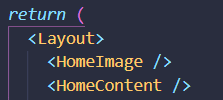


**19**. **HomeContent.js:** **creamos** el **componente**, importamos **getImage**, **GatsbyImage** y **homeContentQuery.js. Desestructuramos** la información del query, obtenemos el **path** de la **imagen** y **renderizamos** la **información**. Para desplegar la imagen utilizamos GatsbyImage.

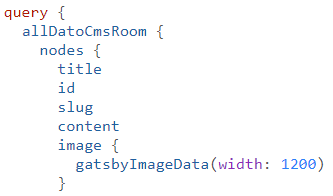


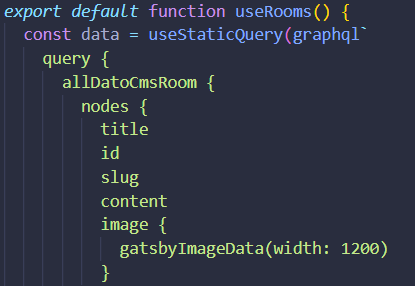
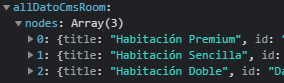


**20**. **index.js**: importamos y utilizamos **HomeImage.js** & **HomeContent**.**js**



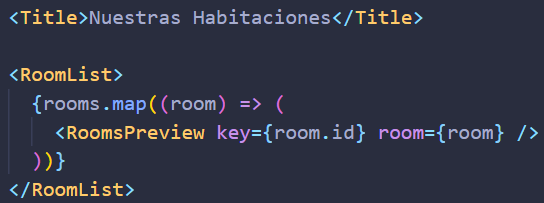
**MOSTRANDO HABITACIONES EN LA HOME**

**21**. Hacemos la **consulta** de toda la **data** de **habitaciones** en la **interfaz gráfica** para **ver** qué **resultado nos trae**.

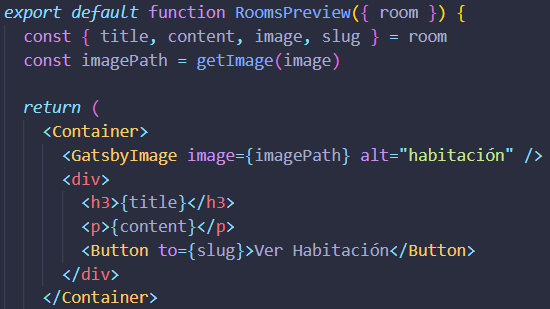
**22**. **useRooms.js**: **creamos** el **hook useRooms** y hacemos el **pedido** con el **query** del **paso anterior**. Como podemos ver la **información** con todas las **habitaciones** **esta** en **allDatoCmsRoom.nodes** en un **array** de **objetos**, razón por la cual vamos a hacer que el **hook** **retorne** **ese array**.



**23**. **Index.js**: **utilizamos** el **hook** y realizamos un **map** del **array** que nos trae **renderizando** en **cada iteración** el **componente** <**RoomsPreview**> que vamos a crear en el siguiente paso. Al **título** y a la **lista** le hacemos un **styled-component**.



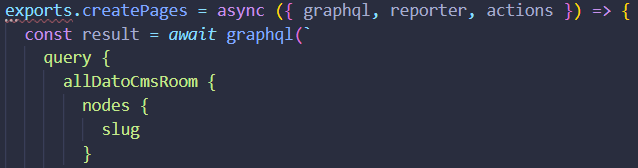
**24**. **RoomPreview.js**: **desestructuramos** las **props** y las **utilizamos** como contenido. En la **imagen** debemos **importar getImage y GatsbyImage**. Creamos los styled-components y cuando el **usuario** haga **click** en el <**Button**> lo llevara a la **URL** del **slug**, por ejemplo <http://localhost:8000/hab3>.

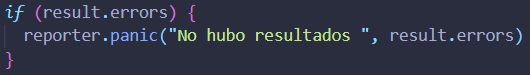


**PÁGINAS DINAMICAS DE CADA HABITACIÓN**

**25. Gatsby-node.js:** aquí debemos poner la **configuración** para la creación de las **páginas** **dinámicas**. Iniciaremos con **exports.createPages** y creamos una **función asincrónica**.

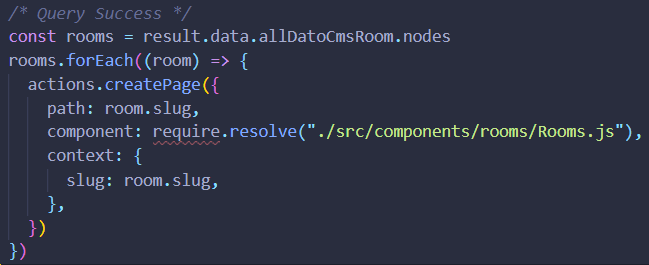
Debemos **desestructurar** **3 parámetros**: **graphql** (para hacer la consulta), **reporter** (para alertar en caso de error) y **actions** (para crear las páginas).

**a.** Hacemos una consulta trayendo el slug de todas las habitaciones.

**b.** Validamos que no haya errores.

**c**. Guardamos en una constante la información que viene del query (es un array).

**d.** Con un **forEach** **recorremos** el **array** **creando una página** en **cada iteración** con **action.createPage()** que **recibe** **3 parámetros**: **path** (es la URL de la página a crear), **component** (componente donde se desplegara la información) y **context** (será la variable que nos servirá para hacer la consulta de la información de cada habitación en particular)

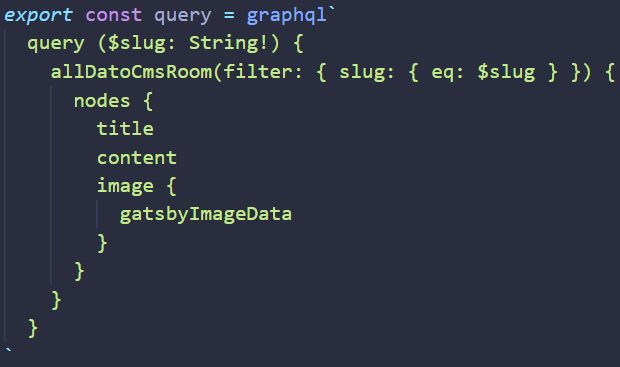


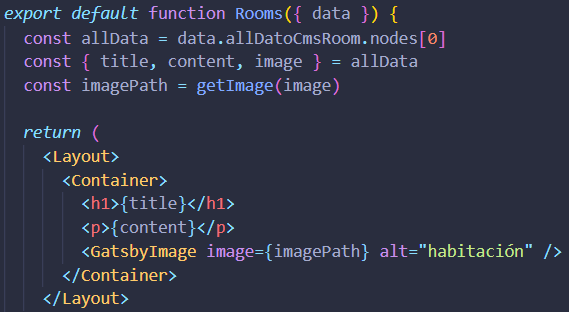
**26. Rooms.js**: **creamos** el **componente** y hacemos una **consulta** **dinámica**. En este caso **no se usa useStaticQuery** y se debe **exportar el query**. La lógica es la siguiente:

**a.** **Utilizamos** la **variable** slug (**$slug**) que nos **viene** desde el **context** de **Gatsby-node.js** y **establecemos** que es un **String** **obligatorio**.

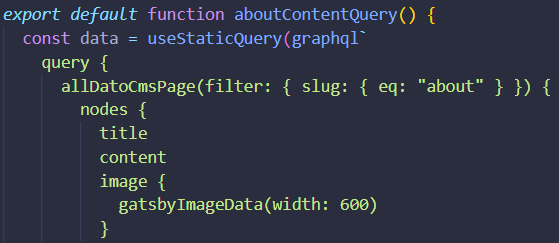
**b.** En **allDatoCmsRoom** **filtramos** los **registros** a **través** del **slug** diciendo que **debe** que **ser** **igual** a la **variable** **$slug.** Esto nos traerá el registro de la habitación que coincida con el slug pasado.

**c.** Tomamos los datos del registro filtrado.



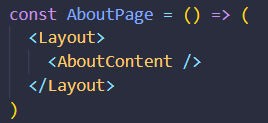
**27. Rooms.js**: el **componente** debe **recibir** como **props** “**data**” (que **contiene** la **información** de la **consulta**), luego de ella desestructuramos los datos y los utilizamos para mostrar el contenido.

**PÁGINA DE ABOUT**

**28**. **AboutContentQuery.js**: **clonamos** el archivo **homeContentQuery.js** y los **únicos cambios** que vamos a realizar van a ser en los **styled-components** y en **slug** del **query**.

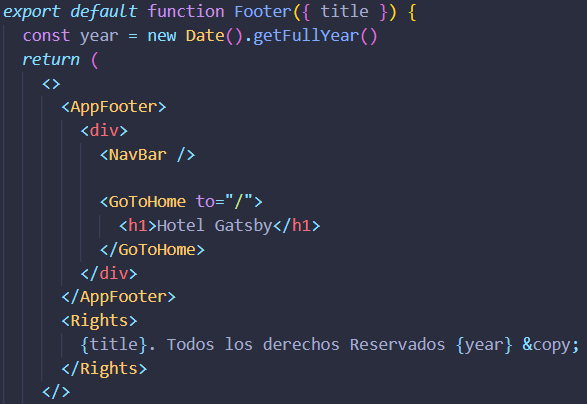


**29**. **AboutContent.js**: clonamos el **HomeContent.js** y únicamente le cambiamos el import del query por **AboutContentQuery.js**

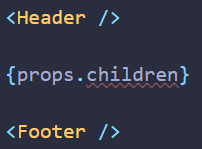
**30**. **about.js**: importamos y utilizamos el componente **AboutContent.js**

**FOOTER**

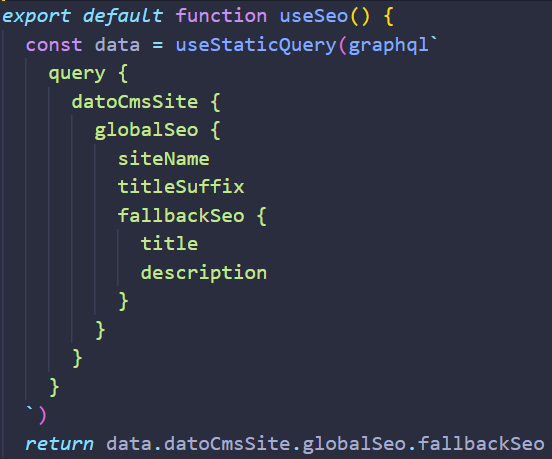
**31. Footer.js**: **clonamos** el componente **Header.js** y lo renombramos. Le pasamos la **prop** “**title**”, **cambiamos** los **styled-components** y hacemos un **enroque** entre la **barra de navegación** y el **título**. **Obtenemos** el **año actual** y lo utilizamos al final del componente para **mostrar los derechos**.



**32. Layout.js**: importamos y **utilizamos** el **Footer.js** luego del props.children.



**SEO**

**33. useSeo.js:** creamos el hook y hacemos una **consulta** para **obtener** el “**title**” y “**description**” del SEO que creamos en **Dato CMS**. Luego retornamos el objeto que nos devuelve.

**34. Layout.js:** **importamos** y **desestructuramos** la información del **hook** y la **usamos** para **completar** la etiqueta <**meta**> de **descripción** y para **pasarle** el **título** de la página al **Footer**.



