**Primeros pasos con gulp e instalar dependencias**

Iniciamos la terminar y con npm init inicializamos el proyecto

Seguimos las indicaciones de la terminal y al final nos va crear un archivo package.json, permite ejecutar scripts de npm y de almacenar las dependencias.

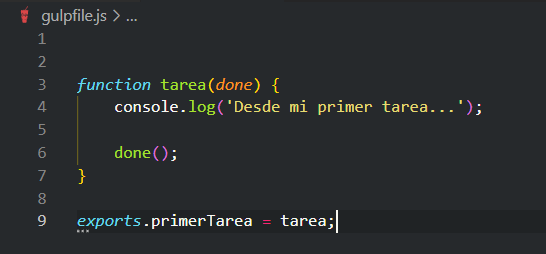
**Instalamos gulp**

**npm i gulp --save-dev** es una dependencia de desarrollo.

Se crea la carpeta node\_modules y en el package.json se crean las dependencias

Creamos manualmente un archivo llamado **gulpfile.js**

**Creando primer tarea en Gulp**



En consola es gulp primerTarea

**Compilar SASS con gulp**

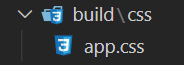
Instalamos la dependencia de SASS.

**npm i --save-dev sass gulp-sass**

En el gulpfile creamos la función que va a compilar la hoja de sass a css



Y la compila acá

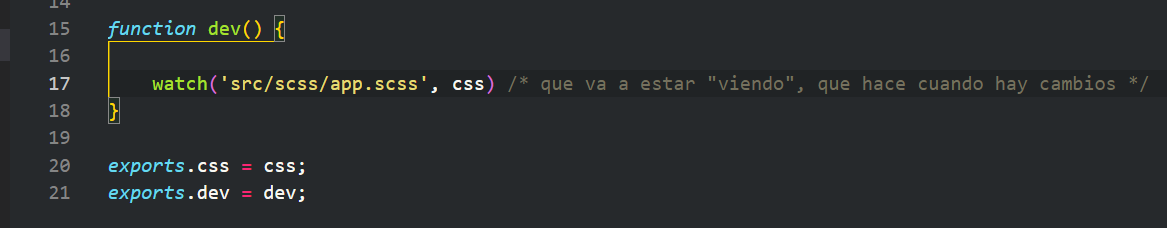


Una vez compilada la ligamos a nuestro HTML para que empiece a tomar los estilos

**Agregar un watch**

Escucha por los cambios en el app.scss para no tener que ejecutar en la consola el comando para que guarde los cambios.

En el gulpfile importamos el watch, y hacemos una nueva función



Y en consola usamos **gulp dev** y en automático va a estar atento a los cambios.

**Añadir autoprefixr**

Instalamos estas 2 dependencias

**npm i --save-dev autoprefixer gulp-postcss**

Sirve para crear código que va a ser soportado por navegadores que no son completamente compatibles con css.

**Corriendo múltiples tareas por default**

Son funciones que se exportan mediante

*exports*.default = lafuncion;

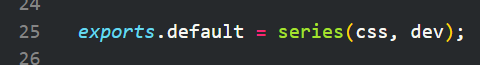
Y que en consola al ser llamadas con gulp se ejecutan en automático sin tener que poner su nombre.

Hay 2 formas de usarlo, se tienen que importar

// Series: Se inicia una tarea y hasta que finaliza inicia la siguiente

// Parallel - Todas inician al mismo tiempo

El gulp queda



**Dividir el código de SASS en diferentes archivos**

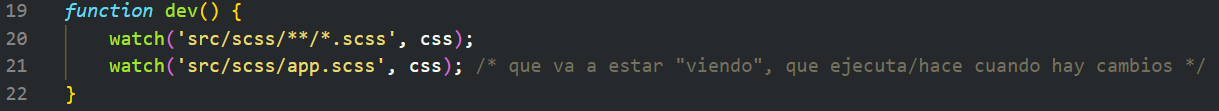
En la carpeta scss se crea una carpeta llamada “header” y ahí se colocará todo el código relacionado con el header, dentro contendrá un archivo que SIEMPRE iniciará con guión bajo, significa que ese archivo se va a incluir en otro.

Para que se tome en cuenta durante la compilación, en el app.scss debemos colocar:



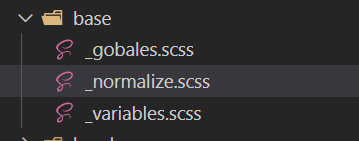
**Escuchar por los cambios en todos los archivos**

Dentro de la función dev

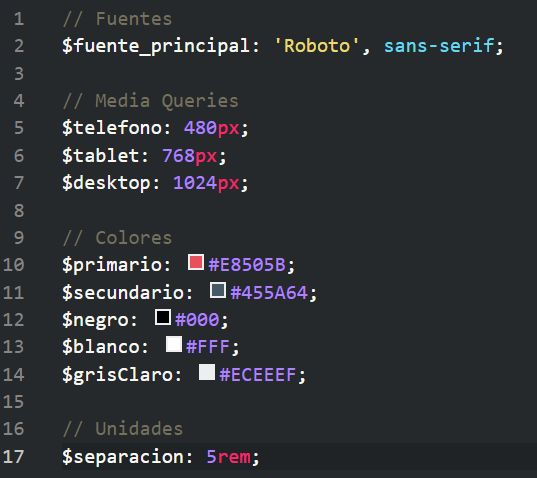
**

**Creando variables en SASS**

Dentro de la carpeta sass creamos una nueva carpeta llamada base con los siguientes archivos



Creamos las variables



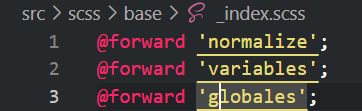
Y copiamos el código de normalize dentro de su respectivo archivo.

**Utilizando @use y @forward**

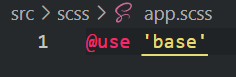
Dentro de la carpeta base creamos un archivo **\_index.scss** donde se van a listar los archivos que van a estar en esa carpeta.



En el \_index,scss:

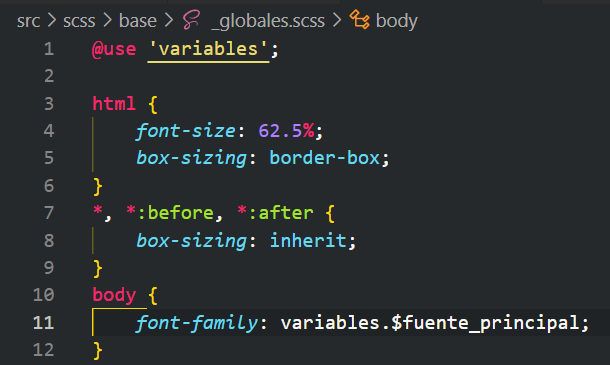


En el app.scss

 Por default va a tomar el \_index

**Importar variables en otros archivos**

La importamos con @use

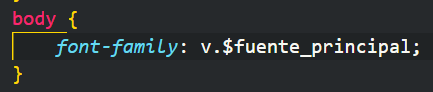


Y para usarla es como un arreglo en JS: variables.$fuente\_principal

Para simplificar, desde el use que lo importa podemos definirle un “alias”



Y al usarla solo se escribe



**Agregando el resto de CSS al global.**

@use 'variables' as v;

html {

*font-size*: 62.5%;

*box-sizing*: border-box;

}

\*, \*:before, \*:after {

*box-sizing*: inherit;

}

body {

*font-family*: v.$fuente\_principal;

*font-size*: 1.6rem;

}

p {

*color*: v.$negro;

*font-size*: 2rem;

}

.contenedor {

*width*: 95%;

*max-width*: 120rem;

*margin*: 0 auto;

}

a {

*text-decoration*: none;

}

img {

*max-width*: 100%;

*display*: block;

}

h1, h2, h3 {

*margin*: 0 0 v.$separacion 0;

}

h1{

*font-size*: 4rem;

}

h2{

*font-size*: 3.6rem;

}

h3{

*font-size*: 3.2rem;

}

**Creando una serie de utilidades.**

En base creamos un **\_utilidades.scss**

Y lo incluimos en el **\_index.scss**

**\_utilidades.scss**

@use 'variables' as v;

.text-center {

*text-align*: center;

}

.bg-primario {

*background-color*: v.$primario;

}

**Agregando más archivos a la compilación**

**HTML Creando el header del proyecto**

**Creando una tarea para las imágenes**

La carpeta de imágenes se coloca dentro de la carpeta src

En el gulpfile vamos a crear una tarea para las imágenes

*function* imagenes(*done*) {

    src('src/img/\*\*/\*')

        .pipe( dest('build/img') );

    done();

}

La incluimos en el exports.default y en el dev un watch





**Aligerar imágenes con gulp**

Instalamos una nueva dependencia **npm i --save-dev gulp-imagemin**

En el gulpfile

// Imágenes

*const* imagemin = require('gulp-imagemin');

Y en la función

*function* imagenes(*done*) {

    src('src/img/\*\*/\*')

        .pipe( imagemin({ optimizationLevel: 3 }) )

        .pipe( dest('build/img') );

    done();

}

**Utilizar gulp para crear imágenes webp**

Instalamos una nueva dependencia **npm i -–save-dev gulp-webp**

En el gulpfile

*const* webp = require('gulp-webp')

*function* convertwebp() {

    return src('src/img/\*\*/\*.{png,jpg}')

        .pipe( webp() )

        .pipe( dest('build/img') )

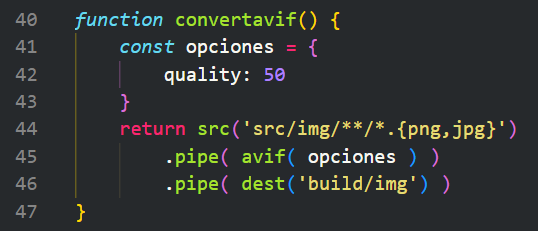
}

Y se añade al exports.default

**Utilizar gulp para crear imágenes Avif**

Instalamos una dependencia **npm i --save-dev gulp-avif**

*const* avif = require('gulp-avif');



**Creando el css de nuestro header.**

Para el sombreado negro de la imagen usamos ::before y ajustamos position relative y absolute, para el before se usa &::before para que el & quite el espacio que se renderiza de scss a css, si no en la hoja queda .header ::before

@use 'base/variables' as v;

.header {

*padding*: 5rem 0;

*position*: relative;

    &::before {

        content: '';

*background-color*: v.$negro;

*position*: absolute;

*top*: 0;

*left*: 0;

*right*: 0;

*bottom*: 0;

*opacity*: .6;

    }

    .contenido-header {

*position*: relative;

    }

}

.header-inicio {

*background-image*: url('../img/header\_inicio.jpg');

}

**Mixins en CSS**

Forma de reutilizar código

En base se crea **\_mixins.scss**

@mixin color {

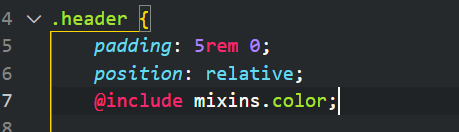
*color*: black;

*font-size*: 2rem;

}

Se crea un mixin de esa forma, color es el nombre y lo que está dentro de las llaves es el cuerpo del mixin.

Para usarlo se importa a la hoja donde estemos trabajando y se llama





Y son reutilizables

**Otro tipo de mixin**

@mixin bgColor($color) {

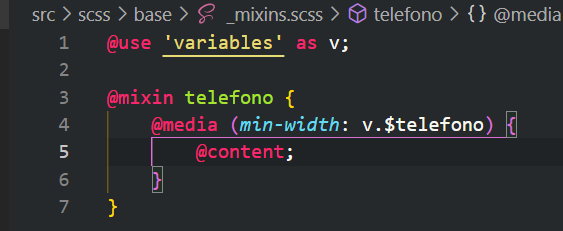
*background-color*: $color ;

}

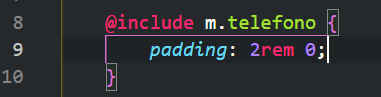
Se le pasa el valor

@include m.bgColor(red)

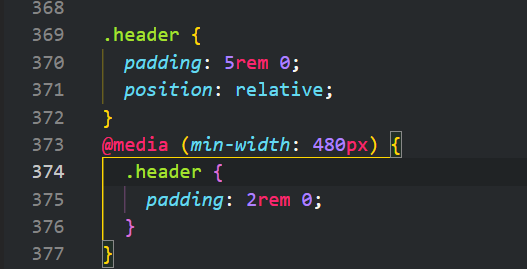
**Mixins para MQ**

****

@content indica que dentro de esas llaves habrá código, entonces al usar el mixin será:



Y lo que está dentro de las llaves será el @content, por ende al compilar en la hoja app.css se compilará..



**Shorthand para mixins**

Un snippet nada más

**SCSS a la barra de navegación**

Creamos un \_barra.scss en la carpeta header y lo incluimos en el index para que se compile

@use 'base/mixins' as m;

.barra {

*display*: flex;

*flex-direction*: column;

    @include m.tablet {

*flex-direction*: row;

*justify-content*: space-between;

*align-items*: center;

     }

    .logo {

*width*: 50%;

*margin*: 0 auto;

        @include m.tablet {

*width*: 20rem;

*margin*: 0;

         }

    }

}

**Ajustando la barra de navegación.**

Ya teníamos el archivo \_navegacion.scss y sobre ese vamos a trabajar

@use 'base/variables' as v;

@use 'base/mixins' as m;

.nav-principal {

*display*: flex;

*flex-direction*: column;

*margin-top*: 5rem;

    @include m.tablet {

*flex-direction*: row;

*gap*: 1rem;

*margin-top*: 0;

     }

    a {

*text-align*: center;

*color*: v.$blanco;

*font-size*: 2rem;

*text-transform*: uppercase;

*font-weight*: 700;

*margin-bottom*: 1rem;

        @include m.tablet {

*margin-bottom*: 0;

         }

         &:hover {

*color*: v.$secundario;

         }

         &.activo {

*color*: v.$secundario;

         }

    }

}

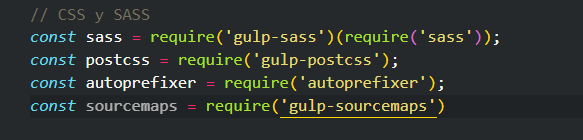
**Finalizando el header**

**Añadir un Sourcemap y sus ventajas**

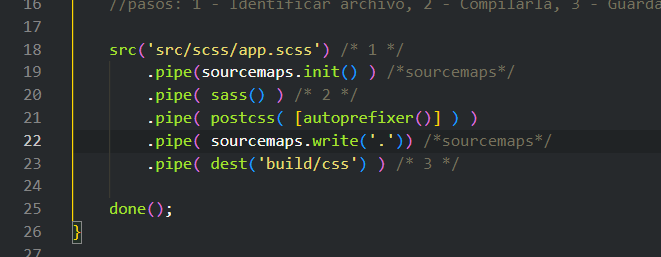
Cuando está hecho el deployment, se carga solo la hoja de estilos app.css, entonces si nos piden un cambio, si, el navegador nos dirá en que parte de la hoja está pero, ¿Cómo saber en cual de todos los archivos sass está esa línea? Para ello es el sourcefile

Se instala la dependencia de desarrollo **npm i –save-dev gulp-sourcemaps**

Creamos una variable en el gulpfile



Y en la function de css



Y con eso al inspeccionar un elemento en el navegador va a dar la ubicación de la línea de código y en que archivo de sass se encuentra.

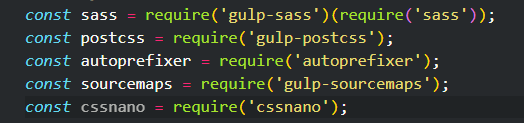
Y se crea en el build/css un archivo app.css.map

**Mejorar tu código con CSS nano**

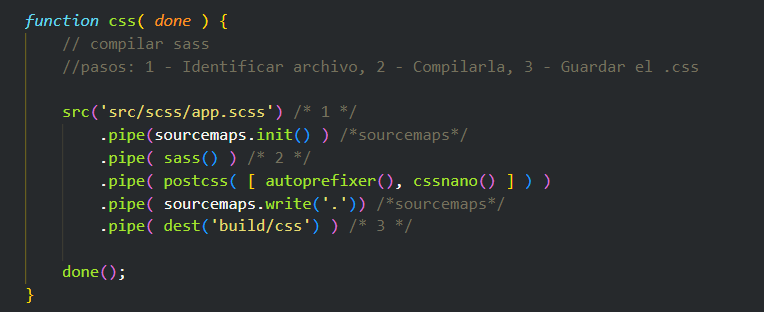
Es una forma de optimizar el código y hacer la hoja de estilos más ligera

Dependencia npm i –save-dev cssnano

Y en el gulpfile le creamos su variable



Y se ejecuta en la función de css junto con autoprefixer





Y al ejecutar el gulp dev, la hoja de estilos se va a comprimir a un tamaño menor al original y lo minifica haciéndolo imposible de leer.

