

EXPLICACIÓN DE FUNCIONAMIENTO SASS

ÍNDICE

¿QUÉ ES SASS?.....	2
Establecer variables.....	2
Anidación más simple	3
Utilizar mixins.....	3
Herencias.....	3
Importar	3
Rapidez.....	3
¿QUÉ ES GULP.js?	3
¿QUÉ ES Node.js?.....	4
¿QUÉ ES NPM?	4
¿QUÉ ES package.json?	5
¿CÓMO crear el primer archivo de SASS?	8
Creación de un watch.....	11
¿CÓMO instalar Gulp?.....	12
¿CÓMO compilar SASS con Gulp?	16
Creación de un watch para gulp.....	20
¿CÓMO UTILIZAR fuentes directamente desde Google fonts?	21
CREACIÓN de variables en SASS.....	23
AGREGAR watchs a todos los archivos	26
El uso de NORMALIZE.....	27
El uso de @use @import @forward	28
Creación y enlazamiento de todos los archivos del proyecto de forma ordenada	29
¿QUE SON LOS Mixins?	31
¿Cómo se crean?	31
¿Cómo se utilizan?	31
¿USANDO Mixins para MEDIA Querys?.....	32
¿CÓMO agregar videos a una web?	34
¿MIXINS para modificar activamente el número de columnas de un GRID?	35
¿Cómo evitar demasiado especificidad en el CSS compilado resultante?	36
¿CÓMO saber al inspeccionar en que archivo SCSS está lo que estás inspeccionando con el cursor?.....	37
¿CÓMO ejecutar varias tareas seguidas, una tras otra?	37

¿CÓMO mejorar el Performance web?	39
Minificar CSS.....	39
Minificar JS	40
Carga diferida de imágenes con LAZY LOADING.....	41
Características en la carga de videos.....	43
video - HTML: Lenguaje de etiquetas de hipertexto MDN (mozilla.org)	43
Carga de imágenes de la galería primero en baja resolución y luego al clicar en alta	44
Utilizar formatos modernos para las imágenes.....	47
Formato AVIF (aún mejor que .webp)	51
¿Cuál debo utilizar AVIF o WebP?	51

¿QUÉ ES SASS?

SASS: Syntactically Awesome StyleSheet

SASS es un CSS con superpoderes. Se le considera un estándar de la industria

SASS es un preprocesador de CSS compatible con todas sus versiones. Por lo tanto, se trata de una herramienta utilizada por los desarrolladores web para traducir un código de hojas de estilo no estándar a un código CSS estándar, legible por la mayoría de los navegadores. La principal utilidad de SASS es la de hacer más simple la escritura del código CSS, además de brindar diversas utilidades que a día de hoy el CSS no puede ofrecer.

Literalmente, sus siglas significan “hoja de estilos sintácticamente impresionante” y sus creadores lo califican como el lenguaje de prolongación CSS de grado profesional más maduro, estable y potente del mundo.

Ventajas de utilizar SASS frente a CSS convencional

Las hojas de estilo de un sitio web cada vez son más complejas y difíciles de mantener. En este punto es dónde un preprocesador de CSS puede ser de gran utilidad y SASS permite emplear funcionalidades que no existen en CSS. Algunas de las principales ventajas del uso de SASS serían:

Establecer variables

Al igual que sucede en cualquier lenguaje de programación, las variables aquí van a permitir guardar información para volver a emplearla cuándo sea preciso. Lo más habitual es establecer variables para colores o fuentes, por ejemplo. Si por alguna razón, hubiese que cambiarlas, solamente sería necesario ir al lugar en el que están declaradas y asignarles un nuevo valor.

Anidación más simple

Usar anidamiento en SASS permite escribir el código CSS con la misma estructura visual que el HTML. De este modo, se simplifica el uso de los selectores y se ofrece a los programadores un formato más visual y jerarquizado para seleccionar elementos anidados.

Utilizar mixins

Se trata de grupos de código que se definen con la norma @ mixin seguida del nombre que quiera darse y que, posteriormente, se pueden emplear con la regla @ include más el nombre que se haya establecido. El mixin es algo comparable a lo que sería una función en otro lenguaje de programación y permiten reutilizar secciones íntegras de código escribiendo funciones con los parámetros que se definan.

Herencias

A través de SASS será posible unificar declaraciones, otorgándoles un nombre que irá siempre precedido del signo del tanto por ciento (%). Después, a través de la regla @ extend seguida del nombre de la declaración, será posible emplear esas órdenes. Su peculiaridad es que no se procesan si no se utilizan, por ello, el código procesado estará totalmente libre de declaraciones no empleadas.

Importar

A través de la norma @ import es posible dividir el código CSS en diferentes ficheros y esto constituye una ventaja, puesto que hace posible tener el código distribuido en varios ficheros y luego generar un solo CSS.

Rapidez

Empleando SASS los desarrolladores ahorran mucho tiempo porque es posible acortar el código que deben escribir, así como el número de ficheros a implementar. Aumenta, por lo tanto, la productividad.

¿QUÉ ES GULP.js?

Gulp es una herramienta, en forma de script en NodeJS, que te ayuda a automatizar tareas comunes en el desarrollo de una aplicación, como pueden ser:

- Comprimir SASS y JS.
- Crear imágenes ligeras.
- Minificar código para producción.

A nivel general, mejora la performance de un sitio web.

Gulp utiliza javascript por lo que es necesario controlarlo bien.

Para poder usar gulp hace falta: [Node.js](#) y [NPM](#)

¿QUÉ ES Node.js?

Node.js, que es un entorno de ejecución que incluye todo lo necesario para ejecutar un programa escrito en JavaScript.

Node.js es un entorno de ejecución de un solo hilo, de código abierto y multiplataforma para crear aplicaciones de red y del lado del servidor rápidas y escalables. Se ejecuta en el motor de ejecución de JavaScript V8, y utiliza una arquitectura de E/S basada en eventos y sin bloqueos, lo que la hace eficiente y adecuada para aplicaciones en tiempo real.

¿QUÉ ES NPM?

npm es parte esencial de Node.js, el entorno de ejecución de JavaScript en el lado del servidor basado en el motor V8 de Google. Es muy seguramente la principal razón del gran éxito de Node permitiendo que cientos de desarrolladores puedan compartir paquetes de software entre distintos proyectos.

npm responde a las siglas de Node Package Manager o manejador de paquetes de node, es la herramienta por defecto de JavaScript para la tarea de compartir e instalar paquetes.

Tal como reza su documentación, npm se compone de al menos dos partes principales.

- Un repositorio online para publicar paquetes de software libre para ser utilizados en proyectos Node.js
- Una herramienta para la terminal (command line utility) para interactuar con dicho repositorio que te ayuda a la instalación de utilidades, manejo de dependencias y la publicación de paquetes.

Es decir, en tu proyecto basado en Node — que actualmente incluye los proyectos de aplicaciones web que utilizan Node para su proceso de compilación y generación de archivos — utilizarás la utilidad de línea de comandos (cli) para consumir paquetes desde el repositorio online, un listado gigantesco de soluciones de software para distintos problemas disponibles públicamente en npmjs.com y para manejar dependencias, y para ello necesitas un archivo de configuración que le diga a npm que este es un proyecto node.

¿QUÉ ES package.json?

El package.json es un archivo fundamental en proyectos basados en Node.js y se utiliza para almacenar metadatos relevantes sobre el proyecto. Este archivo se encuentra en la raíz del proyecto y cumple varias funciones importantes.

Funciones package.json

Gestión de dependencias: El package.json enumera todas las dependencias del proyecto, tanto las de producción (dependencies) como las de desarrollo (devDependencies). Esto permite que cualquiera que clone el proyecto pueda instalarlas fácilmente con npm install.

Scripts: Permite definir scripts personalizados bajo la clave scripts que se pueden ejecutar con npm run <script-name>. Estos scripts pueden incluir comandos para iniciar el servidor, compilar el código, ejecutar pruebas, y más. Esto proporciona una manera conveniente de automatizar tareas comunes del proyecto.

Proceso:

Número	Proceso	Cómo se hace
1	Crear el archivo package.json del proyecto	Npm init (a continuación te irá haciendo preguntas para registrar los datos en el archivo).

Una vez está instalado el archivo package.json veremos como instalar dependencias en él.

Package.json es importante porque va a almacenar que dependencias requiere un proyecto, pero también sus versiones, de esta forma, en un equipo de trabajo se puede compartir este package.json y los desarrolladores van a saber que versiones tienen que instalar y cuáles se requieren para que el proyecto funcione correctamente.

Ahora abrimos el terminal e instalaremos la primera dependencia, la herramienta que utilizaremos será npm (Node Package Manager):

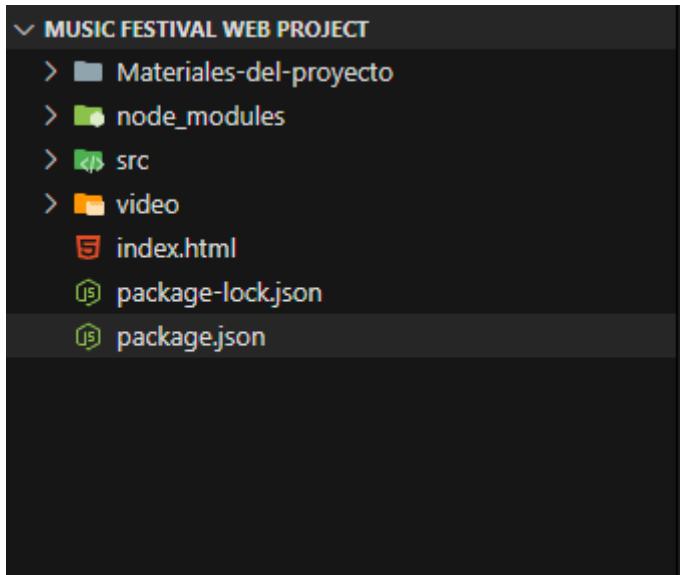
Número	Proceso	Cómo se hace
2	Instalar una dependencia que necesitamos.	Npm install (nombre_dependencia) Ó Npm i (nombre_dependencia) Para este caso instalaremos la dependencia sass:

Npm i sass

Para buscar que tipo de dependencia queremos instalar podemos ir a la web: [npm | Home \(npmjs.com\)](https://www.npmjs.com)

The screenshot shows two screenshots of the npm website. The top screenshot is a search results page for 'sass' with several packages listed: 'sass' (exact match), 'sass-loader', 'node-sass', and 'sass-embedded'. The bottom screenshot is a detailed view for the 'sass' package, showing its version (1.77.6), dependencies (3), dependents (10,090), and 254 versions. It includes sections for Readme, Code, Dependencies, Install (with a command line example), Repository (github.com/sass/dart-sass), Homepage (github.com/sass/dart-sass), Weekly Downloads (13,350.373), Version (1.77.6), License (MIT), Unpacked Size (5.22 MB), Total Files (36), Issues (74), and Pull Requests (6).

Después de instalar la dependencia sass notarás varias diferencias:



Se ha creado una carpeta llamada `node_modules` que almacenará todas las dependencias. En un proyecto normal esta carpeta llega a pesar 200 – 300 Mb.

Normalmente la carpeta `node_modules` no se sube a un servidor debido a su peso, en este caso por ejemplo que estamos usando la dependencia sass, no es necesario ya que se compilará el sass en tu ordenador en local para generar un css que es el que finalmente será utilizado. Por esto sass no se requiere como una “dependencia del proyecto” sino que se le conocerá como una “dependencia de desarrollo del proyecto”. Por defecto se pondrá como dependencia del proyecto, pero para cambiarla a dependencia de desarrollo del proyecto, tendrás que:

Número	Proceso	Cómo se hace
2	Instalar una dependencia que necesitamos.	<code>npm i --save-dev (nombre de la dependencia sin ())</code> <code>npm i --save-dev sass</code> ó <code>npm i -D sass</code>

Después de hacer esto verás que en el archivo `package.json` está ahora como “`devDependencies`” y no como “`dependencias`” que es como estaba anteriormente. Se sigue almacenando en `node_modules`.

Existe otro tipo de dependencias

El archivo `package-lock.json` almacenará las dependencias necesarias para el propio funcionamiento de las dependencias, por lo que NO TOCAR.

VENTAJA de todo esto:

En el caso en el que se elimine la carpeta `node_modules` para subir a un repositorio o transferir a otro compañero de trabajo (se eliminaría porque es un archivo pesado e innecesario de

transferir), el receptor podrá reconstruir el archivo node_modules con los comandos (esto pasará si tienen intactos los archivos package-lock.json y package.json):

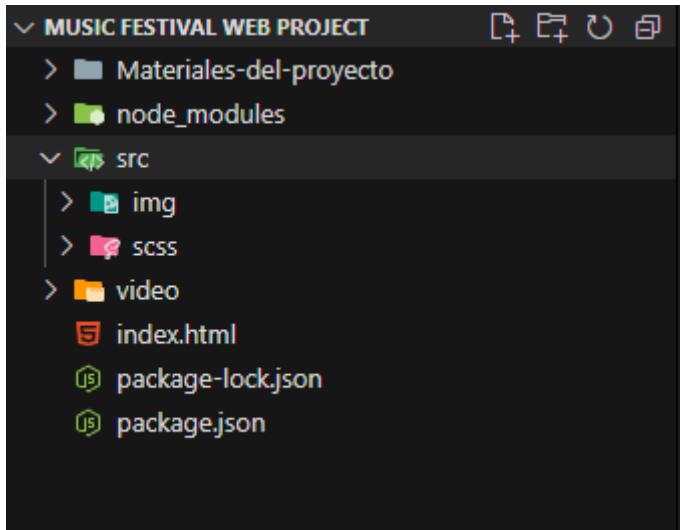
Número	Proceso	Cómo se hace
2	Reinstalar la carpeta node_modules con todas las dependencias necesarias para nuestro proyecto.	npm i

Para resumir, los tres tipos de dependencias que existen son:

1. Dependencias del proyecto (Project Dependencies):
 - Estas son las bibliotecas y herramientas necesarias para que la aplicación funcione en el entorno de producción.
 - Ejemplo: Una base de datos, una biblioteca de autenticación.
2. Dependencias de desarrollo (Development Dependencies):
 - Estas son las herramientas y bibliotecas necesarias solo durante el desarrollo y las pruebas del proyecto.
 - Ejemplo: Herramientas de prueba, linters, compiladores.
3. Dependencias de las dependencias (Transitive Dependencies):
 - Estas son dependencias requeridas por las bibliotecas que el proyecto utiliza directamente.
 - Ejemplo: Si una biblioteca A depende de una biblioteca B, entonces B es una dependencia transitiva de tu proyecto.

¿CÓMO crear el primer archivo de SASS?

Para empezar y respetar las convenciones, tendrás que crear una nueva carpeta dentro de "src" llamada "scss".



Y a su vez, dentro crearemos el archivo “app.scss”. Si vincularas a un .html este archivo .scss no marcaría ningún error, pero no funcionaría porque es necesario compilarlo, y para esto tendrás que ir al “package.json” e ir a la sección “scripts” y ahí deberás modificarlo para mandar a llamar al compilador que ya existe dentro de node_modules bin.

```

{
  "name": "music-festival-web-project",
  "version": "1.0.0",
  "description": "Aprendiendo SASS y NPM",
  "main": "index.js",
  "scripts": {
    "sass": "sass src/scss:dist/css"
  },
  "keywords": [
    "SASS",
    "NPM",
    "Gulp"
  ],
  "author": "Derimán Tejera Fumero",
  "license": "ISC",
  "devDependencies": {
    "sass": "^1.77.6"
  }
}

```

Explicación paso a paso del script “sass”:

- sass: Este es el compilador de Sass (requiere que tengas Sass instalado).
- src/scss: Es la carpeta de origen donde están ubicados tus archivos SCSS.
- dist/css: Es la carpeta de destino donde se guardarán los archivos CSS compilados.

Para realizar este ejemplo se ha creado un código sencillo en app.scss:

The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER**: Shows the project structure under "MUSIC FESTIVAL WEB PROJECT". It includes:
 - dist
 - Materiales-del-proyecto
 - node_modules
 - .bin
 - sass
 - sass.cmd
 - sass.ps1
 - anymatch
 - binary-extensions
 - braces
 - chokidar
 - fill-range
 - glob-parent
 - immutable
 - is-binary-path
 - is-extglob
 - is-glob
 - is-number
 - normalize-path
 - picomatch
 - readdirp
 - sass
 - source-map-js
 - to-regex-range
 - .package-lock.json
- src

- EDITOR**: The file "app.scss" is open, showing the following code:

```
1 body {
2   background-color: red;
3 }
```
- TERMINAL**: Shows the command being run in the terminal:

```
PS C:\Users\Usuario\Desktop\Music festival web project> npm run sass
> music-festival-web-project@1.0.0 sass
> sass src/scss:dist/css
```

En la imagen de arriba puede verse como se ha llamado al script sass para complilar el archivo y generar un archivo .css que si pueda ser usado por el navegador (ten en cuenta que tu .html debe estar vinculado con este archivo).

Para ejecutar todos los script contenido en la sección “scripts” de package.json se hará mediante el comando:

Número	Proceso	Cómo se hace
1	Ejecutar scripts de package.json	Npm run (nombre del script) En este caso: Npm run sass

En muchos casos en lugar de una carpeta con el nombre “dist” donde almacenar el resultado de la compilación, se llame “build”.

```

{
  "name": "music-festival-web-project",
  "version": "1.0.0",
  "description": "Aprendiendo SASS y NPM",
  "main": "index.js",
  "scripts": {
    "sass": "sass src/scss:build/css"
  },
  "keywords": [
    "SASS"
  ]
}

```

PS C:\Users\Usuario\Desktop\Music festival web project> npm run sass
> music-festival-web-project@1.0.0 sass
> sass src/scss:dist/css
PS C:\Users\Usuario\Desktop\Music festival web project> npm run sass
> music-festival-web-project@1.0.0 sass
> sass src/scss:build/css
PS C:\Users\Usuario\Desktop\Music festival web project>

Se ha compilado y ha aparecido el archivo app.css para poder ser utilizado.

```

<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Festival de Música</title>
    <link rel="stylesheet" href="build/css/app.css" /> | ←
  </head>
  <body>
    <h1>Hola Mundo</h1>
  </body>
</html>

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Usuario\Desktop\Music festival web project> npm run sass
> music-festival-web-project@1.0.0 sass
> sass src/scss:dist/css
PS C:\Users\Usuario\Desktop\Music festival web project> npm run sass
> music-festival-web-project@1.0.0 sass
> sass src/scss:build/css
PS C:\Users\Usuario\Desktop\Music festival web project>

Cualquier modificación de los estilos NO PUEDES hacerla desde el .css, sino desde el .scss y luego compilarlo, de lo contrario corres riesgo de sobrescribir todo lo que hayas hecho en el .css o que lo haga otra persona al usar el .scss

Creación de un watch

SI, NO ES EFICIENTE QUE CADA VEZ QUE HAGAS CAMBIOS EN EL ESTILO, PARA PODER VERLOS REFLEJADOS TENGAS QUE COMPILAR CONTINUAMENTE, PARA RESOLVERLO:

```

1 {
2   "name": "music-festival-web-project",
3   "version": "1.0.0",
4   "description": "Aprendiendo SASS y NPM",
5   "main": "index.js",
6   "scripts": {
7     "sass": "sass --watch src/scss:build/css" ←
8   },
9   "keywords": [
10     "SASS",
11     "NPM",
12     "Gulp"
13 ],
14   "author": "Derimán Tejera Fumero",
15   "license": "ISC",
16   "devDependencies": {
17     "sass": "1.77.6"
18   }
19 }
20

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Usuario\Desktop\Music festival web project> **npm run sass**

> music-festival-web-project@1.0.0 sass

> sass --watch src/scss:build/css

Sass is watching for changes. Press Ctrl-C to stop.

Con esto se crea un watch que permanecerá a la espera de cambios (pero antes tienes que ejecutarlo), lo malo es que bloqueará la consola hasta que pares tu la ejecución, o crees otra pestaña de terminal.

¿CÓMO instalar Gulp?

Número	Proceso	Cómo se hace
1	Instalar Gulp	Npm i –save-dev gulp

Si te has equivocado en el comando y se instala como Dependencies y no como devDependencies, entonces usa el comando de nuevo con el nombre de la dependencia, si ya existe, la cambiará de tipo de dependencia:

Número	Proceso	Cómo se hace
1	Instalar Gulp	Npm i –save-dev gulp

Ahora, en lugar de tener muchos scripts que ejecutar en package.json

The screenshot shows the VS Code interface with the project tree on the left and the package.json file open in the editor on the right. The project tree includes files like index.html, package-lock.json, package.json, gulpfile.js, app.scss, and src/scss. The package.json file contains the following JSON code:

```
1 {  
2   "name": "music-festival-web-project",  
3   "version": "1.0.0",  
4   "description": "Aprendiendo SASS y NPM",  
5   "main": "index.js",  
6   "scripts": {  
7     "sass": "sass --watch src/scss:build/css"  
8   },  
9   "keywords": [  
10    "SASS",  
11    "NPM",  
12    "Gulp"  
13  ],  
14  "author": "Derimán Tejera Fumero",  
15  "license": "ISC",  
16  "devDependencies": {  
17    "gulp": "^5.0.0",  
18    "sass": "1.77.6"  
19  }  
21 }
```

Two red arrows point from the text "gulpfile.js" in the first section of the text to the "gulp" entry in the "devDependencies" object of the package.json file.

En lugar de esto vamos a crear un nuevo archivo llamado gulpfile.js que las concentrará todas, en el caso del contenido de este archivo, lo que se hará es crear funciones.

The screenshot shows the VS Code interface with the file gulpfile.js open. The code in the editor is:

```
1 export function hola() {  
2   console.log("Hola desde Gulpfile.js");  
3 }  
4 |
```

En este caso se ha creado una función simple de mensaje por consola, a la que se le ha añadido “export” para poder llamarla desde package.json. Ahora lo añadimos a package.json para luego poder ejecutar el script desde consola:

```
1  {
2    "name": "music-festival-web-project",
3    "version": "1.0.0",
4    "description": "Aprendiendo SASS y NPM",
5    "main": "index.js",
6    "scripts": {
7      "sass": "sass --watch src/scss:build/css",
8      "hola": "gulp hola"
9    },
10   "keywords": [
11     "SASS",
12     "NPM",
13     "Gulp"
14   ],
15   "author": "Derimán Tejera Fumero",
16   "license": "ISC",
17   "devDependencies": {
18     "gulp": "^5.0.0",
19     "sass": "^1.77.6"
20   }
21 }
```

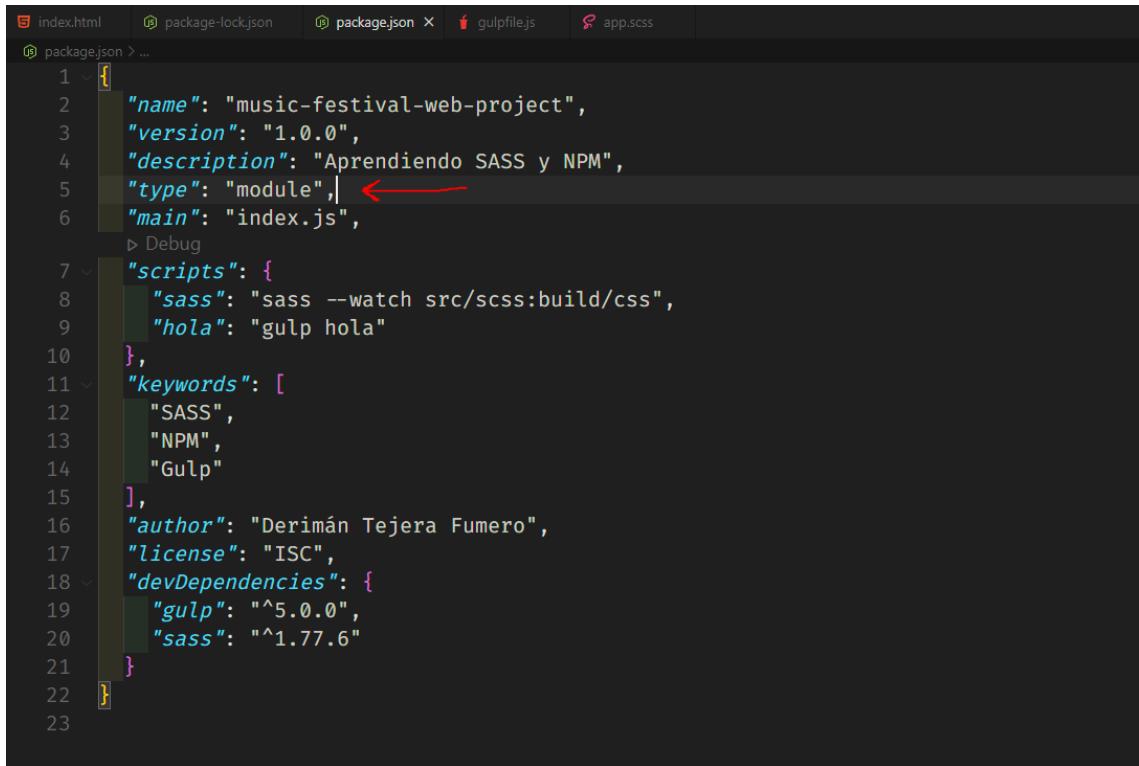
Y ejecutamos con (npm run hola):

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Usuario\Desktop\Music festival web project> npm i --seve-dev gulp
PS C:\Users\Usuario\Desktop\Music festival web project> npm run hola
> music-festival-web-project@1.0.0 hola
> gulp hola

C:\Users\Usuario\Desktop\Music festival web project\gulpfile.js:1
export function hola() {
^^^^^

SyntaxError: Unexpected token 'export'
  at internalCompileFunction (node:internal/vm:128:18)
  at wrapSafe (node:internal/modules/cjs/loader:1280:20)
  at Module._compile (node:internal/modules/cjs/loader:1332:27)
  at Module._extensions..js (node:internal/modules/cjs/loader:1427:10)
  at Module.load (node:internal/modules/cjs/loader:1206:32)
  at Module._load (node:internal/modules/cjs/loader:1022:12)
```

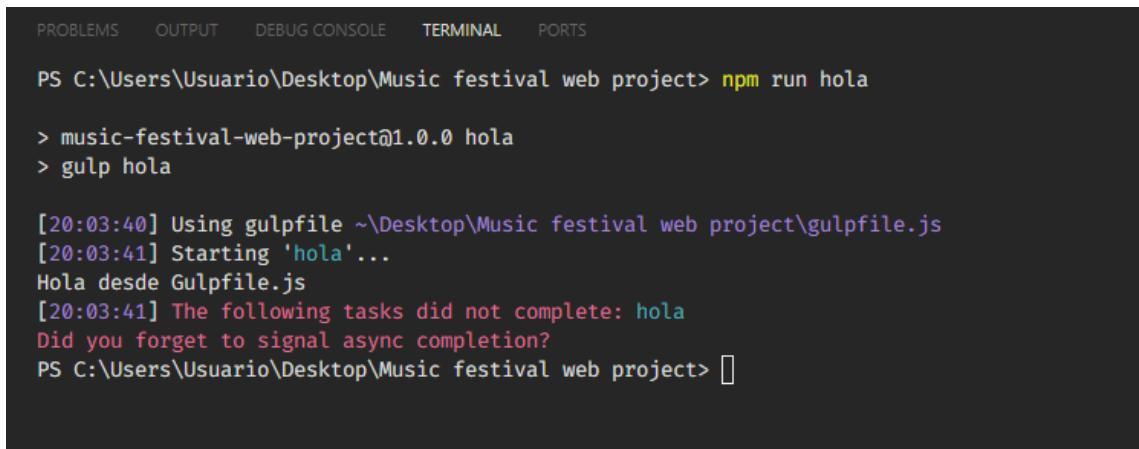
Como verás, también ha generado un **error** al no reconocer “export”. Para que lo reconozcas deberás añadir la siguiente línea:



```
index.html package-lock.json package.json gulpfile.js app.scss
package.json > ...
1 {
2   "name": "music-festival-web-project",
3   "version": "1.0.0",
4   "description": "Aprendiendo SASS y NPM",
5   "type": "module",| ←
6   "main": "index.js",
7   > Debug
8   "scripts": {
9     "sass": "sass --watch src/scss:build/css",
10    "hola": "gulp hola"
11  },
12  "keywords": [
13    "SASS",
14    "NPM",
15    "Gulp"
16  ],
17  "author": "Derimán Tejera Fumero",
18  "license": "ISC",
19  "devDependencies": {
20    "gulp": "^5.0.0",
21    "sass": "^1.77.6"
22  }
23 }
```

Así se especificará que debe usar la sintaxis moderna, ya que por default usará: commonjs que no reconocerá export como ya vistes.

Una vez hechos estos cambios, volvemos a ejecutar el comando:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Usuario\Desktop\Music festival web project> npm run hola
> music-festival-web-project@1.0.0 hola
> gulp hola
[20:03:40] Using gulpfile ~\Desktop\Music festival web project\gulpfile.js
[20:03:41] Starting 'hola'...
Hola desde Gulpfile.js
[20:03:41] The following tasks did not complete: hola
Did you forget to signal async completion?
PS C:\Users\Usuario\Desktop\Music festival web project> []
```

Ahora se ha ejecutado correctamente, mostrando por consola el mensaje de la función, pero como verás, no considera completa la tarea, para ello habrá que avisar desde la función cuándo se ha completado:

```
index.html package-lock.json package.json gulpfile.js app.scss  
gulpfile.js > ...  
1 export function hola(done) {  
2   console.log("Hola desde Gulpfile.js");  
3  
4   done();  
5 }  
6
```

Ejecutamos otra vez:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
PS C:\Users\Usuario\Desktop\Music festival web project> npm run hola  
> music-festival-web-project@1.0.0 hola  
> gulp hola  
[20:06:15] Using gulpfile ~\Desktop\Music festival web project\gulpfile.js  
[20:06:15] Starting 'hola'...  
Hola desde Gulpfile.js  
[20:06:15] Finished 'hola' after 2.22 ms  
PS C:\Users\Usuario\Desktop\Music festival web project>
```

Ahora Gulp sabe que la tarea ha finalizado.

¿CÓMO compilar SASS con Gulp?

Instalamos la dependencia:

Número	Proceso	Cómo se hace
1	Instalar la dependencia gulp-sass	Npm i -save-dev gulp-sass

```
1  {
2    "name": "music-festival-web-project",
3    "version": "1.0.0",
4    "description": "Aprendiendo SASS y NPM",
5    "type": "module",
6    "main": "index.js",
7    > Debug
8    "scripts": {
9      "sass": "sass --watch src/scss:build/css",
10     "hola": "gulp hola"
11   },
12   "keywords": [
13     "SASS",
14     "NPM",
15     "Gulp"
16   ],
17   "author": "Derimán Tejera Fumero",
18   "license": "ISC",
19   "devDependencies": {
20     "gulp": "^5.0.0",
21     "gulp-sass": "^5.1.0", ←
22     "sass": "^1.77.6"
23   }
24 }
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Usuarlo\Desktop\Music festival web project> npm i --save-dev gulp-sass
added 14 packages, and audited 161 packages in 3s
14 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\Usuarlo\Desktop\Music festival web project> []
```

Ahora para poder utilizar estas funciones deberás modificar gulpfile.js

A screenshot of a code editor showing a Gulpfile.js file. The file contains the following code:

```
1 import gulpSass from "gulp-sass";
2
3 export function hola(done) {
4   console.log("Hola desde Gulpfile.js");
5
6   done();
7 }
8
```

Ahora vamos a indicar que quieres usar sass, pero usando la dependencia de gulp-sass:

A screenshot of a code editor showing a Gulpfile.js file. The file contains the following code:

```
1 import * as dartSass from "sass";
2 import gulpSass from "gulp-sass";
3
4 const sass = gulpSass(dartSass);
5
```

A screenshot of a code editor showing a Gulpfile.js file. The file contains the following code:

```
1 import * as dartSass from "sass";
2 import gulpSass from "gulp-sass";
3
4 const sass = gulpSass(dartSass);
5
6 export function css(done) {
7   done();
8 }
```

En la anterior imagen ya podemos ver como package.json y gulpfile.js están correctamente comunicados.

Ahora incluiremos pipes para poder ir controlando la ejecución de las diferentes funciones (los pipes te dan el control sobre el orden en el que se irán ejecutando):

```
index.html package-lock.json package.json gulpfile.js app.scss
gulpfile.js > ...
1 import { src, dest } from "gulp";
2 import * as dartSass from "sass";
3 import gulpSass from "gulp-sass";
4
5 const sass = gulpSass(dartSass);
6
7 export function css(done) {
8   src("src/scss/app.scss")
9     .pipe(sass())
10    .pipe(dest("build/css"));
11  done();
12}
13|
```

Lo anterior es equivalente a:

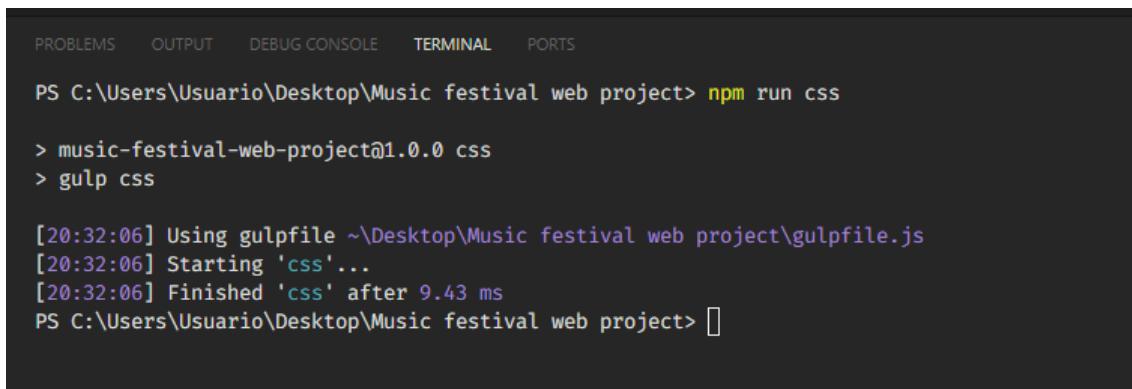
```
index.html package-lock.json package.json gulpfile.js app.scss
packagejson > {} scripts
1 {
2   "name": "music-festival-web-project",
3   "version": "1.0.0",
4   "description": "Aprendiendo SASS y NPM",
5   "type": "module",
6   "main": "index.js",
7   "scripts": {
8     "sass": "sass --watch src/scss:build/css",
9     "css": "gulp css"
10   },
11   "keywords": [
12     "SASS",
13     "NPM",
14     "Gulp"
15   ],
16   "author": "Derimán Tejera Fumero",
17   "license": "ISC",
18   "devDependencies": {
19     "gulp": "^5.0.0",
20     "gulp-sass": "^5.1.0",
21     "sass": "^1.77.6"
22   }
23 }
```

```

1 ~ {
2   "name": "music-festival-web-project",
3   "version": "1.0.0",
4   "description": "Aprendiendo SASS y NPM",
5   "type": "module",
6   "main": "index.js",
7   > Debug
7 ~ "scripts": {
8   "sass": "sass --watch src/scss:build/css",
9   "css": "gulp css" ←
10  },
11 ~ "keywords": [
12   "SASS".

```

Ahora borramos la carpeta build y todo su contenido. Y a continuación ejecutamos run, lo que logra volver a crear la carpeta build:



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

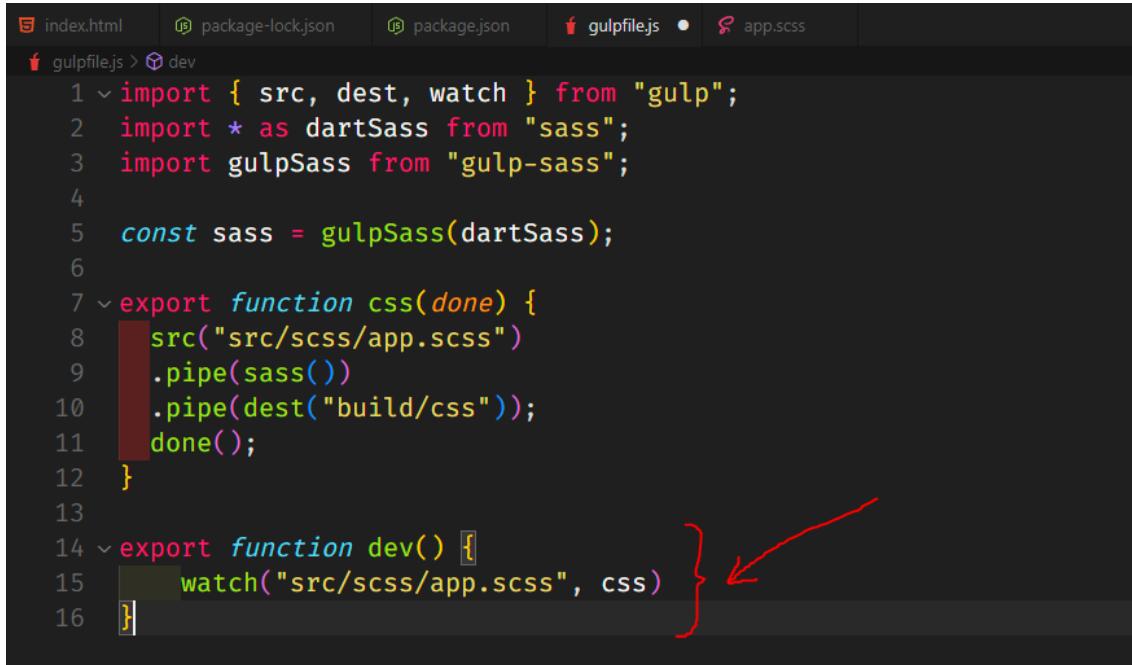
PS C:\Users\Usuario\Desktop\Music festival web project> npm run css

> music-festival-web-project@1.0.0 css
> gulp css

[20:32:06] Using gulpfile ~\Desktop\Music festival web project\gulpfile.js
[20:32:06] Starting 'css'...
[20:32:06] Finished 'css' after 9.43 ms
PS C:\Users\Usuario\Desktop\Music festival web project> []

```

Creación de un watch para gulp



```

index.html package-lock.json package.json gulpfilejs app.scss

gulpfilejs > ⚡ dev
1 ~ import { src, dest, watch } from "gulp";
2 import * as dartSass from "sass";
3 import gulpSass from "gulp-sass";
4
5 const sass = gulpSass(dartSass);
6
7 ~ export function css(done) {
8   src("src/scss/app.scss")
9     .pipe(sass())
10    .pipe(dest("build/css"));
11    done();
12 }
13
14 ~ export function dev() {
15   watch("src/scss/app.scss", css) } ←
16 }

```

```
1  {
2    "name": "music-festival-web-project",
3    "version": "1.0.0",
4    "description": "Aprendiendo SASS y NPM",
5    "type": "module",
6    "main": "index.js",
7    "scripts": [
8      "sass": "sass --watch src/scss:build/css",
9      "dev": "gulp dev"
10    ],
11    "keywords": [
12      "SASS",
13      "NPM",
14      "Gulp"
15    ],
16    "author": "Derimán Tejera Fumero",
17    "license": "ISC",
18    "devDependencies": {
19      "gulp": "^5.0.0",
20      "gulp-sass": "^5.1.0",
21      "sass": "^1.77.6"
22    }
23  }
24
```

Ahora estará activado el watch a la espera de cambios:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Usuario\Desktop\Music festival web project> npm run dev

> music-festival-web-project@1.0.0 dev
> gulp dev

[20:37:30] Using gulpfile ~\Desktop\Music festival web project\gulpfile.js
[20:37:30] Starting 'dev'...
```

Para cancelarlo usar CTRL+C

¿CÓMO UTILIZAR fuentes directamente desde Google fonts?

Lo primero es ir a la web Google Fonts y buscar la fuente Montserrat

Google Fonts

Specimen Type tester Glyphs About & license Remove font family

Montserrat

Designed by Julieta Ulanovsky, Sol Matas, Juan Pablo del Peral, Jacques Le Bailly

Whereas disregard and contempt for human rights have resulted

Select preview text: Continent Language

Styles

Type here to preview text 48px

Google Fonts

Specimen Type tester Glyphs About & license

Montserrat

Designed by Julieta Ulanovsky, Sol Matas, Juan Pablo del Peral, Jacques Le Bailly

Google Fonts

1 font family selected

Montserrat Variable

Everyone has the right to freedom

Share Remove all Get embed code Download all (1)

Google Fonts

← Embed code

Montserrat Variable

Whereas recognition

Italic Full axis One value
0 - 1

Weight Full axis One value
100 - 900
2 styles
Italic: 0 Weight: 100 - 900

Web Android iOS Flutter

Embed code in the <head> of your html

```
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,100..900&display=swap" rel="stylesheet">
```

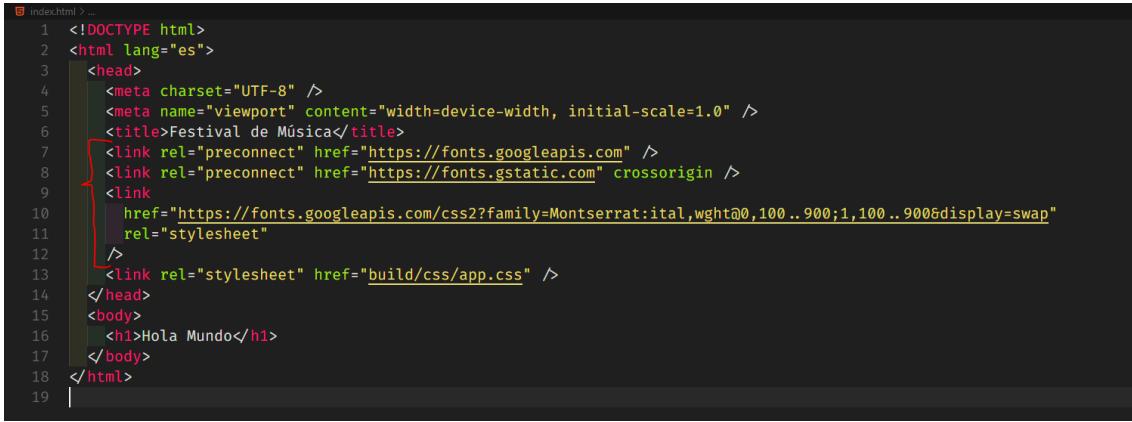
Copy code

Montserrat: CSS class for a variable style

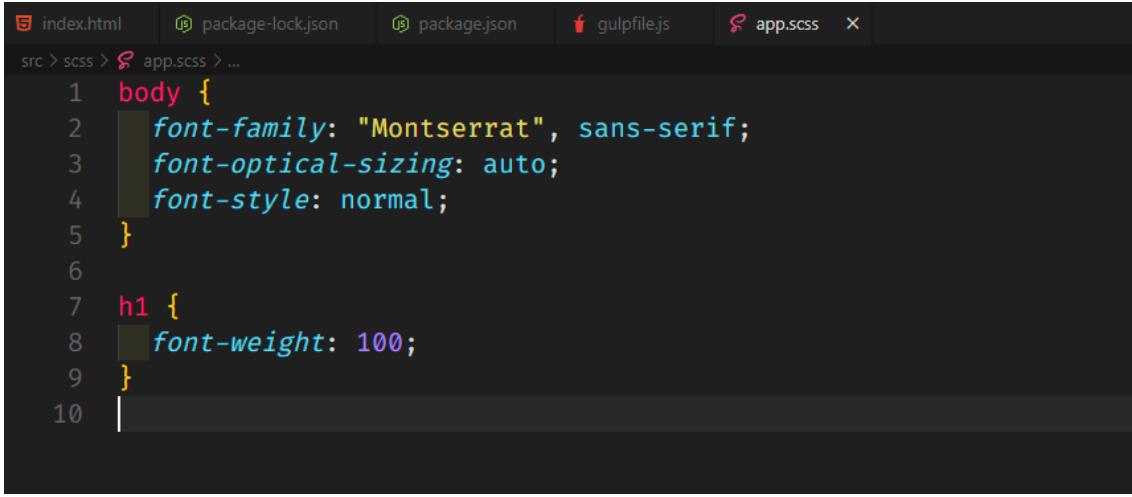
```
// <uniqueifier>: Use a unique and descriptive class name
// <weight>: Use a value from 100 to 900
.montserrat-<uniqueifier> {
  font-family: "Montserrat", sans-serif;
  font-optical-sizing: auto;
  font-style: normal;
}
```

Copy code

HEAD AL CSS



```
index.html > ...
1  <!DOCTYPE html>
2  <html lang="es">
3    <head>
4      <meta charset="UTF-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6      <title>Festival de Música</title>
7      <link rel="preconnect" href="https://fonts.googleapis.com" />
8      <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
9      <link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,100..900;1,100..900&display=swap" rel="stylesheet"
10     />
11     <link rel="stylesheet" href="build/css/app.css" />
12   </head>
13   <body>
14     <h1>Hola Mundo</h1>
15   </body>
16 </html>
17
18
19
```



```
index.html package-lock.json package.json gulpfile.js app.scss
src > scss > app.scss > ...
1  body {
2    font-family: "Montserrat", sans-serif;
3    font-optical-sizing: auto;
4    font-style: normal;
5  }
6
7  h1 {
8    font-weight: 100;
9  }
10
```

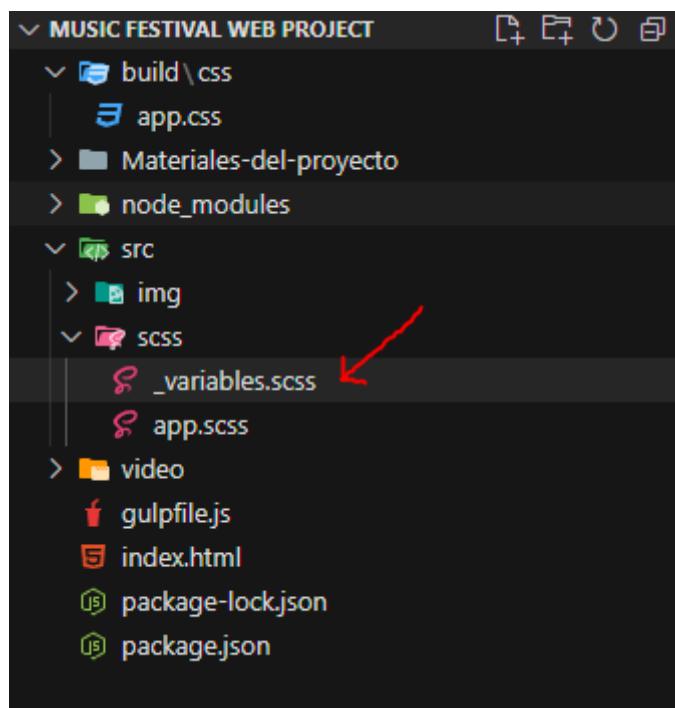
CREACIÓN de variables en SASS

```

src > scss > app.scss > ...
1 $Fuente_principal: "Montserrat", sans-serif;
2
3 body {
4   font-family: "Montserrat", sans-serif;
5   font-optical-sizing: auto;
6   font-style: normal;
7 }
8
9 h1 {
10   font-family: $Fuente_principal;
11 }
12
13 p {
14   font-family: $Fuente_principal;
15 }
16
17 a {
18   font-family: $Fuente_principal;
19 }
20

```

Una de las propiedades de SASS es que permite dividir la hoja de estilos en diferentes partes, para ello tendrás que crear un nuevo archivo iniciando con barra-baja _ para que interprete correctamente que es una subparte de un archivo .scss mayor:



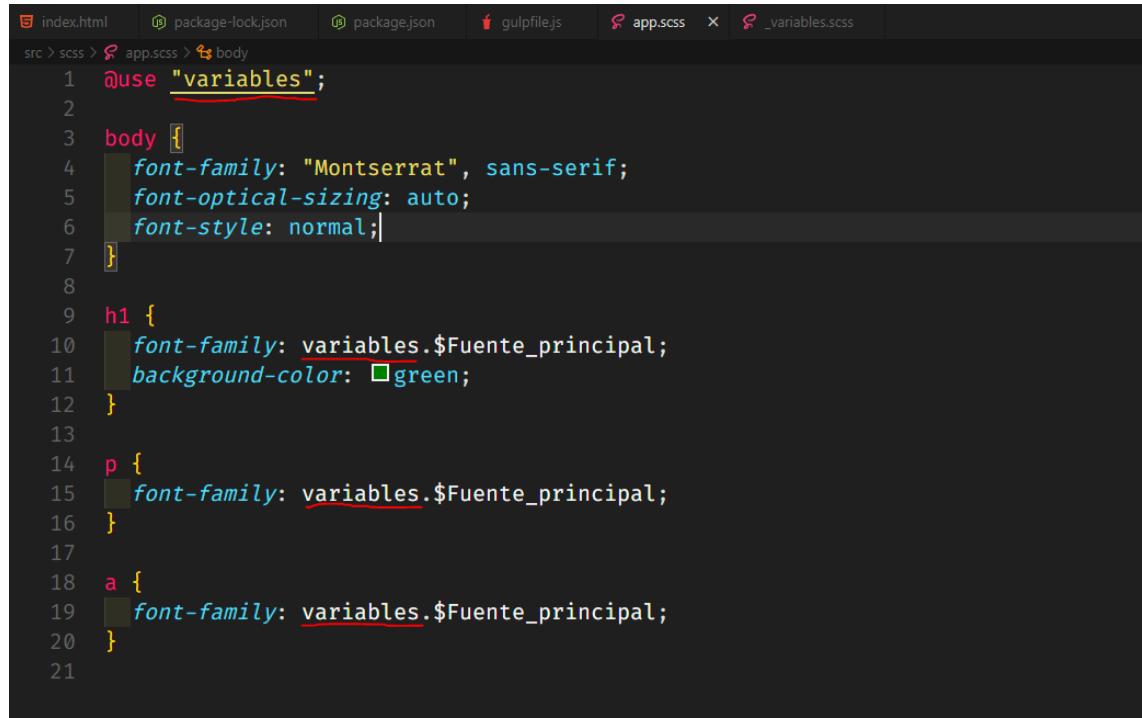
El contenido de _variables.scss:



```
index.html package-lock.json package.json gulpfile.js app.scss _variables.scss

src > scss > _variables.scss > ...
1 $Fuente_principal: "Montserrat", sans-serif;
```

Para poder usarlo desde el archivo principal de scss que utilizar @use “nombre del archivo sin _, en este caso: @use “variables”;



```
index.html package-lock.json package.json gulpfile.js app.scss x _variables.scss

src > scss > app.scss > body
1 @use "variables";
2
3 body {
4   font-family: "Montserrat", sans-serif;
5   font-optical-sizing: auto;
6   font-style: normal;
7 }
8
9 h1 {
10   font-family: variables.$Fuente_principal;
11   background-color: green;
12 }
13
14 p {
15   font-family: variables.$Fuente_principal;
16 }
17
18 a {
19   font-family: variables.$Fuente_principal;
20 }
```

Para poder utilizarlo sin generar errores, debes usar el nombre del archivo indicado arriba, variable. Seguido de la variable, pero esto se puede evitar utilizando:

```
index.html package-lock.json package.json gulpfile.js app.scss _variables.scss
src > scss > app.scss > ...
1 @use "variables" as *;
2
3 body {
4   font-family: "Montserrat", sans-serif;
5   font-optical-sizing: auto;
6   font-style: normal;
7 }
8
9 h1 {
10   font-family: $Fuente_principal;
11   background-color: green;
12 }
13
14 p {
15   font-family: $Fuente_principal;
16 }
17
18 a {
19   font-family: $Fuente_principal;
20 }
```

Pero es una buena práctica la de utilizar un nombre recordado por ejemplo v para las variables:

```
index.html package-lock.json package.json gulpfile.js app.scss _variables.scss
src > scss > app.scss > ...
1 @use "variables" as v;
2
3 body {
4   font-family: "Montserrat", sans-serif;
5   font-optical-sizing: auto;
6   font-style: normal;
7 }
8
9 h1 {
10   font-family: v.$Fuente_principal;
11   background-color: green;
12 }
13
14 p {
15   font-family: v.$Fuente_principal;
16 }
17
18 a {
19   font-family: v.$Fuente_principal;
20 }
```

AGREGAR watchs a todos los archivos

Hasta este momento, las modificaciones realizadas únicamente por ejemplo sobre `_variables`, no se están aplicando inmediatamente sobre el archivo .css

```
index.html package-lock.json package.json gulpfile.js app.scss _variables.scss app.css

1 import { src, dest, watch } from "gulp";
2 import * as dartSass from "sass";
3 import gulpSass from "gulp-sass";
4
5 const sass = gulpSass(dartSass);
6
7 export function css(done) {
8     src("src/scss/app.scss")
9         .pipe(sass().on("error", sass.logError))
10        .pipe(dest("build/css"));
11    done();
12 }
13
14 export function dev() {
15     watch("src/scss/app.scss", css);
16     watch("src/scss/**/*.{scss,sass}", css);
17 }
```

The code editor shows a Gulpfile.js script. Handwritten annotations are present: a red 'X' is next to the original 'css' export, and a blue checkmark is next to the new 'dev' export. Red arrows point from the handwritten text to the corresponding code lines.

El uso de NORMALIZE

Utilizar: [Normalize.css: Make browsers render all elements more consistently.](https://necolas.github.io/normalize.css/)
[\(necolas.github.io\)](https://necolas.github.io/normalize.css/)

Ó

Utilizar: [sindresorhus/modern-normalize: 🎨 Normalize browsers' default style \(github.com\)](https://github.com/sindresorhus/modern-normalize)

The code editor shows the file structure of a project named 'MUSIC FESTIVAL WEB PROJECT'. In the 'src' directory, there is a 'base' folder containing '_normalize.scss'. The file content is as follows:

```
/*
details {
  display: block;
}

/* Add the correct display in all browsers.
summary {
  display: list-item;
}

/* Misc
template {
  /* Add the correct display in IE 10+.
  */

  /* Add the correct display in all browsers.
  */
}
```

Recordar añadirlo para que lo pueda encontrar:

```

index.html package-lock.json package.json gulpfile.js app.scss _normalize.scss _variables.scss app.css 2
src > scss > app.scss > a
1 @use "base/variables" as v;
2 @use "base/normalize"; ←
3
4 body {
5   font-family: "Montserrat", sans-serif;
6   font-optical-sizing: auto;
7   font-style: normal;
8 }
9
10 h1 {
11   font-family: v.$Fuente_principal;
12 }
13
14 p {
15   font-family: v.$Fuente_principal;
16 }
17
18 a {
19   font-family: v.$Fuente_principal;
20 }
21

```

Ahora al compilarlo, podrás ver como lo ha aplicado completamente al css:

```

EXPLORER index.html package-lock.json package.json gulpfile.js app.scss _normalize.scss _variables.scss app.css 2
OPEN EDITORS index.html package-lock.json package.json gulpfile.js app.scss _normalize.scss _variables.scss app.css build/css
src > scss > base > _normalize.scss > ...
320 */
321
322 details {
323   display: block;
324 }
325
326 /*
327   * Add the correct display in all browsers.
328   */
329
330 summary {
331   display: list-item;
332 }
333
334 /* MISC
335 */
336
337 /**
338   * Add the correct display in IE 10+.
339   */
340
341 template {
342   display: none;
343 }
344
345 /**
346   * Add the correct display in IE 10.
347   */
348

```

El uso de @use @import @forward

@import EN DESUSO

- Función: Importa archivos Sass o CSS.
- Problemas: Importa el archivo entero y ejecuta su contenido en el mismo contexto, lo que puede causar problemas de duplicación y rendimiento.

@use

- Función: Importa un archivo Sass y pone su contenido en un namespace, evitando conflictos de nombres y mejorando la modularidad.
- Características:
 - No ejecuta el código globalmente.
 - Permite controlar qué se expone con @forward.

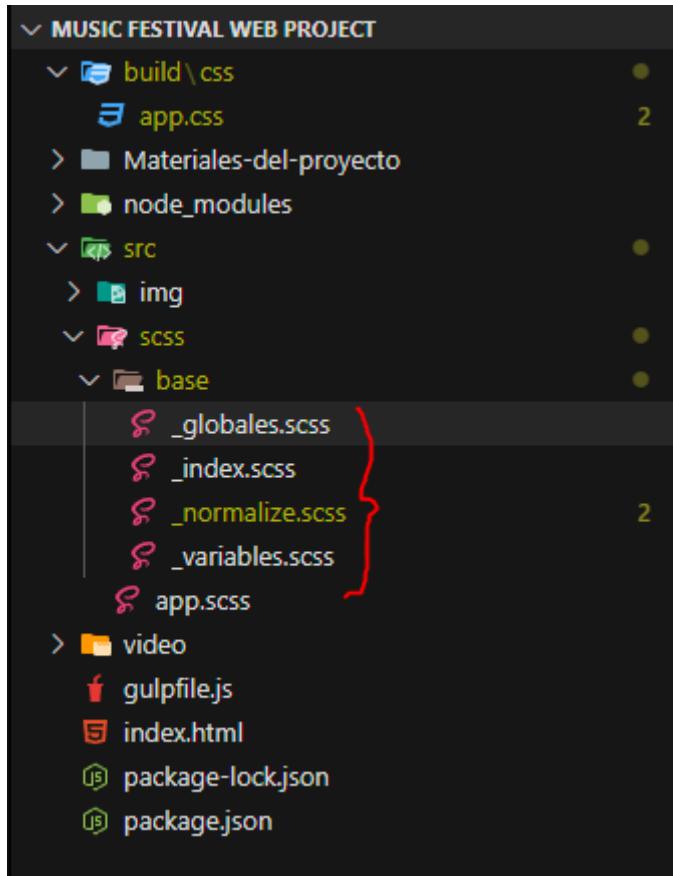
@forward

- Función: Reexporta un archivo Sass para que otros archivos que usen este módulo puedan acceder a su contenido.
- Características:
 - Controla qué se reexporta.
 - Facilita la creación de puntos de entrada.

Utilizaremos forward cuando queramos traernos TODO el contenido de una hoja de estilos y no requieras de un name space. Dentro no hay mixins ni variables ni nada, por lo que convine usar este método.

En el caso en el que queramos acceder a cosas específicas dentro del archivo, entonces se utilizará @use.

Creación y enlazamiento de todos los archivos del proyecto de forma ordenada



Este será el índice que conecta todos los archivos:

```

index.html package-lock.json package.json gulpfile.js app.scss _normalize.scss 2 _index.scss x _variables.scss app.css 2 _globales.scss
src > scss > base > _index.scss
1 @forward "normalize";
2 @forward "variables";
3 @forward "globales";
4

```

Primero carga normalize, luego variables y finalmente globales.

En globales estarán los estilos de la página:

```

index.html package-lock.json package.json gulpfile.js app.scss _normalize.scss 2 _index.scss x _variables.scss app.css 2 _globales.scss x
src > scss > base > _globales.scss ...
1 @use "variables" as v;
2
3 body {
4   font-family: v.$Fuente_principal;
5   font-optical-sizing: auto;
6   font-style: normal;
7   color: v.$negro;
8 }
9

```

Y en variables, las variables que utilizaremos:

```
index.html package-lock.json package.json gulpfile.js app.scss normalize.scss index.scss variables.scss app.css 2 globales.scss
src > scss > base > _variables.scss > ...
1 $Fuente_principal: "Montserrat", sans-serif;
2
3 // Colores
4 $verde: #4cb8b3;
5 $rosa: #f53756;
6 $amarillo: #fdde00;
7 $morado: #752f97;
8 $negro: #000000;
9 $blanco: #ffffff;
10 |
```

¿QUE SON LOS Mixins?

Los mixins en SASS te van a permitir escribir código que pueda ser re-utilizado en tus hojas de estilos.

De esta forma la cantidad de clases o código repetido puede ser considerablemente menor.

También con los Mixins tu código HTML será semánticamente mejor.

¿Cómo se crean?

@ mixin nombre-mixin

¿Cómo se utilizan?

@include nombre-mixin

```

31
32 @mixin contenedor {
33   width: 95%;
34   max-width: 120rem;
35   margin: 0 auto;
36 }
37
38 .header-contenedor {
39   @include contenedor();
40 }
41
42 .nav-contenedor {
43   @include contenedor();
44 }
45

```

CREACIÓN Y DEFINICIÓN

Uso 1

Uso 2

¿USANDO Mixins para MEDIA Querys?

Definimos las propiedades del media query en _header.scss:

```

1 @use "base/variables" as v;
2 @use "base/mixins" as m;
3
4 .header {
5   background-color: v.$verde;
6
7   @include m.telefono {
8     color: v.$blanco;
9     width: 10px;
10    height: 1px;
11  }
12
13 .contenido-header {
14   padding: 2rem;
15   display: flex;
16   justify-content: space-between;
17   align-items: center;
18 }
19

```

Creamos el media query y usaremos la propiedad **content** para aplicar el contenido indicado para ese media query desde _header.scss:

```
index.html _index.scss app.scss _header.scss _variables.scss _mixins.scss app.css 2  
src > scss > base > _mixins.scss > ...  
  
1 @mixin telefono {  
2   @media (min-width: 768px) {  
3     @content;  
4   }  
5 }  
6 |
```

Esto creará el media query al compilarlo con las propiedades indicadas:

```
.header {  
  background-color: #4cb8b3;  
}  
@media (min-width: 768px) {  
  .header {  
    color: #ffffff;  
    width: 10px;  
    height: 1px;  
  }  
}  
.header .contenido-header {  
  padding: 2rem;  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
}
```

Mixins que debes utilizar en los proyectos:

_mixins.scss:

```
@use "variables" as v;  
  
@mixin telefono {  
  @media (min-width: v.$telefono) {  
    @content;  
  }  
}
```

```

        }

@mixin tablet {
  @media (min-width: v.$tablet) {
    @content;
  }
}

@mixin desktop {
  @media (min-width: v.$desktop) {
    @content;
  }
}

@mixin desktopXL {
  @media (min-width: v.$desktopXL) {
    @content;
  }
}

```

Mixins que debes utilizar en los proyectos:

_variables.scss:

```

$fuente_principal: "Montserrat", sans-serif;

// Colores
$verde: #4cb8b3;
$rosa: #f53756;
$amarillo: #fdde00;
$morado: #752f97;
$negro: #000000;
$blanco: #ffffff;

// Media Queries
$telefono: 480px;
$tablet: 768px;
$desktop: 1200px;
$desktopXL: 1400px;

```

¿CÓMO agregar videos a una web?

```

<video autoplay muted loop>
  <source src="video/dj.mp4" type="video/mp4" />
  <source src="video/dj.ogv" type="video/ogg" />
  <source src="video/dj.webm" type="video/webm" />
</video>

```

Si se selecciona autoplay, será necesario añadir muted, de lo contrario exploradores como Chrome lo bloquearán.

Se han añadido varios formatos para soportarlo en diferentes exploradores y dispositivos, usar los mas comunes en primer lugar, por ejemplo mp4, o ogg para exploradores muy antiguos.

¿MIXINS para modificar activamente el número de columnas de un GRID?

```
index.html _video.scss app.scss _header.scss _variables.scss _mixins.scss X festival.scss
src > scss > base > _mixins.scss > grid
  ↳
33 @mixin grid($columnas) {
34   display: grid;
35   grid-template-columns: repeat($columnas, 1fr);
36   gap: 5rem;
37 }
38
```



```
index.html _video.scss app.scss _header.scss _variables.scss _mixins.scss festival.scss X _index.scss app.css 2
src > scss > layout > festival.scss > ...
1 @use "base/variables" as v;
2 @use "base/mixins" as m;
3
4 .sobre-festival {
5   @include m.contenedor;
6
7   @include m.tablet {
8     @include m.grid(2);
9   }
10 }
11
12 .contenido-festival {
13   margin-top: 5rem;
14   h2 {
15     margin-bottom: 2rem;
16   }
17   .fecha {
18     color: v.$verde;
19     font-weight: 700;
20   }
21 }
22
```

Así desde el .scss podemos indicarle de cuánto queremos que sean las columnas y automáticamente el mixin las creará.

Además, podemos indicarles valores por defecto para en el caso en el que no se los trasmitamos:

```
index.html _video.scss app.scss _header.scss _variables.scss _mixins.scss festival.scss _index.scss app.css 2  
src > scss > base > _mixins.scss > ...  
20  
21 @mixin desktopXL {  
22     @media (min-width: v.$desktopXL) {  
23         @content;  
24     }  
25 }  
26  
27 @mixin contenedor {  
28     width: 95%;  
29     max-width: 120rem;  
30     margin: 0 auto;  
31 }  
32  
33 @mixin grid($columnas: 1, $gap: 5rem) {  
34     display: grid;  
35     grid-template-columns: repeat($columnas, 1fr);  
36     gap: $gap;  
37 }  
38 |
```

¿Cómo evitar demasiado especificidad en el CSS compilado resultante?

En lugar de:

```
.pase {  
    .pase-nombre {  
        font-size: 3.6rem;  
        font-weight: 900;  
        text-align: center;  
        color: v.$blanco;  
    }  
}  
  
.pase .pase-nombre {  
    font-size: 3.6rem;  
    font-weight: 900;  
    text-align: center;  
    color: #ffffff;
```

SCSS antes de compilar.

CSS resultante después de compilar.

Hacer esto:

```
.pase {  
    &-nombre {  
        font-size: 3.6rem;  
        font-weight: 900;  
        text-align: center;  
        color: v.$blanco;  
    }  
}  
  
.pase-nombre {  
    font-size: 3.6rem;  
    font-weight: 900;  
    text-align: center;  
    color: #ffffff;
```

SCSS antes de compilar.

CSS resultante después de compilar.

¿CÓMO saber al inspeccionar en que archivo SCSS está lo que estás inspeccionando con el cursor?

Añadir estas líneas al archivo gulpfile.js para que al compilar se genere el mapa

```
index.html footer.scss gulpfile.js
gulpfile.js > css > sourcemaps
1 import { src, dest, watch } from "gulp";
2 import * as dartSass from "sass";
3 import gulpSass from "gulp-sass";
4
5 const sass = gulpSass(dartSass);
6
7 export function css(done) {
8     src("src/scss/app.scss", { sourcemaps: true })
9         .pipe(sass().on("error", sass.logError))
10        .pipe(dest("build/css", { sourcemaps: "." }));
11    done();
12 }
13
14 export function dev() {
15     watch("src/scss/**/*.scss", css);
16 }
17
```

¿CÓMO ejecutar varias tareas seguidas, una tras otra?

Utilizaremos series:

```
index.html footer.scss app.js gulpfile.js package.json
gulpfile.js > ...
1 import { src, dest, watch, series } from "gulp";
2 import * as dartSass from "sass";
3 import gulpSass from "gulp-sass";
4
5 const sass = gulpSass(dartSass);
6
7 /* Esta función lo que hará es coger una copia del archivo app.js y llevarla a la carpeta build para poderla agregar.*/
8 export function js(done) {
9     src("src/js/app.js").pipe(dest("build/js"));
10    done();
11 }
12
13 export function css(done) {
14     src("src/scss/app.scss", { sourcemaps: true })
15         .pipe(sass().on("error", sass.logError))
16         .pipe(dest("build/css", { sourcemaps: "." }));
17     done();
18 }
19
20 export function dev(done) {
21     watch("src/scss/**/*.scss", css);
22     done();
23 }
24
25 export default series(js, css, dev);
```

```
index.html footer.scss app.js gulpfile.js package.json
package.json > ...
1 {
2   "name": "music-festival-web-project",
3   "version": "1.0.0",
4   "description": "Aprendiendo SASS y NPM",
5   "type": "module",
6   "main": "index.js",
7   > Debug
8   "scripts": {
9     "sass": "sass --watch src/scss:build/css",
10    "dev": "gulp dev" ←
11  },
12  "keywords": [
13    "SASS",
14    "NPM",
15    "Gulp"
16  ],
17  "author": "Derimán Tejera Fumero",
18  "license": "ISC",
19  "devDependencies": {
20    "gulp": "^5.0.0",
21    "gulp-sass": "^5.1.0",
22    "sass": "^1.77.6"
23  }
24 }
```

Lo que ocurre es que arranca series y aborda la primera tarea que es js y la finaliza, luego va a css y la finaliza y para terminar hace lo mismo con dev:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
[21:11:48] Using gulpfile ~\Desktop\Music festival web project\gulpfile.js
[21:11:48] Starting 'default'...
[21:11:48] Starting 'js'...
[21:11:48] Finished 'js' after 6.61 ms
[21:11:48] Starting 'css'...
[21:11:48] Finished 'css' after 3.74 ms
[21:11:48] Starting 'dev'...
[21:11:48] Finished 'dev' after 5.9 ms
[21:11:48] Finished 'default' after 21 ms
```

En lugar de series también se puede usar parallel que lo que hace es iniciar todas las tareas a la vez e ir finalizándolas también a la vez. Mientras que series va paso a paso, pudiendo controlar cuales se inician y su orden exacto, lo que puede ser muy útil en según que casos.

¿CÓMO mejorar el Performance web?

Minificar CSS

Gulpfile.js sin modificar y que NO hace la minificación del archivo CSS:

```
export function css(done) {
  src("src/scss/app.scss", { sourcemaps: true })
    .pipe(sass().on("error", sass.logError))
    .pipe(dest("build/css", { sourcemaps: "." }));
  done();
}
```

Gulpfile.js modificado para poder hacer la minificación sobre el archivo CSS:

```
index.html  app.css  gulpfile.js
gulpfile.js > ...
1 import { src, dest, watch, series } from "gulp";
2 import * as dartSass from "sass";
3 import gulpSass from "gulp-sass";
4
5 const sass = gulpSass(dartSass);
6
7 /* Esta función lo que hará es coger una copia del archivo app.js y llevarla a la carpeta build para poderla agregar.*/
8 export function js(done) {
9   src("src/js/app.js").pipe(dest("build/js"));
10
11   done();
12 }
13
14 export function css(done) {
15   src("src/scss/app.scss", { sourcemaps: true })
16     .pipe(sass({ outputStyle: "compressed" }).on("error", sass.logError))
17     .pipe(dest("build/css", { sourcemaps: "." }));
18   done();
19 }
20
21 export function dev(done) {
22   watch("src/scss/**/*.scss", css);
23   watch("src/js/**/*.js", js);
24   done();
25 }
26
27 export default series(js, css, dev);
28 |
```

Para minificar el archivo CSS únicamente hay que ir al archivo “[gulpfile.js](#)” y añadir a .pipe(sass) el “outputStyle: “compressed””, por defecto está en “expanded” pero si le especificamos “compressed” y ejecutamos de nuevo el comando en el terminal de [npm run dev](#), y volvemos al archivo CSS podremos ver las modificaciones realizadas para la minificación:

1. **Eliminación de espacios en blanco:** Se eliminan los espacios, tabulaciones y nuevas líneas innecesarias.
2. **Eliminación de comentarios:** Todos los comentarios se eliminan.

3. **Reducción de nombres de selectores y propiedades:** Se pueden abbreviar ciertos nombres, aunque esto es más común en minificadores avanzados.
4. **Uso de valores más cortos:** Por ejemplo, #ffffff se convierte en #fff.
5. **Eliminación de unidades redundantes:** Valores de cero (0) no necesitan unidades (0px se convierte en 0).
6. **Agrupación de reglas:** Reglas similares se agrupan para evitar redundancias.

Antes de minificar:

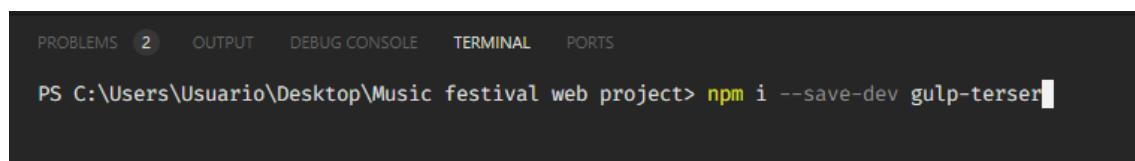
Nombre	Fecha de modificación	Tipo	Tamaño
# app.css	26/06/2024 18:38	Archivo de origen ...	14 KB
app.css.map	26/06/2024 18:38	Archivo MAP	21 KB

Después de minificar:

Nombre	Fecha de modificación	Tipo	Tamaño
# app.css	26/06/2024 18:29	Archivo de origen ...	7 KB
app.css.map	26/06/2024 18:29	Archivo MAP	20 KB

Minificar JS

Para minificar los archivos JS necesitaremos descarga antes por terminal la dependencia Terser, para ello utilizaremos el comando: [npm i --save-dev gulp-terser](#)



```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Usuario\Desktop\Music festival web project> npm i --save-dev gulp-terser
```

Para que pueda funcionar, tenemos que añadir 2 líneas de código:

```

index.html gulpfile.js app.css 2
gulpfile.js > ...
1 import { src, dest, watch, series } from "gulp";
2 import * as dartSass from "sass";
3 import gulpSass from "gulp-sass";
4
5 const sass = gulpSass(dartSass);
6
7 import terser from "gulp-terser"; ←
8
9 /* Esta función lo que hará es coger una copia del archivo app.js y llevarla a la carpeta build para poderla agregar.*/
10 export function js(done) {
11   src("src/js/app.js")
12     .pipe(terser())
13     .pipe(dest("build/js"));
14
15   done();
16 }
17
18 export function css(done) {
19   src("src/scss/app.scss", { sourcemaps: true })
20     .pipe(sass({ outputStyle: "compressed" }).on("error", sass.logError))
21     .pipe(dest("build/css", { sourcemaps: "." }));
22   done();
23 }
24
25 export function dev(done) {
26   watch("src/scss/**/*.scss", css);
27   watch("src/js/**/*.js", js);
28   done();
29 }
30
31 export default series(js, css, dev);
32

```

Antes de minificar:

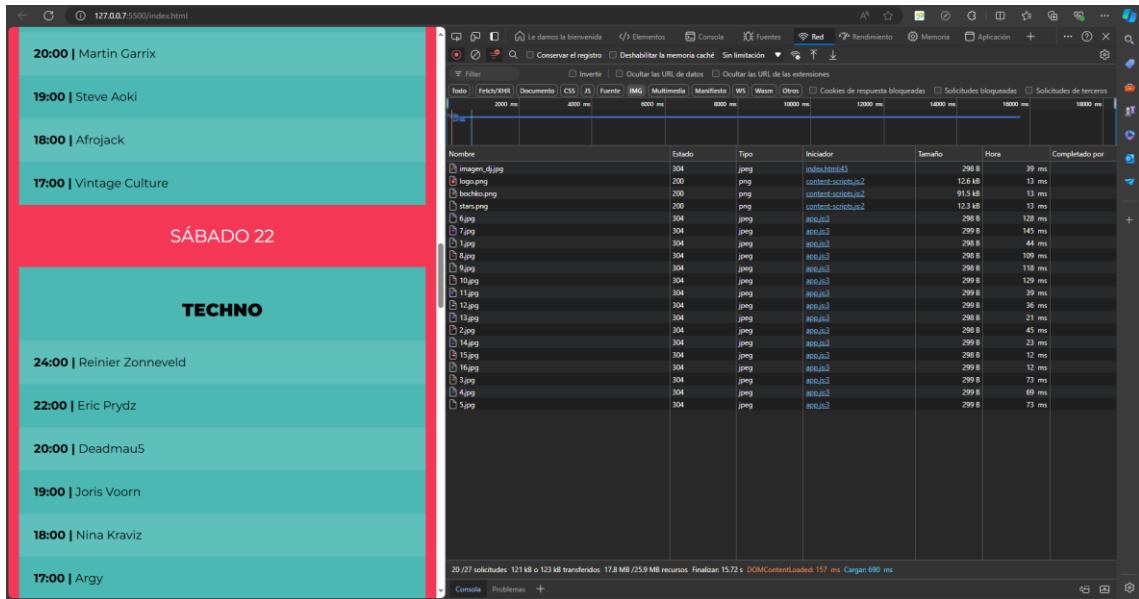
Nombre	Fecha de modificación	Tipo	Tamaño
app.js	26/06/2024 19:00	Archivo de origen ...	5 KB

Después de minificar:

Nombre	Fecha de modificación	Tipo	Tamaño
app.js	26/06/2024 18:59	Archivo de origen ...	3 KB

Carga diferida de imágenes con LAZY LOADING

Antes de nada vamos a ver lo que pesa la página al hacer una carga de ella (Usar F12 y luego ir a la pestaña RED y luego a IMG):



El peso es de 17,8 MB.

Añadiendo Lazy Load se puede hacer que no se carguen las imágenes hasta que no entren en el campo de visión del usuario, por lo tanto no son necesarias.

Para añadir este tipo de carga a las imágenes cargadas desde el HTML, debemos añadir los siguientes atributos:

```
<section class="sobre-festival">
  <div class="imagen">
    
  </div>
```

Normalmente con `loading="lazy"` puede no ser suficiente, por lo que deberemos especificar el tamaño estimado en anchura `width` y altura `height` para que funcione. Estos valores no tienen que ser exactos, únicamente deben ser aproximados.

Para aplicar el `loading="lazy"` a las imágenes cargadas desde el JS deberemos:

```

1 index.html
2 gulpfile.js
3 app.js
4 app.css 2
5
6 build > js > app.js > crearGaleria
7 20 }
8 21
9 22 function crearGaleria() {
10 23 const CANTIDAD_IMAGENES = 16;
11 24 const galeria = document.querySelector(".galeria-imagenes");
12
13 25 for (let i = 1; i <= CANTIDAD_IMAGENES; i++) {
14 26 /* La convención dice que debe ponerse en mayúscula, en lugar de img, será IMG, en lugar de div, será DIV. */
15 27 const imagen = document.createElement("IMG");
16 28 imagen.loading = "lazy";
17 29 [ ]Imagen.width = "300";
18 30Imagen.height = "200";
19 31 imagen.src = `src/img/gallery/full/${i}.jpg`;
20 32 imagen.alt = "Imagen de Galería";
21 33
22 34 // Event Handler: Esto es detectar y responder a la interacción de un usuario, por ejemplo por su click.
23 35 imagen.onclick = function () {
24 36 mostrarImagen(i);
25 37 }
26 38
27 39 galeria.appendChild(imagen);
28 40
29 41 }
30 42
31 43
32 44 function mostrarImagen(i) {
33 45 const imagen = document.createElement("IMG");
34 46 imagen.src = `src/img/gallery/full/${i}.jpg`;
35 47 imagen.alt = "Imagen de Galería";
36
37 48

```

Recordar que el orden de aparición de los atributos es importante, por lo que para que aparezcan los primeros antes de la búsqueda de la imagen, deben ponerse antes, justo después de la creación del elemento “IMG”.

Características en la carga de videos

[Tecnología para desarrolladores web](#) > [HTML: Lenguaje de etiquetas de hipertexto](#) > [Referencia de Elementos HTML](#) > [video](#)

preload Error 548523 en Firefox ↗

El objetivo de este atributo enumerado es proporcionar una sugerencia al navegador sobre qué cree el autor que llevará a la mejor experiencia para el usuario. Puede tener uno de los siguientes valores:

- none: sugiere bien que el autor cree que el usuario no tendrá que consultar ese video, bien que el servidor desea minimizar su tráfico; es decir, esta sugerencia indica que no se debe almacenar en caché este video.
- metadatos: sugiere que aunque el autor piensa que el usuario no tendrá que consultar este video, es razonable capturar los metadatos (p. ej. longitud).
- auto: sugiere que el usuario necesita tener prioridad; es decir, esta sugerencia indica que, si es necesario, se puede descargar el video completo, incluso aunque el usuario no vaya a usarlo.
 - la *cadena vacía*: que es un sinónimo del valor auto.

Si no está configurado, su valor predeterminado está definido por el navegador (es decir, cada navegador puede elegir su propio valor predeterminado), aunque la especificación aconseja que se establezca a metadata.

[video - HTML: Lenguaje de etiquetas de hipertexto | MDN \(mozilla.org\)](#)

En este caso es necesario usar el video en “auto” porque es un elemento central del diseño de la web.

Para el caso del proyecto se utilizaría el caso de auto:

```
index.html  gulpfile.js  app.js  app.css  2
index.html > html > body
  2  <html lang="es">
  15 <body>
  27   <div class="video">
  28     <div class="overlay">
  29       <div class="contenedor-contenido-video">
  32         </div>
  33       </div>
  34       <video autoplay muted loop preload="auto">
  35         <source src="video/dj.mp4" type="video/mp4" />
  36         <source src="video/dj.ogv" type="video/ogg" />
  37         <source src="video/dj.webm" type="video/webm" />
  38       </video>
  39     </div>
  40
```

Aunque tal y como dice en la documentación, el valor "auto" es el valor por defecto, así que no hay que escribirlo.

Carga de imágenes de la galería primero en baja resolución y luego al clicar en alta

Para lograr esto usaremos la librería:

[sharp - npm \(npmjs.com\)](#)

```
index.html  gulpfile.js  app.js  app.css  2
gulpfile.js > drop
  14  }
  15
  16  export function css(done) {
  17    src("src/scss/app.scss", { sourcemaps: true })
  18      .pipe(sass({ outputStyle: "compressed" }).on("error", sass.logError))
  19      .pipe(dest("build/css", { sourcemaps: "." }));
  20    done();
  21  }
  22
  23  export async function crop(done) {
  24    const inputFolder = "src/img/gallery/full";
  25    const outputFolder = "src/img/gallery/thumb";
  26    const width = 250;
  27    const height = 180;
  28    if (!fs.existsSync(outputFolder)) {
  29      fs.mkdirSync(outputFolder, { recursive: true });
  30    }
  31    const images = fs.readdirSync(inputFolder).filter((file) => {
  32      return /\.jpg$/i.test(path.extname(file));
  33    });
  34    try {
  35      images.forEach((file) => {
  36        const inputFile = path.join(inputFolder, file);
  37        const outputFile = path.join(outputFolder, file);
  38        sharp(inputFile)
  39          .resize(width, height, {
  40            position: "centre",
  41          })
  42          .toFile(outputFile);
  43      });
  44
  45      done();
  46    } catch (error) {
  47      console.log(error);
  48    }
  49  }
```

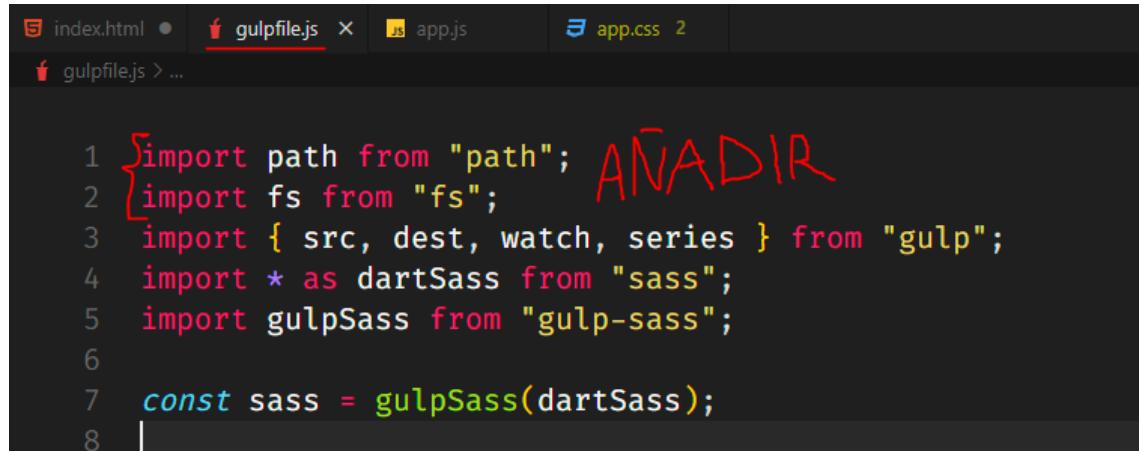
CSS

Código
Node.js

Lo que hace la función crop es buscar las imágenes de la galería y generar otras nuevas a partir de ellas con las especificaciones de width y height indicadas también en crop.

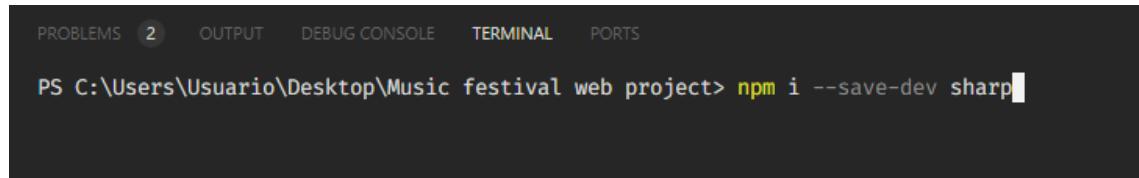
Este código es reutilizable si aplicamos las localizaciones de los archivos correctos para nuestro proyecto.

Recordar que el script añadido anteriormente utiliza las dependencias de Sharp y Fs, ambas existen en Node.js y podemos utilizar las dependencias de Node.js porque el gulpfile.js utiliza Node.js.



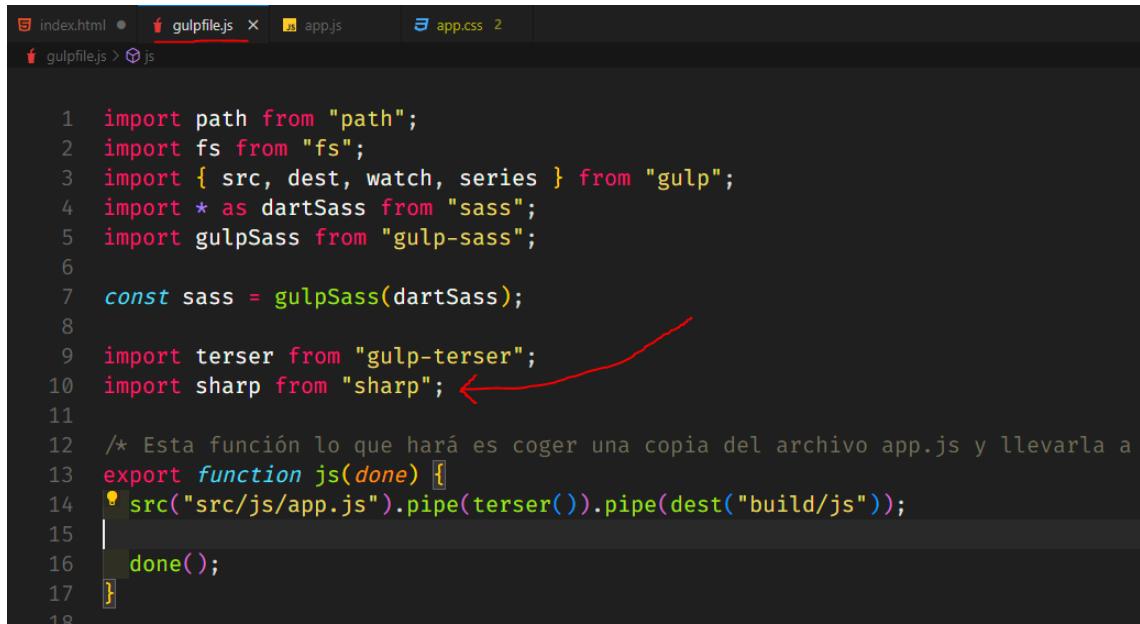
```
1 import path from "path"; AÑADIR
2 import fs from "fs";
3 import { src, dest, watch, series } from "gulp";
4 import * as dartSass from "sass";
5 import gulpSass from "gulp-sass";
6
7 const sass = gulpSass(dartSass);
8 |
```

Y quedaría descargar la dependencia de Sharp (que no está incluida en Node.js como si lo están path y fs:



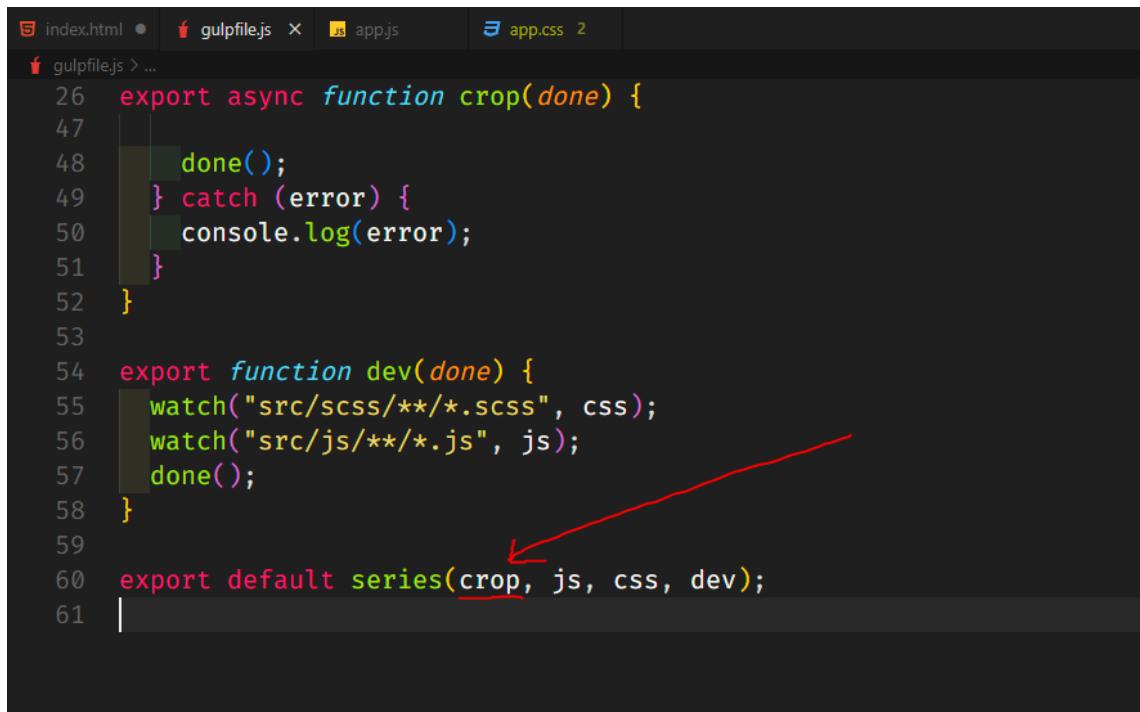
```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Usuario\Desktop\Music festival web project> npm i --save-dev sharp
```

También la tendremos que importar:



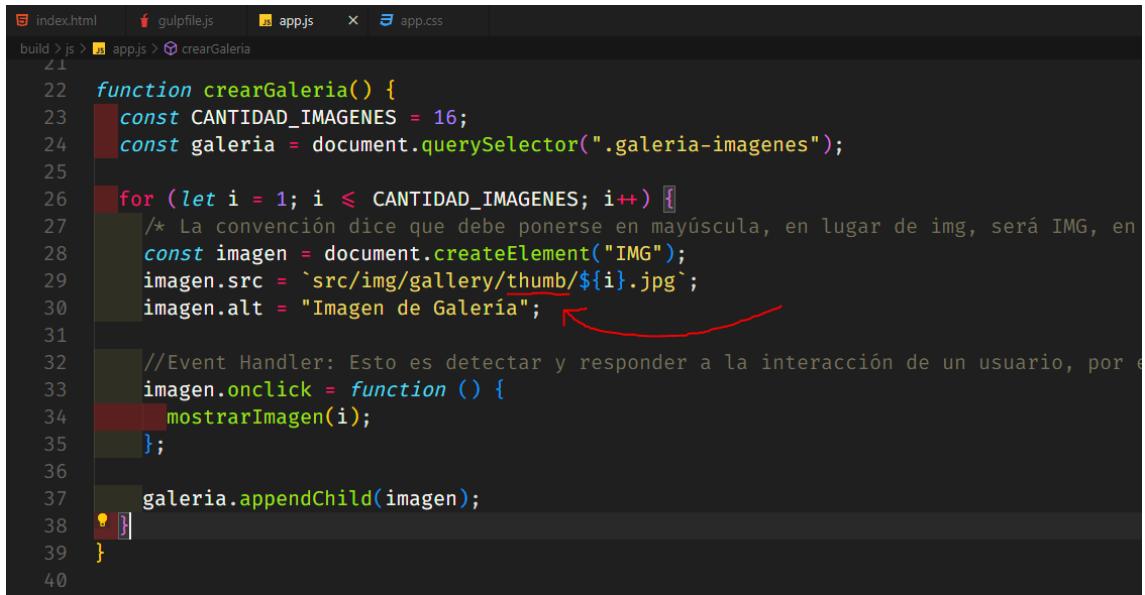
```
1 import path from "path";
2 import fs from "fs";
3 import { src, dest, watch, series } from "gulp";
4 import * as dartSass from "sass";
5 import gulpSass from "gulp-sass";
6
7 const sass = gulpSass(dartSass);
8
9 import terser from "gulp-terser";
10 import sharp from "sharp"; ←
11
12 /* Esta función lo que hará es coger una copia del archivo app.js y llevarla a
13 export function js(done) {
14     src("src/js/app.js").pipe(terser()).pipe(dest("build/js"));
15
16     done();
17 }
```

También añadiremos crop a series para que se ejecute el primero cuando ejecutemos [npm run dev](#):



```
1 index.html ●  gulpfile.js X  app.js  app.css  2
2 gulpfile.js > ...
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 export async function crop(done) {
27
28     done();
29 } catch (error) {
30     console.log(error);
31 }
32
33
34 export function dev(done) {
35     watch("src/scss/**/*.{scss,sass}", css);
36     watch("src/js/**/*.{js,jsx}", js);
37     done();
38 }
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60 export default series(crop, js, css, dev);
61 |
```

Ahora solo quedaría cambiar la ruta de carga de las imágenes mostradas en galería en el archivo app.js y cambiarla por la nueva carpeta thumb.



```
index.html gulpfile.js app.js app.css
build > js > app.js > crearGaleria
21
22 function crearGaleria() {
23     const CANTIDAD_IMAGENES = 16;
24     const galeria = document.querySelector(".galeria-imagenes");
25
26     for (let i = 1; i <= CANTIDAD_IMAGENES; i++) {
27         /* La convención dice que debe ponerse en mayúscula, en lugar de img, será IMG, en
28         const imagen = document.createElement("IMG");
29         imagen.src = `src/img/gallery/thumb/${i}.jpg`;
30         imagen.alt = "Imagen de Galería"; */
31
32         //Event Handler: Esto es detectar y responder a la interacción de un usuario, por ejemplo
33         imagen.onclick = function () {
34             mostrarImagen(i);
35         };
36
37         galeria.appendChild(imagen);
38     }
39 }
40
```

Utilizar formatos modernos para las imágenes

Webp es un formato de imagen moderno que soporta transparencias (o no), además no pierde calidad y tiene un tamaño considerablemente menor.

Tiene una compresión significativamente mejor que JPEG y PNG.

La mayoría de navegadores modernos ya soportan el formato webp.

Desventajas:

Usan un poco más de código HTML.

No todas las aplicaciones de diseño gráfico soportan este formato.

Para poder generar las imágenes en .webp vamos a necesitar añadir este código a `gulpfile.js`:

```
export async function imagenes(done) {
    const srcDir = './src/img';
    const buildDir = './build/img';
    const images = await glob('./src/img/**/*{jpg,png}')

    images.forEach(file => {
        const relativePath = path.relative(srcDir,
path.dirname(file));
        const outputSubDir = path.join(buildDir, relativePath);
        procesarImagenes(file, outputSubDir);
    });
    done();
}

function procesarImagenes(file, outputSubDir) {
```

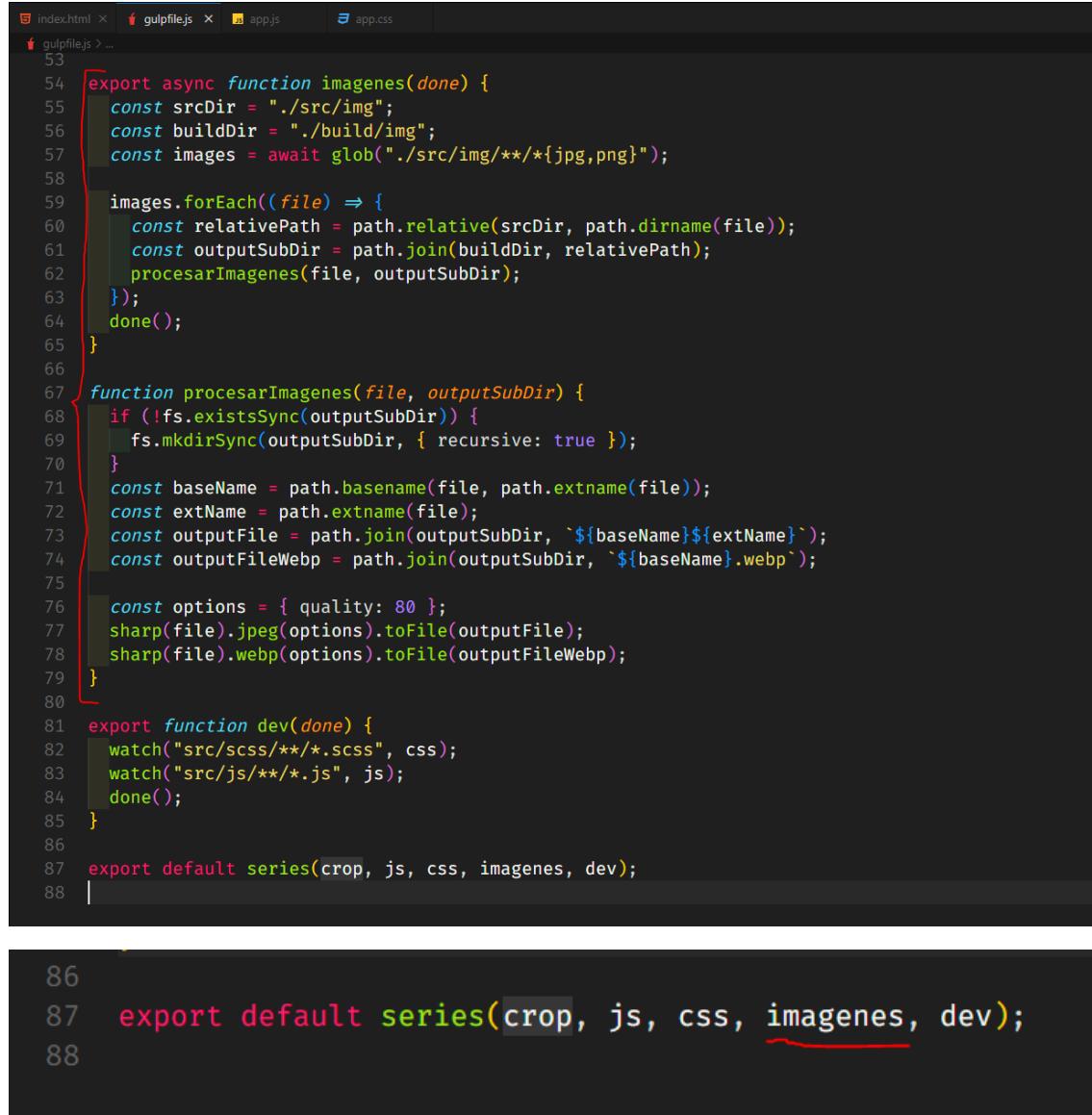
```

        if (!fs.existsSync(outputSubDir)) {
            fs.mkdirSync(outputSubDir, { recursive: true })
        }
        const baseName = path.basename(file, path.extname(file))
        const extName = path.extname(file)
        const outputFile = path.join(outputSubDir,
` ${baseName}${extName}`)
        const outputFileWebp = path.join(outputSubDir,
` ${baseName}.webp`)

        const options = { quality: 80 }
        sharp(file).jpeg(options).toFile(outputFile)
        sharp(file).webp(options).toFile(outputFileWebp)
    }
}

```

Lo vamos a pegar después de la función crop:



```

index.html x  gulpfile.js x  app.js  app.css
gulpfile.js > ...
53
54     export async function imagenes(done) {
55         const srcDir = "./src/img";
56         const buildDir = "./build/img";
57         const images = await glob("./src/img/**/*.{jpg,png}");
58
59         images.forEach((file) => {
60             const relativePath = path.relative(srcDir, path.dirname(file));
61             const outputSubDir = path.join(buildDir, relativePath);
62             procesarImagenes(file, outputSubDir);
63         });
64         done();
65     }
66
67     function procesarImagenes(file, outputSubDir) {
68         if (!fs.existsSync(outputSubDir)) {
69             fs.mkdirSync(outputSubDir, { recursive: true });
70         }
71         const baseName = path.basename(file, path.extname(file));
72         const extName = path.extname(file);
73         const outputFile = path.join(outputSubDir, `${baseName}${extName}`);
74         const outputFileWebp = path.join(outputSubDir, `${baseName}.webp`);

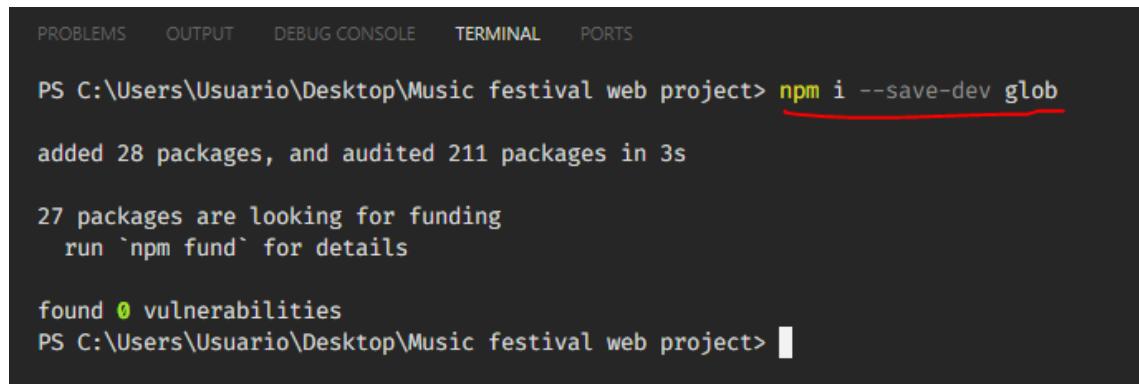
75         const options = { quality: 80 };
76         sharp(file).jpeg(options).toFile(outputFile);
77         sharp(file).webp(options).toFile(outputFileWebp);
78     }
79 }
80
81     export function dev(done) {
82         watch("src/scss/**/*.scss", css);
83         watch("src/js/**/*.js", js);
84         done();
85     }
86
87     export default series(crop, js, css, imagenes, dev);
88

```

Recordar que dev siempre tiene que ser la última de la serie porque es la que está escuchando por los cambios.

Si examinas el código puedes ver que hay dos funciones, la función imágenes localiza las carpetas donde están las imágenes a convertir y luego llama a la segunda función, procesarImagenes que hace lo que indica su nombre, creando dos tipos de archivos de salida, para los formatos [.jpeg](#) y [.webp](#).

Recuerda también que se utiliza una dependencia que debes tener descargada, la dependencia es [glob](#) y está en la función imágenes.



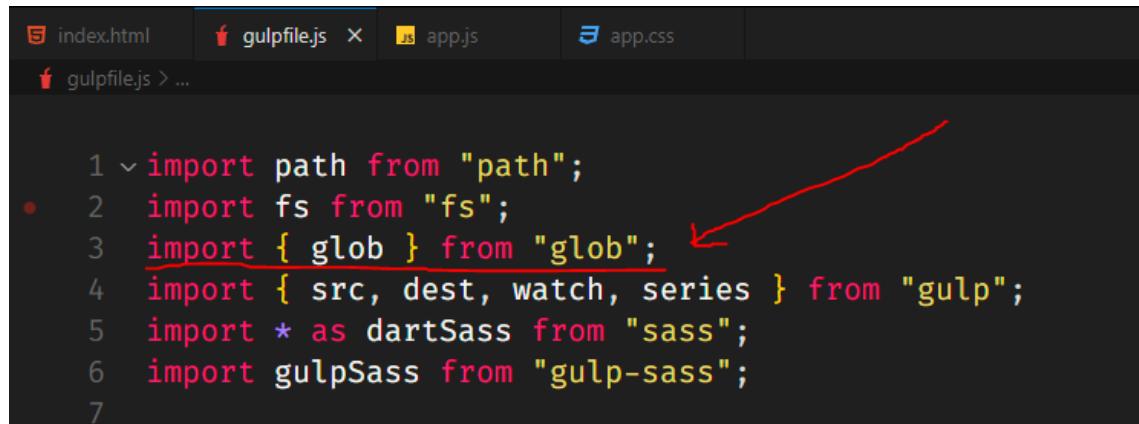
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Usuario\Desktop\Music festival web project> npm i --save-dev glob
added 28 packages, and audited 211 packages in 3s

27 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\Usuario\Desktop\Music festival web project>
```

También deberás añadir el import correspondiente en la parte inicial del archivo gulpfile.js:

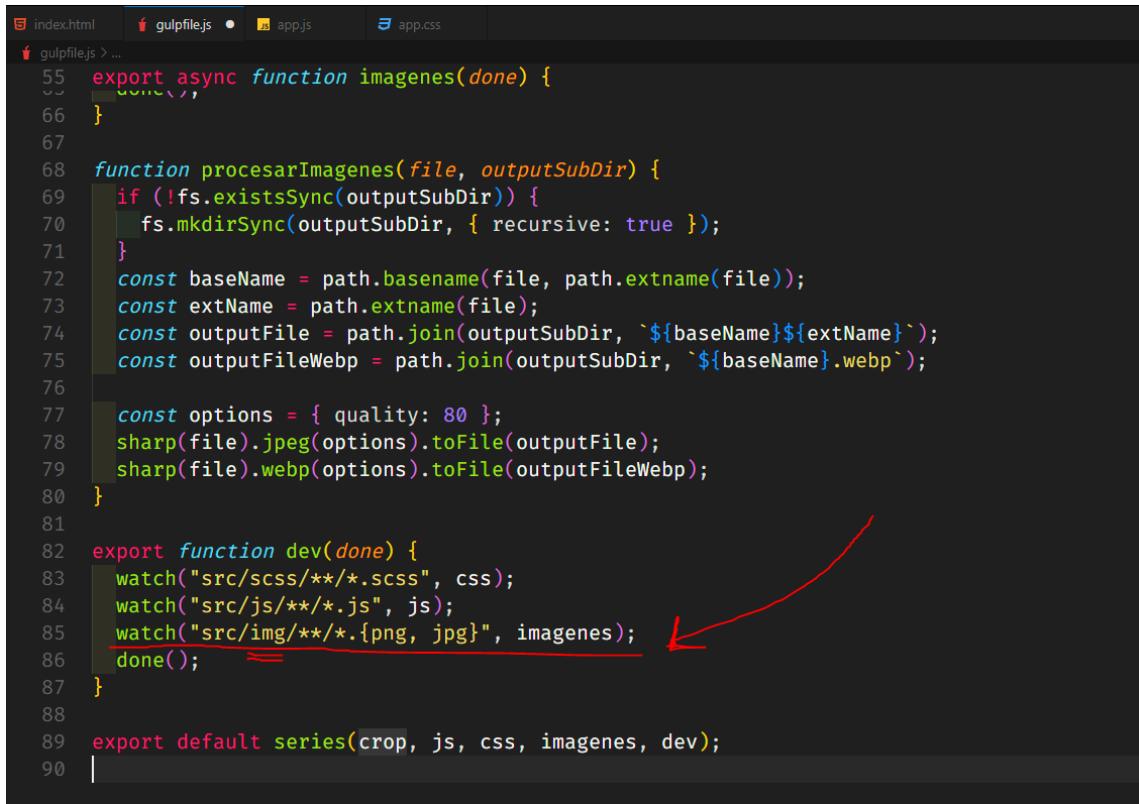


```
index.html gulpfile.js app.js app.css

 gulpfile.js > ...

 1 ~ import path from "path";
 2 import fs from "fs";
 3 import { glob } from "glob"; ↖
 4 import { src, dest, watch, series } from "gulp";
 5 import * as dartSass from "sass";
 6 import gulpSass from "gulp-sass";
 7
```

Y también deberemos añadir la siguiente línea en la función dev para que busque todas las imágenes en los formatos indicados de forma recursiva en la carpeta img:



```

index.html  gulpfile.js  app.js  app.css
 gulpfile.js > ...
  55   export async function imagenes(done) {
  56     done();
  57   }
  58
  59   function procesarImagenes(file, outputSubDir) {
  60     if (!fs.existsSync(outputSubDir)) {
  61       fs.mkdirSync(outputSubDir, { recursive: true });
  62     }
  63     const baseName = path.basename(file, path.extname(file));
  64     const extName = path.extname(file);
  65     const outputFile = path.join(outputSubDir, `${baseName}${extName}`);
  66     const outputFileWebp = path.join(outputSubDir, `${baseName}.webp`);
  67
  68     const options = { quality: 80 };
  69     sharp(file).jpeg(options).toFile(outputFile);
  70     sharp(file).webp(options).toFile(outputFileWebp);
  71   }
  72
  73   export function dev(done) {
  74     watch("src/scss/**/*.{scss}", css);
  75     watch("src/js/**/*.{js}", js);
  76     watch("src/img/**/*.{png, jpg}", imagenes); ←
  77     done(); ===
  78   }
  79
  80   export default series(crop, js, css, imagenes, dev);
  81
  82
  83
  84
  85
  86
  87
  88
  89
  90

```

Y ahora volvemos a ejecutar el comando en terminal: npm run dev y si vamos a los archivos de la web, encontraremos en los archivos de gallery>thumb las fotos en los formatos .jpg y .webp.

Usuarios	>	Usuario	>	Escritorio	>	Music festival web project	>	build	>	img	>	gallery	>	thumb
Nombre				Fecha		Tipo		Tamaño		Etiquetas				
1.jpg				27/06/2024 11:50		Archivo JPG		8 KB						
1.webp				27/06/2024 11:50		Archivo WEBP		6 KB						
2.jpg				27/06/2024 11:50		Archivo JPG		10 KB						
2.webp				27/06/2024 11:50		Archivo WEBP		7 KB						
3.jpg				27/06/2024 11:50		Archivo JPG		12 KB						
3.webp				27/06/2024 11:50		Archivo WEBP		10 KB						
4.jpg				27/06/2024 11:50		Archivo JPG		12 KB						
4.webp				27/06/2024 11:50		Archivo WEBP		10 KB						
5.jpg				27/06/2024 11:50		Archivo JPG		11 KB						
5.webp				27/06/2024 11:50		Archivo WEBP		9 KB						
6.jpg				27/06/2024 11:50		Archivo JPG		6 KB						
6.webp				27/06/2024 11:50		Archivo WEBP		4 KB						
7.jpg				27/06/2024 11:50		Archivo JPG		10 KB						
7.webp				27/06/2024 11:50		Archivo WEBP		8 KB						
8.jpg				27/06/2024 11:50		Archivo JPG		11 KB						
8.webp				27/06/2024 11:50		Archivo WEBP		9 KB						
9.jpg				27/06/2024 11:50		Archivo JPG		10 KB						
9.webp				27/06/2024 11:50		Archivo WEBP		8 KB						
10.jpg				27/06/2024 11:50		Archivo JPG		12 KB						
10.webp				27/06/2024 11:50		Archivo WEBP		9 KB						
11.jpg				27/06/2024 11:50		Archivo JPG		9 KB						
11.webp				27/06/2024 11:50		Archivo WEBP		7 KB						
12.jpg				27/06/2024 11:50		Archivo JPG		13 KB						
12.webp				27/06/2024 11:50		Archivo WEBP		11 KB						
13.jpg				27/06/2024 11:50		Archivo JPG		9 KB						
13.webp				27/06/2024 11:50		Archivo WEBP		7 KB						
14.jpg				27/06/2024 11:50		Archivo JPG		9 KB						
14.webp				27/06/2024 11:50		Archivo WEBP		7 KB						
15.jpg				27/06/2024 11:50		Archivo JPG		9 KB						
15.webp				27/06/2024 11:50		Archivo WEBP		7 KB						
16.jpg				27/06/2024 11:50		Archivo JPG		12 KB						
16.webp				27/06/2024 11:50		Archivo WEBP		9 KB						

Fíjate en la diferencia de peso de los diferentes formatos.

Formato AVIF (aún mejor que .webp)

Las imágenes AVIF (AV1 Image File Format) son un formato de archivo de imagen basado en el códec de video AV1.

Este formato se diseñó pensando en la eficiencia en la compresión y la calidad de imagen, lo que lo hace particularmente atractivo para su uso en la web.

Ventajas:

Compresión altamente eficiente: AVIF ofrece una compresión de imágenes muy eficiente, lo que resulta en archivos significativamente más pequeños en comparación con otros formatos populares como JPEG, PNG y WebP.

Calidad de imagen excepcional: A pesar de su alta tasa de compresión, AVIF mantiene una calidad de imagen superior

Compatibilidad: La mayoría de los navegadores modernos soportan el formato AVIF.

¿Cuál debo utilizar AVIF o WebP?

Gracias a la etiqueta <picture> podemos especificar diferentes formatos, en caso de que uno no sea soportado, irá al siguiente <source> hasta llegar al fallback.

El orden recomendado es primero AVIF ya que es el más ligero de los 3, después WebP y finalmente JPG o fallback.

¿Cómo hay que modificar la función procesarImagenes para convertir también a AVIF?

La dependencia Sharp también soporta AVIF, así que las modificaciones se pueden hacer sobre la función que ya tenemos de procesarImagenes:

```

index.html      gulpfile.js x app.js    app.css
 gulpfile.js > ...
 68  function procesarImagenes(file, outputSubDir) {
 69    if (!fs.existsSync(outputSubDir)) {
 70      fs.mkdirSync(outputSubDir, { recursive: true });
 71    }
 72    const baseName = path.basename(file, path.extname(file));
 73    const extName = path.extname(file);
 74    const outputFile = path.join(outputSubDir, `${baseName}${extName}`);
 75    const outputFileWebp = path.join(outputSubDir, `${baseName}.webp`);
 76    const outputFileAvif = path.join(outputSubDir, `${baseName}.avif`);

 77    const options = { quality: 80 };
 78    sharp(file).jpeg(options).toFile(outputFile);
 79    sharp(file).webp(options).toFile(outputFileWebp);
 80    sharp(file).avif().toFile(outputFileAvif);
 81  }

 82 }

 83 export function dev(done) {
 84   watch("src/scss/**/*.scss", css);
 85   watch("src/js/**/*.js", js);
 86   watch("src/img/**/*.{png, jpg}", imagenes);
 87   done();
 88 }

 89 }

 90 export default series(crop, js, css, imagenes, dev);

 91

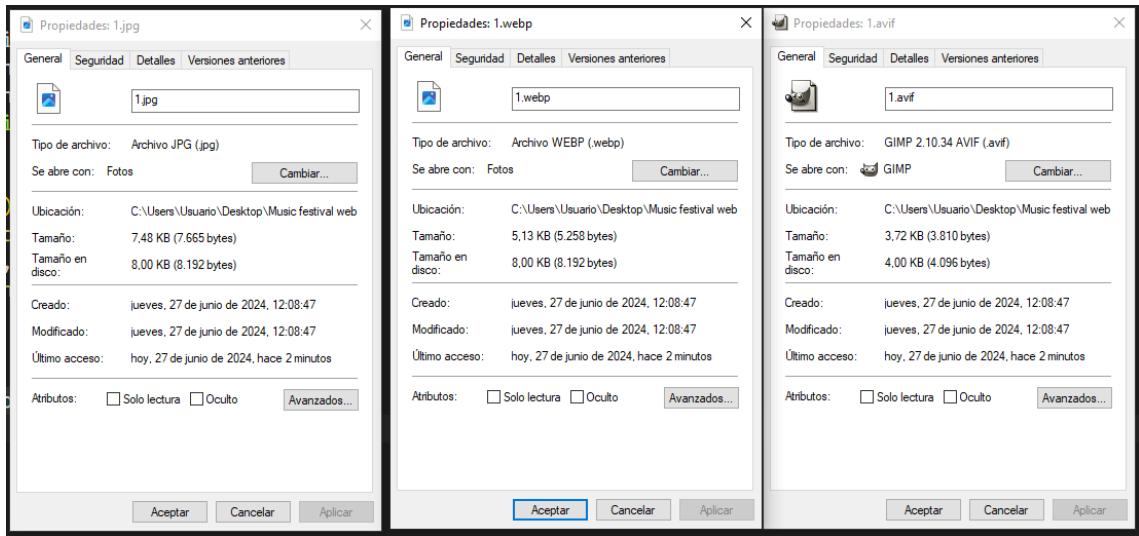
 92

```

Lo que hemos hecho es duplicar las líneas que ya teníamos para otros formatos y modificarlas para generar imágenes también en el nuevo formato .avif. Nota: quality: 80 indica la calidad de las imágenes, siendo 100 la misma calidad y por tanto 80 es con una calidad 20 puntos inferior.

Por alguna razón si mantenemos (option) para el formato .avif genera un archivo de mayor peso, por lo que lo quitamos de la línea.

Usuarios	Usuario	Escritorio	Music festival web project	build	img	gallery	thumb
Nombre		Fecha		Tipo	Tamaño		Etiquetas
1.avif		27/06/2024 12:08		GIMP 2.10.34 AVIF	4 KB		
1.jpg		27/06/2024 12:08		Archivo JPG	8 KB		
1.webp		27/06/2024 12:08		Archivo WEBP	6 KB		
2.avif		27/06/2024 12:08		GIMP 2.10.34 AVIF	6 KB		
2.jpg		27/06/2024 12:08		Archivo JPG	10 KB		
2.webp		27/06/2024 12:08		Archivo WEBP	7 KB		
3.avif		27/06/2024 12:08		GIMP 2.10.34 AVIF	7 KB		
3.jpg		27/06/2024 12:08		Archivo JPG	12 KB		
3.webp		27/06/2024 12:08		Archivo WEBP	10 KB		
4.avif		27/06/2024 12:08		GIMP 2.10.34 AVIF	7 KB		
4.jpg		27/06/2024 12:08		Archivo JPG	12 KB		
4.webp		27/06/2024 12:08		Archivo WEBP	10 KB		
5.avif		27/06/2024 12:08		GIMP 2.10.34 AVIF	7 KB		
5.jpg		27/06/2024 12:08		Archivo JPG	11 KB		
5.webp		27/06/2024 12:08		Archivo WEBP	9 KB		
6.avif		27/06/2024 12:08		GIMP 2.10.34 AVIF	3 KB		
6.jpg		27/06/2024 12:08		Archivo JPG	6 KB		
6.webp		27/06/2024 12:08		Archivo WEBP	4 KB		
7.avif		27/06/2024 12:08		GIMP 2.10.34 AVIF	6 KB		
7.jpg		27/06/2024 12:08		Archivo JPG	10 KB		
7.webp		27/06/2024 12:08		Archivo WEBP	8 KB		
8.avif		27/06/2024 12:08		GIMP 2.10.34 AVIF	7 KB		
8.jpg		27/06/2024 12:08		Archivo JPG	11 KB		
8.webp		27/06/2024 12:08		Archivo WEBP	9 KB		
9.avif		27/06/2024 12:08		GIMP 2.10.34 AVIF	6 KB		
9.jpg		27/06/2024 12:08		Archivo JPG	10 KB		
9.webp		27/06/2024 12:08		Archivo WEBP	8 KB		
10.avif		27/06/2024 12:08		GIMP 2.10.34 AVIF	7 KB		
10.jpg		27/06/2024 12:08		Archivo JPG	12 KB		
10.webp		27/06/2024 12:08		Archivo WEBP	9 KB		
11.avif		27/06/2024 12:08		GIMP 2.10.34 AVIF	5 KB		
11.jpg		27/06/2024 12:08		Archivo JPG	9 KB		
11.webp		27/06/2024 12:08		Archivo WEBP	7 KB		



Ahora tendremos que modificar el .html para que cargue las imágenes en los formatos de menos pesados (.avif) a mas pesados (.jpg) según pueda soportarlos o no los exploradores de los usuarios.

```

index.html <-
<html lang="es">
  <body>
    <section class="sobre-festival">
      <div class="imagen">
        <picture>
          <source srcset="build/img/imagen_dj.avif" type="image/avif" />
          <source srcset="build/img/imagen_dj.webp" type="image/webp" />
          
        </picture>
      </div>
    </section>
  </body>
</html>

```

Una cosa a tener en cuenta es que hay que meterlo todo en una etiqueta `<picture>` y que en el caso de .jpg, no es necesario crear otra línea `<source>` ya que se ejecuta en la etiqueta ``.

También es muy relevante que no se añade width ni height a las características en los casos de .avif y .webp porque las toma de la etiqueta ``, ya que es `` quien ejecutará siempre el renderizado, sea cual sea el formato que sea elegido/soportado por el navegador del usuario, en otras palabras, los `<source>` de .avif y de .webp únicamente sirven para indicar la fuente de la imagen y la etiqueta de identificación del formato de imagen por el navegador.

Ahora debemos modificar la carga de las imágenes de la galería:

Anteriormente estaba así:

```
index.html gulpfile.js app.js
src > js > app.js > ...
21
22 function crearGaleria() {
23     const CANTIDAD_IMAGENES = 16;
24     const galeria = document.querySelector(".galeria-imagenes");
25
26     for (let i = 1; i <= CANTIDAD_IMAGENES; i++) {
27         /* La convención dice que debe ponerse en mayúscula, en lugar de img, será IMG, en lugar de div, será DIV */
28         const imagen = document.createElement("IMG");
29         imagen.loading = "lazy";
30         imagen.width = "300";
31         imagen.height = "200";
32         imagen.src = `src/img/gallery/full/${i}.jpg`;
33         imagen.alt = "Imagen de Galería";
34
35         //Event Handler: Esto es detectar y responder a la interacción de un usuario, por ejemplo por su click.
36         imagen.onclick = function () {
37             mostrarImagen(i);
38         };
39
40         galeria.appendChild(imagen);
41     }
42 }
43
```

Lo modificamos a:

```
index.html gulpfile.js app.js
src > js > app.js > mostrarImagen
21
22 function crearGaleria() {
23     const CANTIDAD_IMAGENES = 16;
24     const galeria = document.querySelector(".galeria-imagenes");
25
26     for (let i = 1; i <= CANTIDAD_IMAGENES; i++) {
27         /* La convención dice que debe ponerse en mayúscula, en lugar de img, será IMG, en lugar de div, será DIV. */
28         const imagen = document.createElement("PICTURE");
29         imagen.innerHTML =
30             `
31             
32             ![Imagen galería](build/img/gallery/thumb/${i}.jpg)`;
33     }
34
35     //Event Handler: Esto es detectar y responder a la interacción de un usuario, por ejemplo por su click.
36     imagen.onclick = function () {
37         mostrarImagen(i);
38     };
39
40     galeria.appendChild(imagen);
41 }
42 }
```

Y también con la función mostrarImagen:

Anteriormente era así:

The screenshot shows a code editor with several tabs at the top: 'index.html', 'gulpfile.js', 'app.js', and 'crearGaleria'. The 'app.js' tab is active, displaying the following JavaScript code:

```
src > js > app.js > crearGaleria
43
44 function mostrarImagen(i) {
45     const imagen = document.createElement("IMG");
46     imagen.src = `src/img/gallery/full/${i}.jpg`;
47     imagen.alt = "Imagen de Galería";
48
49 //Generar Modal:
50 //En esta parte se generará el div.
51 const modal = document.createElement("DIV");
52 modal.classList.add("modal");
53
54 //!Este evento onclick a diferencia del de mostrarImagen no lleva f
55 modal.onclick = cerrarModal;
56
57 //Botón cerrar modal
58 const cerrarModalBtn = document.createElement("BUTTON");
59 cerrarModalBtn.textContent = "X";
60 cerrarModalBtn.classList.add("btn-cerrar");
61 cerrarModalBtn.onclick = cerrarModal;
62
63 modal.appendChild(imagen);
64 modal.appendChild(cerrarModalBtn);
65
66 //Agregar al HTML: Seleccióno el body y añado el modal en el body.
67 const body = document.querySelector("body");
68 body.classList.add("overflow-hidden");
69 body.appendChild(modal);
70 }
```

Y ahora es así:

```
index.html    gulpfile.js    app.js    navegacionFija
build > js > app.js > navegacionFija
44 function mostrarImagen(i) {
45     const imagen = document.createElement("PICTURE");
46     imagen.innerHTML = `
47         <source srcset="build/img/gallery/full/${i}.avif" type="image/avif">
48         <source srcset="build/img/gallery/full/${i}.webp" type="image/webp">
49         
50     `;
51
52     //Generar Modal:
53     //En esta parte se generará el div.
54     const modal = document.createElement("DIV");
55     modal.classList.add("modal");
56
57     //Este evento onclick a diferencia del de mostrarImagen no lleva function porque no se le pasa ningún parámetro
58     modal.onclick = cerrarModal;
59
60     //Botón cerrar modal
61     const cerrarModalBtn = document.createElement("BUTTON");
62     cerrarModalBtn.textContent = "X";
63     cerrarModalBtn.classList.add("btn-cerrar");
64     cerrarModalBtn.onclick = cerrarModal;
65
66     modal.appendChild(imagen);
67     modal.appendChild(cerrarModalBtn);
68
69     //Agregar al HTML: Selecciono el body y añado el modal en el body.
70     const body = document.querySelector("body");
71     body.classList.add("overflow-hidden");
72     body.appendChild(modal);
73 }
74
```

Nota: Las únicas partes modificadas son las marcadas en el corchete rojo, se cambió **IMG** por **PICTURE** y la carpeta **thumb** por **full**.