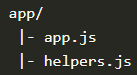
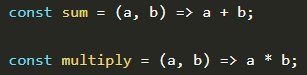
WEBPACK

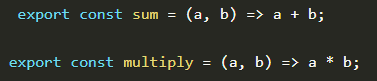
En JavaScript, un módulo es una unidad independiente de código que puede ser reutilizado. Los módulos permiten exponer ciertas partes del código que serán usadas por otros módulos. Esto nos da la flexibilidad suficiente para mantener nuestro código mejor organizado en múltiples scripts.



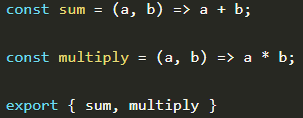
En helpers.js podemos tener algunas funciones que podrán ser usadas en otras partes del código:



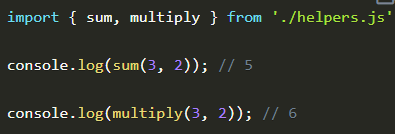
Podemos exportar cada módulo de manera independiente usando el keyword export.



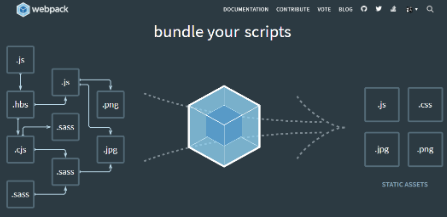
O bien en una sola sentencia.



Para usar estas funciones en app.js usamos el keyword import junto con from para definir la ruta del archivo.



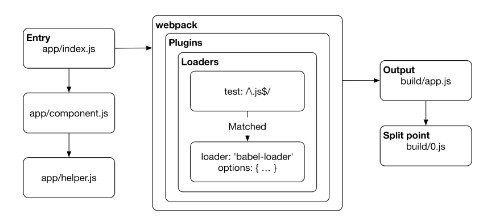
WEBPACK



Webpack es una herramienta muy usada en el desarrollo de aplicaciones en JavaScript modernas. Después de procesar la aplicación, webpack genera internamente un grafo de dependencias lo que le permite generar uno o más bundles. Este bundle contiene el código optimizado de todos los módulos y dependencias de tu aplicación.

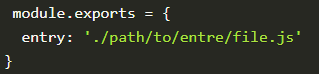
Algunos navegadores no soportan la nueva sintaxis de ES6 como import y export para el uso de módulos. Detrás de escenas, Webpack transpila el código para que el uso de módulos sea soportado en todos los navegadores. Es importante tomar en cuenta que Webpack solo modifica sentencias import y export, para hacer uso de cualquier otra característica de ES6 es necesario usar un transpilador como Babel.

ENTRY / OUTPUT



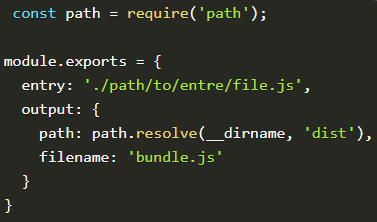
ENTRY

El punto de entrada le dice a webpack cuál módulo debe ser usado para comenzar a crear el grafo de dependencias. Webpack se encargará de buscar todos los módulos y librerías que utilizan dicho punto de entrada. El valor por default es ./src/index.js, pero podemos especificar una ruta diferente.



OUTPUT

La salida especifica donde deben colocarse los bundles resultantes y cómo se deben llamar los archivos. Por default el bundle principal es ./dist/main.js. Para cambiar este valor agregamos una propiedad output cuyo valor es un objeto que nos permite definir la ruta y el nombre del archivo.



A diferencia del entry, cuando estamos definiendo la ruta del output necesitamos una ruta absoluta, por ello usamos path que es un módulo de Node.js. El método path.resolve() genera una ruta con los argumentos que le pasamos, \_\_dirname es una variable cuyo valor es la ruta absoluta, dist es la carpeta donde queremos colocar el bundle final.

LOADERS

Webpack solamente entiende archivos de JavaScript y JSON. Usando loaders webpack puede procesar otro tipo de archivos y convertirlos en módulos que serán agregados al grafo de dependencias.

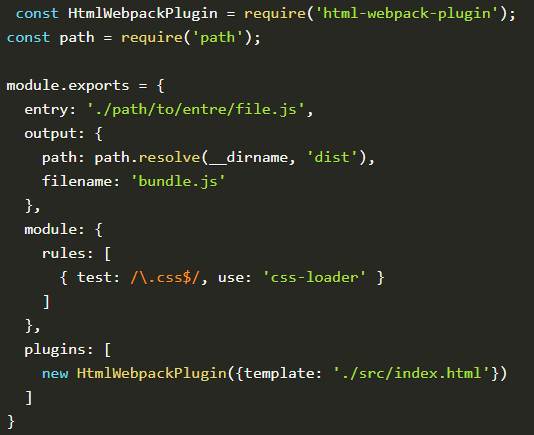


Cada loader tiene dos propiedades, la primera es test, aquí definimos con una expresión regular el tipo de archivos que deseamos procesar, con la segunda propiedad use indicamos el tipo de loader que debe ser usado. En el ejemplo anterior estamos usando raw-loader para procesar todos los archivos txt del proyecto.

Los loaders deben ser instalados previamente: npm install --save-dev raw-loader

PLUGINS

Los loaders nos ayudan a transformar ciertos tipos de módulos, mientras que los plugins permiten una amplia gama de tareas como optimización de los bundles, manejo de assets, inyección de variables de entorno, etc. Para usar un plugin debemos importarlo y agregarlo al arreglo plugins.



La mayoría de los plugins pueden ser configurados. En este ejemplo, html-webpack-plugin genera un archivo html listo para ser usado inyectando automáticamente los bundles generados por webpack.

Los plugins deben ser instalados previamente: npm install --save-dev html-webpack-plugin

WEBPACK POSTWORK

Uno de los beneficios de crear un grafo de dependencias de la manera que lo hace webpack es que cada módulo determina de manera explícita, así se puede evitar incluir en el bundle final módulos que no estén siendo usados.

Webpack nos permite incluir cualquier tipo de archivo que no sea JavaScript mediante loaders. Esto implica que obtenemos los mismos beneficios mencionados anteriormente con todo lo necesario para construir un sitio o aplicación web.

Ya vimos cómo usar css-loader y style-loader para trabajar con estilos CSS. Otro loader bastante útil es file-loader, que nos permite integrar imágenes de manera sencilla.

npm install --save-dev file-loader

Después de instalar el loader podemos agregarlo en webpack.config.js

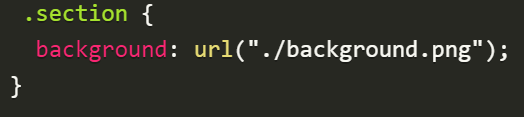


Con esto podemos import MyImage from './my-image.png' para que la imagen sea procesada y agregada en dist/, la variable MyImage que acabamos de crear contiene la url final de la imagen después de ser procesada. Si a la par estamos usando ccs-loader el mismo proceso ocurre cuando hacemos algo como url('./my-image.png') desde CSS. El loader reconoce que se trata de un archivo local y reemplaza './my-image.png' por la url final de la imagen.

USANDO IMÁGENES DESDE JAVASCRIPT



USANDO IMAGEN DESDE CSS



Después de generar el build de la aplicación la imagen debería de haber cambiado el nombre por algo similar a cda45674bb47cbe1ca4b20303.png, esto significa que webpack encontró y procesó la imagen correctamente.