NODE:

Permite correr código de js en el lado del servidor. Por lo que tanto backend como frontend se pueden hacer con JS. También permite desarrollar localmente.

WEBPACK:

Es un empaquetador de módulos. Permite automatizar tareas. Puede ser un poco difícil de configurar inicialmente, pero una vez configurado puedo subirlo a github, descargarlo y ya empezar a trabajar con él en pocos segundos.

Algunas cosas que puedo hacer con webpack:

-Pasar de una versión de JS más nueva a una más antigüa. Puedo trabajar con tranquilidad con versiones nuevas.

-Compilar SASS a CSS

-Compilar de Typescript a JS

-Montar servidores de desarrollo y pruebas

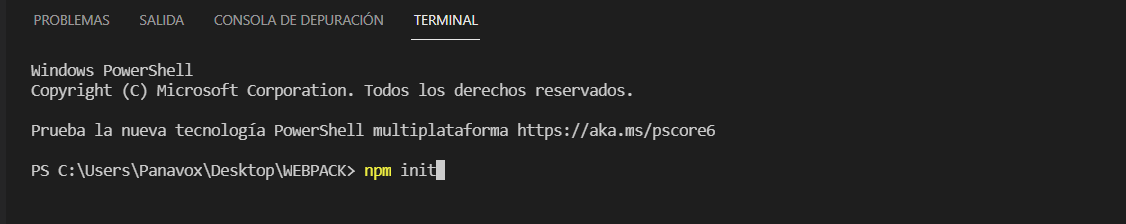
-Gestionar dependencias

-Cargar módulos (Segmentar aplicación para que quede en diferentes archivos, más ordenado)

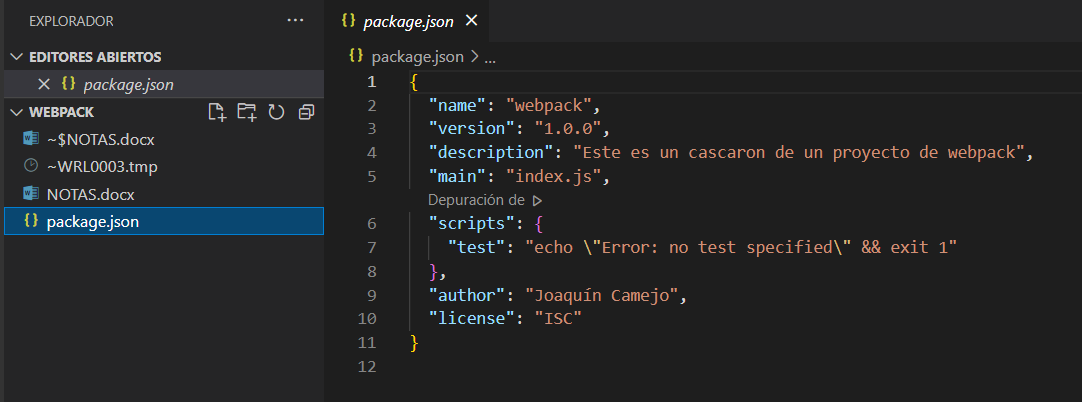
-Minimizar código.

CREANDO UN PROYECTO CON NODE

En la terminal de la carpeta de mi proyecto uso la sentencia **npm init**. Esto nos guía para la creación de nuestro archivo package.json. Este contiene cómo funciona la aplicación y las dependencias necesarias para pasarlas a producción.

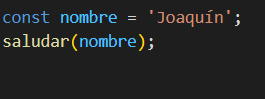


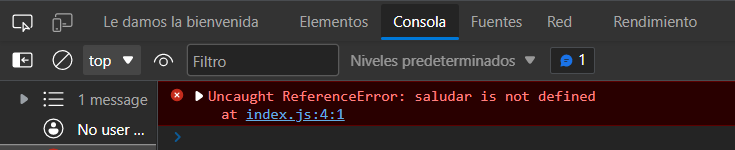
El archivo que se genera una vez iniciado node.



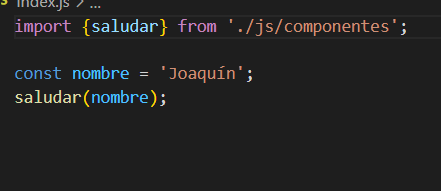
La necesidad de trabajar con diferentes archivos con sus funcionalidades. Importar módulos, librerías en mi proyecto.

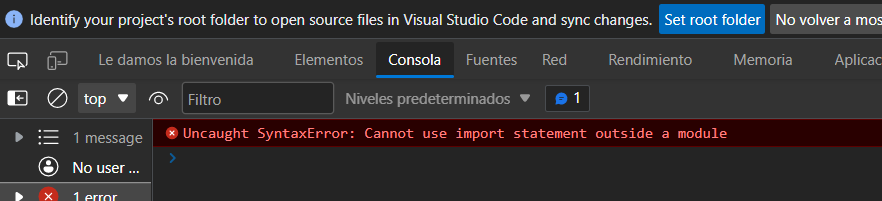
En primer lugar, el problema es que en mi index.js no importo la función de él archivo correspondiente.





Una vez importado el modulo se me genera otro problema, en el cual el navegador no me permite usar está importación si estoy fuera del modulo.



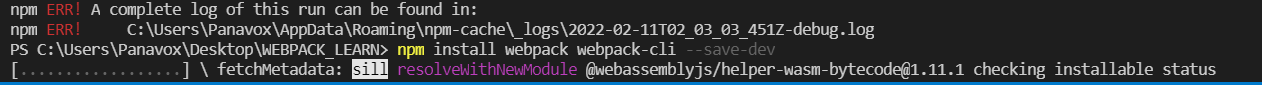


REACT Y ANGULAR TIENEN INCORPORADO WEBPACK POR DEFECTO.

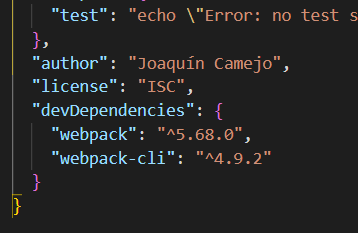
INSTALACIÓN Y CONFIGURACIÓN DE WEBPACK

Se instala webpack una vez inicializado el proyecto node con la sentencia

npm install webpack webpack-cli --save-dev



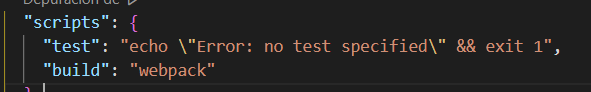
Queda guardado en mi package.json en las dependencias de desarrollo.



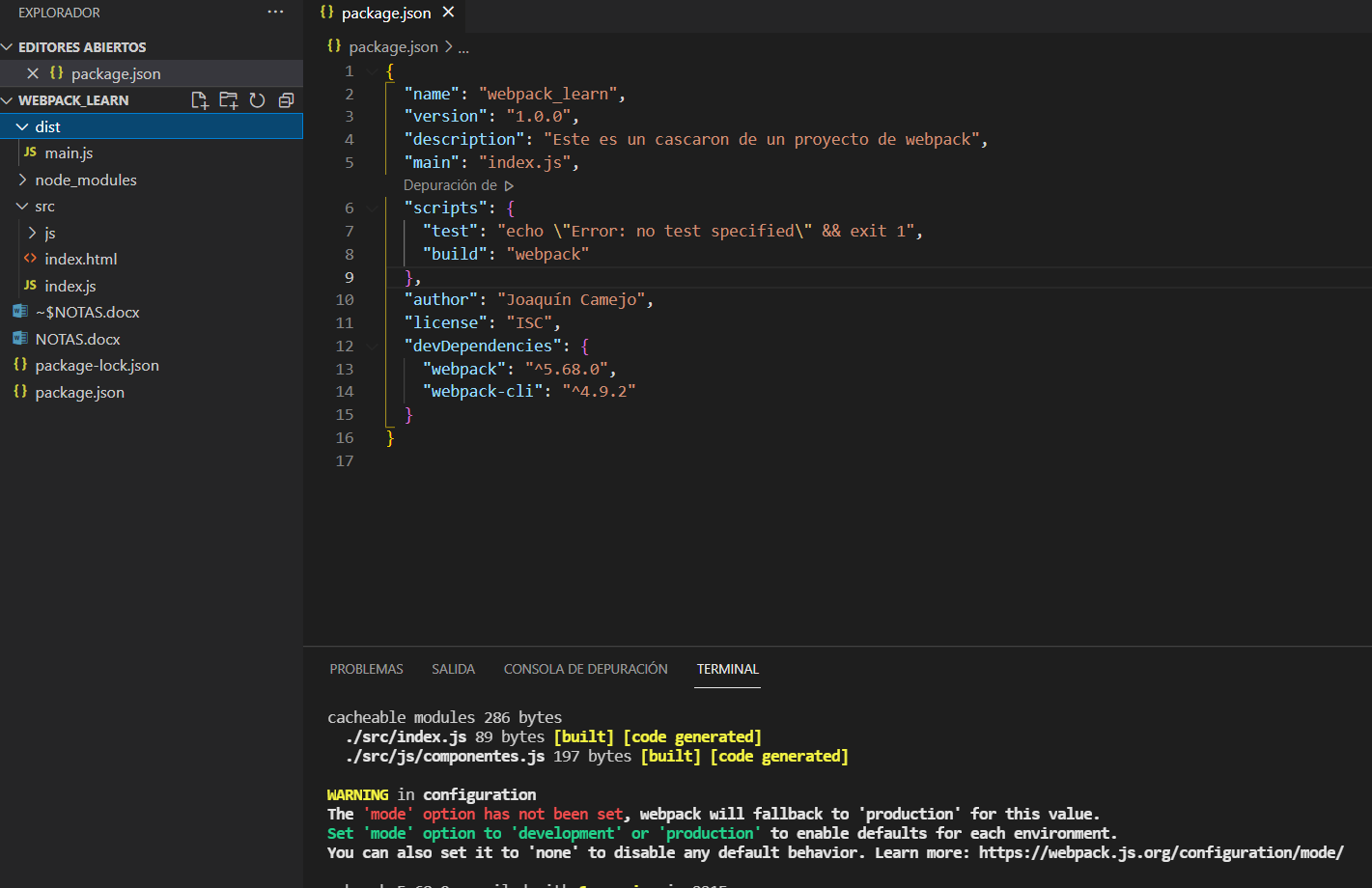
VOLVIENDO AL PROBLEMA DE LOS MODULOS

Anteriormente solo instalamos webpack, pero aún no hemos empezado a trabajar con él. En la parte de scripts (en mi archivo package.json) puedo ejecutar ciertos comandos.

El comando “build” va a ejecutarse y ejecutar el comando webpack que instalamos.

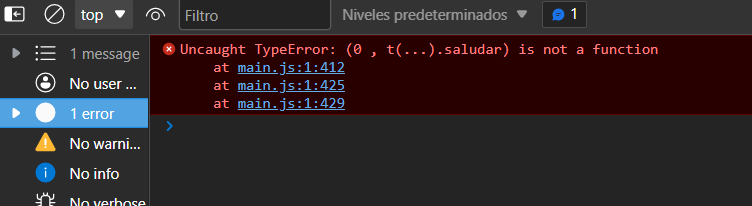


Luego ejecutamos el comando npm run build

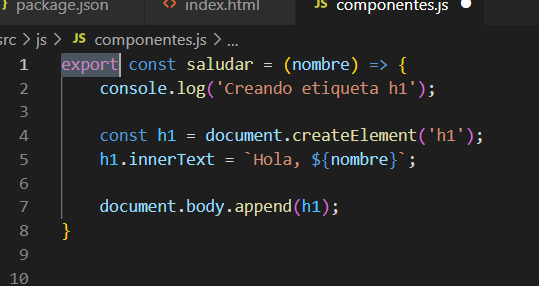


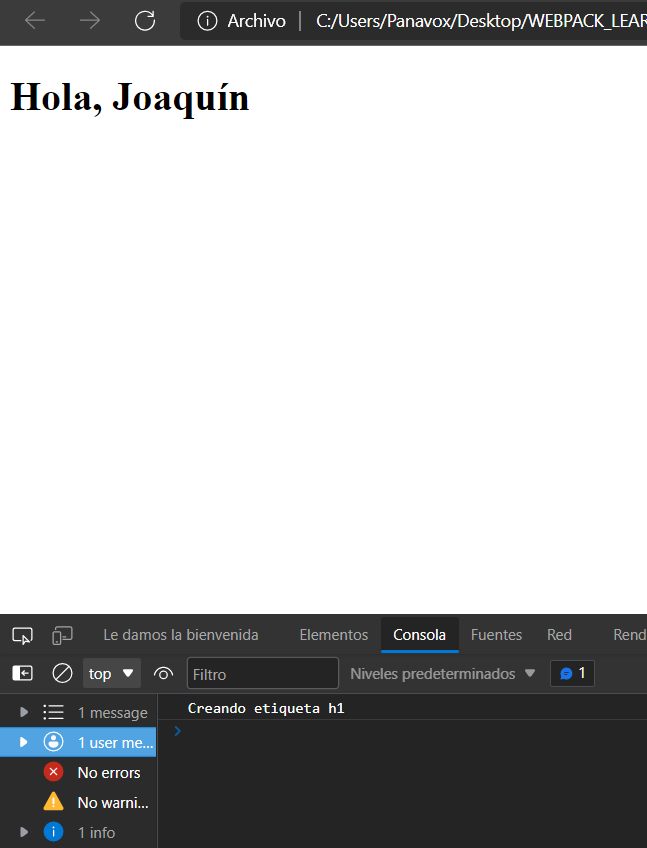
Se crea automáticamente una carpeta dist (distribution) con un archivo main.js.

Una vez modularizada mi aplicación, en el index.html tengo que hacer referencia al main.js. Hecho esto, se me presentará otro problema.



Esto es porque si bien importe la clase desde mi index.js, en mi componentes.js no realicé la exportación de la función. Cada vez que voy a importar una función es necesario exportarla.

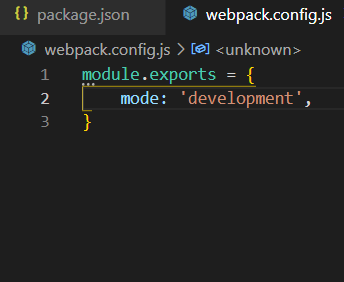




Listo el pollo, pelada la gallina. Para importar tenemos que declarar la exportación (export).

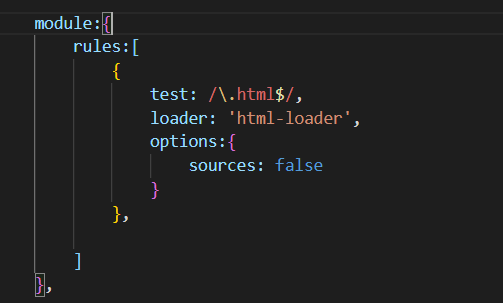
ARCHIVO DE CONFIGURACIÓN DE WEBPACK

Creo un archivo webpack.config.js



Una vez creado el archivo exporto determinados modulos.

En primer lugar tenemos el mode, que puede tomar dos valores: production o development. Por defecto webpack tiene el modo de producción en el cual optimiza(minimiza) el código de nuestra aplicación.

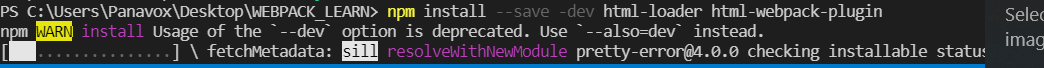


Estas: mode, modules, optimization, plugins; son las opciones más comunes que podemos encontrar en el archivo de configuración de webpack. Modules es module (por error lo puse en plural)

HTML LOADER Y HTML WEBPACK PLUGIN

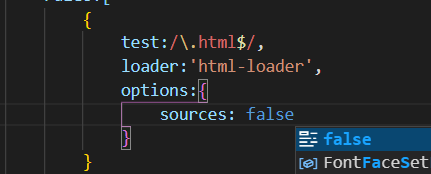
HTML loader: Carga el HTML. Html WEBPACK plugin: Ayuda con la inyección de nuestro html en el bundle final, del archivo js en el archivo html.

1. Instalar las dependencias para desarrollo.



npm install --save-dev html-loader; npm install –save –dev html-webpack-plugin.

1. Trasladar mis archivos html a la carpeta dist

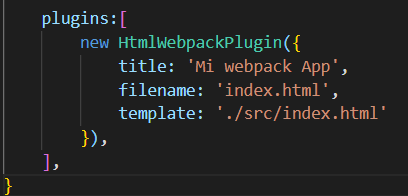


En mi archivo de configuración en modules – rules, agrego el html loader. En primer lugar mediante la expresión regular .html$ busco todos los archivos con terminación html de mi proyecto. En segundo con el html loader los cargo a la carpeta dist. Con las options de source en false, cuando se utiliza alguna imagen por ejemplo también la mueve.

1. Hacemos uso del plugin
2. Importar el html webpack plugin



1. Creo una instancia de mi plugin, y lo configuro. En la configuración puedo elegir titulo, nombre del archivo en la carpeta dist, y el archivo en que va a estar basado. Webpack automáticamente va a inyectar el js, por lo que puedo borrar el script del html en mi src.



WEBPACK SERVER

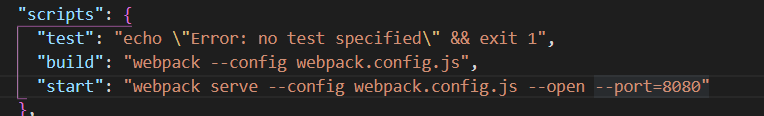
El webpack server, nos permite montar un **servidor de desarrollo** en el cual podemos constatar los **cambios** que vamos haciendo en nuestra aplicación en **tiempo real**. Aunque no esté generado el build, la aplicación se encuentra **trabajando en memoria**.

1-Instalación

npm install –sabe-dev webpack-dev-server

2-configurar los scripts del package.json para servir mi aplicación en un puerto específico (--port) y que se abra una vez servida (--open).

Lo hago con el comando start, tal como se muestra en la imagen.



3-Correrlo en el servidor npm start;

IMPORTAR ESTILOS Y CSS A JS CON WEBPACK

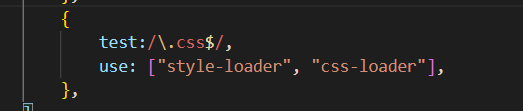
Sirve para que puedamos usar estilos css en nuestra aplicación, lo que hacen los paquetes que vamos a instalar a continuación es transformar en script nuestros archivos con terminación css.

1-Instalar style-loader y css-loader

npm install --save-dev style-loader css-loader

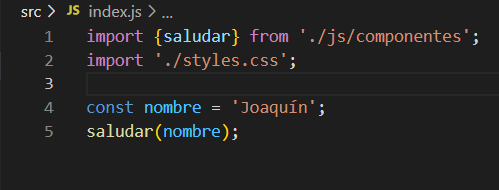
2-Configurar mi archivo webpack.config.js

En module|rules establezco la siguiente configuración



En el test son los archivos que tienen que tomaré, en este caso son los que tienen terminación css.

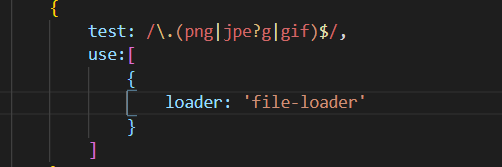
3 Importar en el index.js la hoja de estilos.



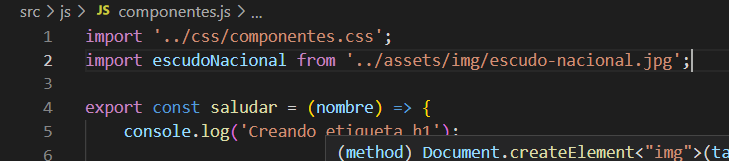
MANEJO DE IMÁGENES

Este paquete sirve para que pueda hacer uso de archivos (generalmente imágenes) en mi aplicación.

1. Intalación npm install –save-dev file-loader
2. Configurar modules|rules.



1. Importar desde mi componentes.js la imagen que se encuentra en el directorio source/assets/img/



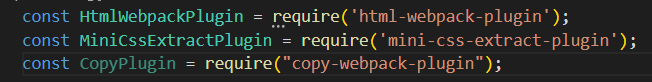
COPY PLUGIN

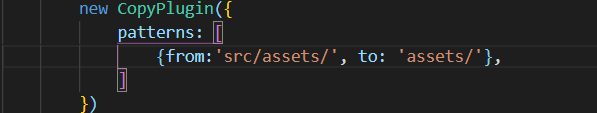
Lo usamos para copiar en mi carpeta dist, directorios estaticos. Con esto podríamos hacer referencia al source desde mi archivo html, anteriormente no podíamos porque una vez generado el dist, el href no encontraba el directorio ya que en el dist no existían los assets. Con esto se copiará la carpeta assets en mi dist y desde el html encontrará el source.

1-Instalación copy-webpack-plugin



2- Configurarlo en mi webpack.config.js



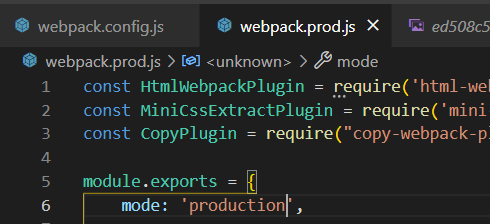


Como corolario, una vez que haga el npm run build, se creará la carpeta assets, la cual copia todo el contenido que se encuentra dentro del from y lo guarda dentro suyo (assets-dist-).

WEBPACK PRODUCTION MODE

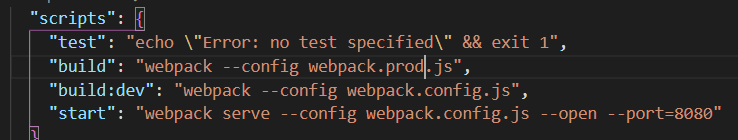
El modo producción sirve para optimizar el código que tenemos, eliminar espacios, adaptarlo para otros navegadores. Es lo que el usuario va a ver, por lo tanto no nos interesa que vea de donde sale cada cosa, ni nuestros comentarios a la hora del desarrollo.

1. Creo una copia de mi webpack. config.js pero con el nombre webpack.prod.js

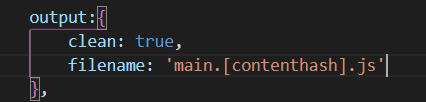


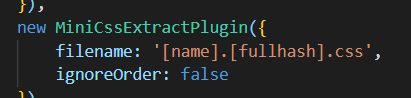
En el mode le cambio a production.

1. En el package.json genero dos builds, uno para desarrollo y otro para producción.



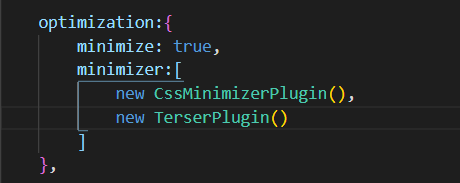
1. Editamos los nombres de salidas que queremos mostrar en el webpack.prod.js





1. Nuestro archivo css no se ha minificado, para esto descargamos y configuramos los siguientes plugins.





NOTA: Las optimizaciones van arriba de los plugins.

INSTALACIÓN Y CONFIGURACIÓN DE BABEL

Esto lo hacemos para que nuestra aplicación sea compatible con navegadores web viejos, que usen versiones más antigüas de JS. Considerando que por ejemplo const y let son bastantes recientes para declarar variables, y navegadores con versiones viejas de JS podrían no reconocer nuestro código.  
Para solucionar esto llega BABEL a nuestro rescate para hacer compatible nuestro código con versiones más viejas de JS.

1-Instalación

npm install --save-dev babel-loader @babel/core

2-Configurando babel en el config.prod.js



3-Adicionalmente tenemos que instalar y configurar un preset de babel

1. npm install @babel/preset-env --save-dev
2. Creo un archivo babel.config.json y pego el siguiente código.

