

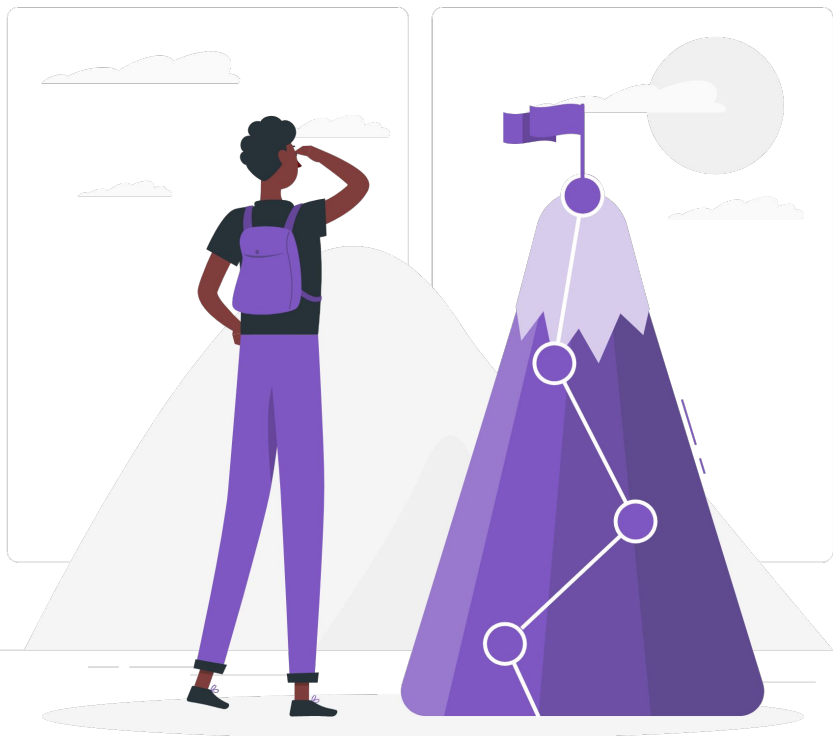
Webpack & pre-processors



webpack +

Sass

dev.f **BABEL**
desarrollamos(personas);



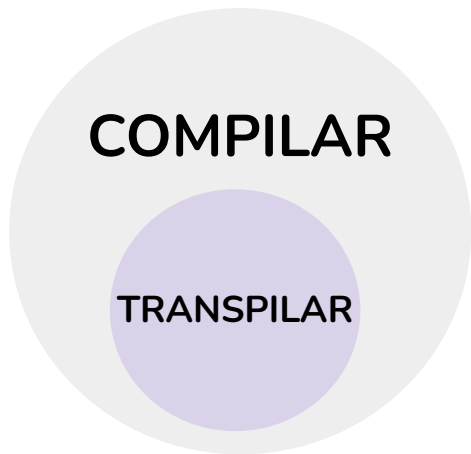
Objetivos

- Entender el concepto de transpilar.
- Comprender que son los pre-procesadores y como nos benefician.
- Entender el papel de **Babel** para la compatibilidad del JavaScript moderno.
- Comprender qué es Webpack, cómo funciona.
- Comenzar la labor de configuración básica de Webpack.

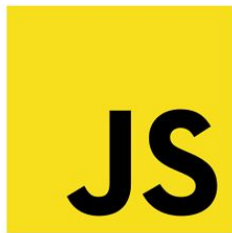
Pre-processors

dev.*f*
desarrollamos(personas);

dev



Transpilación



Compilación



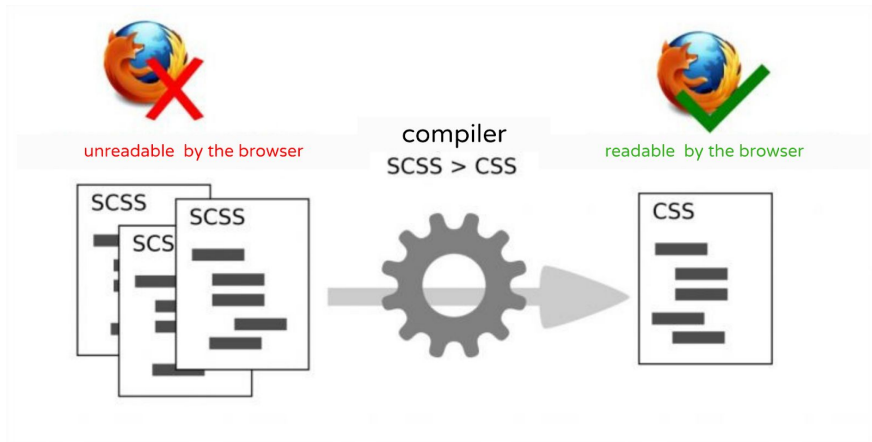
```
Code:
0:  iconst_2
1:  istore_1
2:  iload_1
3:  sipush 1000
6:  if_icmpge      44
9:  iconst_2
10: istore_2
11: iload_2
12: iload_1
13: if_icmpge      31
16: iload_1
17: iload_2
18: irem           # remainder
19: ifne          25
22: goto          38
```

Transpilar

Muchos conocemos el término “**Compilar**”, pero no tanto el de **transpilar**.

La transpilación es un caso particular de la compilación. Todo transpilador es también un compilador pero al revés no siempre aplica.

Si el compilador traduce código entre dos lenguajes que están al mismo nivel de abstracción entonces, estamos ante un **transpilador**. Si traduce código entre lenguajes de diferente nivel de abstracción (típicamente de más alto a más bajo nivel) entonces no lo es.

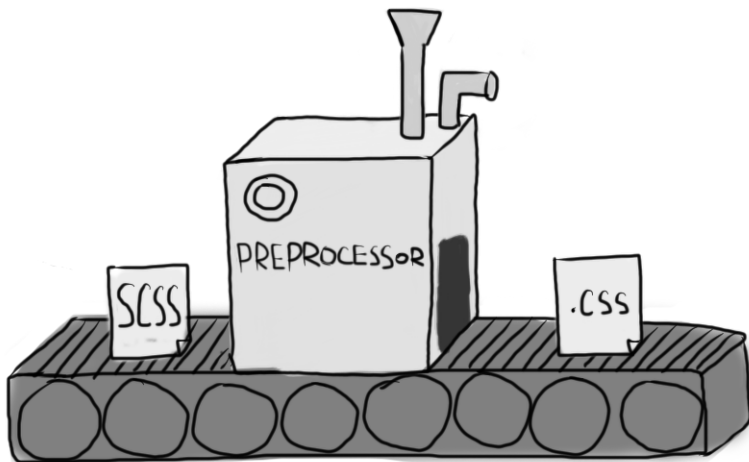


¿Que es un pre-processor?

En programación, **es una herramienta que transpila código** (“Traduce” código de una sintaxis definida a otra) lo cual **le permite optimizar y estandarizar dicho código**.

Son:

- Herramientas altamente ocupados en la industria.
- Proveen de mejores prácticas, shortcuts y facilidades al desarrollador.



pre-processor más conocidos

JS

BABEL

TS

 flow


CoffeeScript

CSS

Sass

{less}

stylus

HTML



pug

Haml



dev.f



```
// Babel Input: ES2015 arrow function  
[1, 2, 3].map( (n) => n+1 );
```

```
// Babel Output: ES5 equivalent  
[1, 2 ,3].map(function(n) {  
  return n+1;  
});
```

BABEL

Babel es una cadena de herramientas que **se utiliza principalmente para convertir el código ECMAScript 2015+ en una versión retrocompatible de JavaScript** en navegadores o entornos actuales y antiguos. Estas son las principales cosas que Babel puede hacer:

- Transformar la sintaxis
- Características de Polyfill (compatibilidad) que faltan en su entorno de destino.
- Transformaciones de código fuente.



SASS

- Sass son las siglas de Syntactically Awesome Stylesheet.
- Sass es una extensión de CSS
- Sass es completamente compatible con todas las versiones de CSS
- Sass reduce la repetición de CSS y por lo tanto ahorra tiempo
- Los archivos de css cada vez son más grandes, más complejas y más difíciles de mantener. Aquí es donde puede ayudar un preprocesador de CSS.
- Sass te permite usar funciones que no existen en CSS, como variables, reglas anidadas, mixins, importaciones, herencia, funciones integradas y otras cosas.

SCSS

```
1  section {  
2    height: 100px;  
3    width: 100px;  
4  
5    .class-one {  
6      height: 50px;  
7      width: 50px;  
8  
9      .button {  
10       color: #074e68;  
11     }  
12   }  
13 }
```

CSS

```
1  section {  
2    height: 100px;  
3    width: 100px;  
4  }  
5  
6  section .class-one {  
7    height: 50px;  
8    width: 50px;  
9  }  
10  
11 section .class-one .button {  
12   color: #074e68;  
13 }
```


Webpack

dev.*f*
desarrollamos(personas);

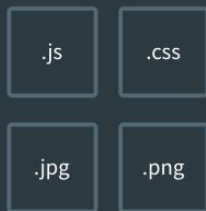
dev



Modules with dependencies



webpack



static assets

¿QUÉ ES?

- **Webpack es un “constructor” de paquetes.**
- Webpack puede encargarse de la agrupación de archivos.
- Webpack puede tener un corredor de tareas.
- Existen complementos de paquetes web desarrollados por la comunidad.
- Estos complementos se utilizan para realizar tareas que generalmente se realizan fuera del paquete web, como limpiar el directorio de compilación o implementar la compilación.
- **Todos los grandes Frameworks de JS lo usan.**
- **Es el estándar para las aplicaciones de lado del cliente.**
- Te permite tener el control sobre diferentes assets (JS,CSS,HTML,imágenes,etc)

Comenzando con Webpack

Configuración de Entorno de Desarrollo





```
//Creamos una carpeta  
mkdir webpack-starter
```

```
//Entramos en la carpeta  
cd webpack-starter
```

```
//Creamos un nuevo proyecto de npm  
npm init -y
```

1. Creación del proyecto

Se asume que comenzaremos un nuevo proyecto que usará webpack, por lo tanto el primer paso consiste simplemente en crear una carpeta vacía, entrar en ella e inicializar npm.



```
//Instalación de Webpack y Webpack-cli  
npm i webpack webpack-cli --save-dev
```

package.json

05.React > 01.webpack-starter > package.json > ...

```
1 {  
2   "name": "01.webpack-starter",  
3   "version": "1.0.0",  
4   "description": "",  
5   "main": "index.js",  
6   "scripts": {  
7     "test": "echo \"Error: no test specified\" && exit 1"  
8   },  
9   "keywords": [],  
10  "author": "",  
11  "license": "ISC",  
12  "devDependencies": {  
13    "webpack": "^5.53.0",  
14    "webpack-cli": "^4.8.0"  
15  }  
16 }
```

2. Instalar webpack

Con npm podemos instalar dependencias para diversos entornos: producción, test, desarrollo, etc.

webpack y webpack-cli son dependencias que se usan solo en el desarrollo de la aplicación, por eso al añadir **--save-dev** le indicamos a npm que son dependencias de desarrollo y las coloca en el **package.json** como **devDependencies**.

3. Colocar el script para construir el proyecto (build)

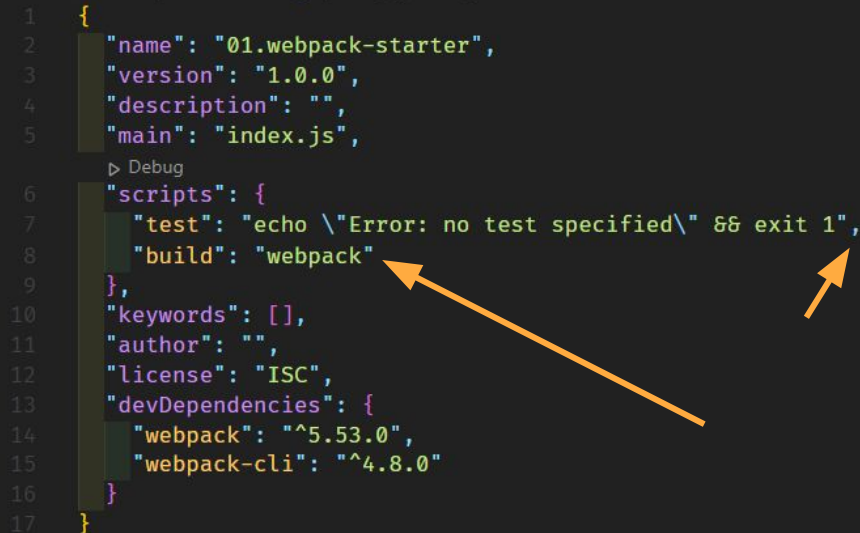
Editamos manualmente el archivo package.json y debajo de “test” en scripts, escribimos nuestro script de “compilación” personalizado, al cual llamaremos build y ejecutará webpack.

“build”: “webpack”

package.json X

05.React > 01.webpack-starter > package.json > ...

```
1  {
2    "name": "01.webpack-starter",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    > Debug
7    "scripts": {
8      "test": "echo \"Error: no test specified\" && exit 1",
9      "build": "webpack"
10   },
11   "keywords": [],
12   "author": "",
13   "license": "ISC",
14   "devDependencies": {
15     "webpack": "^5.53.0",
16     "webpack-cli": "^4.8.0"
17   }
18 }
```



Tip: Recuerda añadir una coma al final de test, para que no marque error de sintaxis.

4. Ejecutar nuestro script Build para transpilar el proyecto

En la consola, para que webpack haga su trabajo, debemos ejecutar el comando:

npm run build

Para que comience el proceso de transpilación del código de acuerdo a la configuración de webpack.



```
//Corremos nuestro script build  
//Que a su vez ejecuta webpack  
npm run build
```

WARNING in configuration

The 'mode' option has not been set, webpack will fallback to 'production' for this value.

Set 'mode' option to 'development' or 'production' to enable defaults for each environment.

You can also set it to 'none' to disable any default behavior. Learn more: <https://webpack.js.org/configuration/mode/>

ERROR in main

Module not found: Error: Can't resolve './src' in 'D:\warde\Documents\GitHub\2021\test-master-code-g9\05.React\01.webpack-starter'

resolve './src' in 'D:\warde\Documents\GitHub\2021\test-master-code-g9\05.React\01.webpack-starter'

using description file: D:\warde\Documents\GitHub\2021\test-master-code-g9\05.React\01.webpack-starter\package.json (relative path: .)

Field 'browser' doesn't contain a valid alias configuration

using description file: D:\warde\Documents\GitHub\2021\test-master-code-g9\05.React\01.webpack-starter\package.json (relative path: ./src)

no extension

Field 'browser' doesn't contain a valid alias configuration

D:\warde\Documents\GitHub\2021\test-master-code-g9\05.React\01.webpack-starter\src doesn't exist

.js

Field 'browser' doesn't contain a valid alias configuration

D:\warde\Documents\GitHub\2021\test-master-code-g9\05.React\01.webpack-starter\src.js doesn't exist

.json

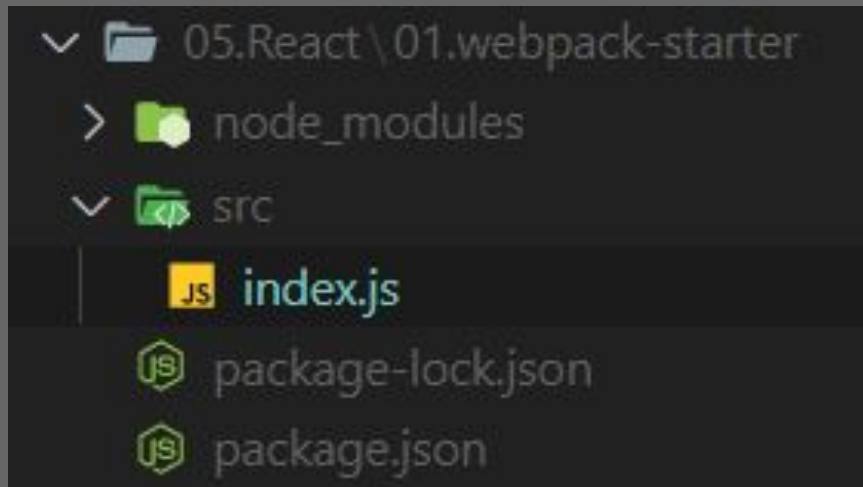
Field 'browser' doesn't contain a valid alias configuration

D:\warde\Documents\GitHub\2021\test-master-code-g9\05.React\01.webpack-starter\src.json doesn't exist

.wasm

Field 'browser' doesn't contain a valid alias configuration

D:\warde\Documents\GitHub\2021\test-master-code-g9\05.React\01.webpack-starter\src.wasm doesn't exist



Configuration

Out of the box, webpack won't require you to use a configuration file. However, it will assume the entry point of your project is `src/index.js` and will output the result in `dist/main.js` minified and optimized for production.

Referencia: <https://webpack.js.org/configuration/>

4a. Entrypoint: Archivo `index.js` en carpeta `./src`

El error anterior se debe a que Webpack necesita algo que se llama: **Entrypoint (o archivo fuente / raíz)**.

Por defecto, busca un archivo **`index.js`** que esté dentro de una carpeta **`./src`**

Por lo que crearemos dicha carpeta y archivo.



IMPORTANTE:

Dentro de la **carpeta src** es donde deberemos **escribir nuestro programa.**

De esta forma webpack leerá su contenido y transformara el código al momento de ejecutar build.

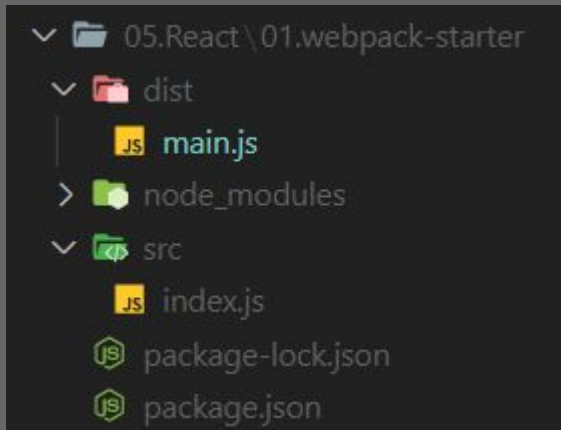
```
$ npm run build

> 01.webpack-starter@1.0.0 build D:\warde\Documents\GitHub\20
21\test-master-code-g9\05.React\01.webpack-starter
> webpack

asset main.js 0 bytes [emitted] [minimized] (name: main)
./src/index.js 1 bytes [built] [code generated]

WARNING in configuration
The 'mode' option has not been set, webpack will fallback to
'production' for this value.
Set 'mode' option to 'development' or 'production' to enable
defaults for each environment.
You can also set it to 'none' to disable any default behavior
. Learn more: https://webpack.js.org/configuration/mode/

webpack 5.53.0 compiled with 1 warning in 166 ms
```



4b. Ejecutar nuevamente build

Ahora que hemos creado la carpeta src y dentro del archivo index.js, procederemos a ejecutar nuevamente el comando:

npm run build

Notaremos que ahora nos crea una carpeta llamada **dist** y dentro un archivo **main.js**



IMPORTANTE:
Dentro de la carpeta dist
webpack colocará el
código optimizado.

La carpeta ***dist*** es la carpeta de ***Distribution***, es la que se sube al servidor para correr en producción. Contiene todos los archivos de nuestra aplicación procesados por webpack.

Entendiendo la optimización de Webpack

dev.*f*
desarrollamos(personas);

dev

Ejemplo de código

Por ejemplo, si dentro de la carpeta src tenemos los archivos: greeting.js e index.js con el siguiente contenido:

```
const greeting = (name) => {  
  return `Hola ${name}, saludos desde JS`;  
};  
  
//hago accesible greeting para otros archivos de JS  
export default greeting;
```



greeting.js

```
import greeting from './greeting';  
  
console.log(greeting('Cesar'));
```



index.js

Resultado de ejecutar webpack

Si ejecutamos nuevamente **npm run build**, veremos qué el archivo **/dist/main.js** tiene un contenido como el siguiente:



```
((()=>{var e={202:(e,o,r)=>{"use strict";r.r(o),r.d(o,{default:()=>t});const t=e⇒`Hola ${e}, saludos desde JS`},o={};function r(t){var n=o[t];if(void 0≠n)return n.exports;var a=o[t]={exports:
{}};return e[t](a,a.exports,r),a.exports}r.d=(e,o)⇒{for(var t in
o)r.o(o,t)&&!r.o(e,t)&&Object.defineProperty(e,t,{enumerable:!0,get:o[t]})},r.o=
(e,o)⇒Object.prototype.hasOwnProperty.call(e,o),r.r=e⇒{"undefined"≠typeof
Symbol&&Symbol.toStringTag&&Object.defineProperty(e,Symbol.toStringTag,
{value:"Module"}),Object.defineProperty(e,"__esModule",{value:!0})},(()=>{const
e=r(202);console.log(e("Cesar"))}})})();
```

Así luce un código optimizado por webpack.

Soporte de HTML con Webpack (Loaders + Plugins)

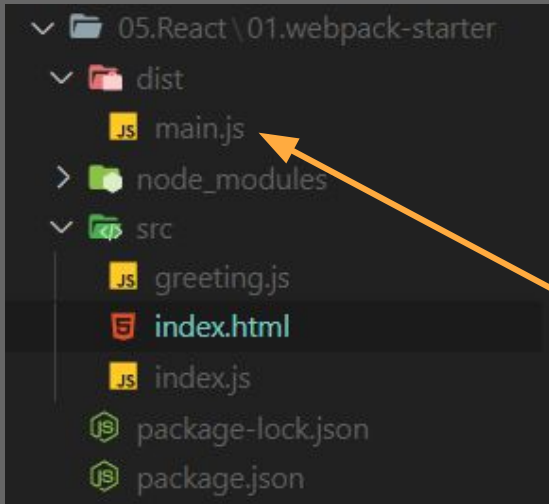
dev.*f*
desarrollamos(personas);

dev

Resultado de `npm run build`

```
asset main.js 609 bytes [compared for emit] [minimized]
runtime modules 670 bytes 3 modules
cacheable modules 233 bytes
  ./src/index.js 74 bytes [built] [code generated]
  ./src/greeting.js 159 bytes [built] [code generated]
```

Observamos que en `/dist` no existe ningún archivo HTML



HTML + Webpack

Por defecto, webpack solo maneja archivos JSON y JavaScript e ignora los archivos HTML/CSS, etc.

Por lo que si creamos un archivo `/src/index.html` y ejecutamos `npm run build`, webpack lo ignoraría.

Para hacer que el `index.html` también se cargue en `/dist`, es necesario usar un **Loader de Webpack**.



Los **Loaders** son una serie de **reglas** que le dicen a **webpack** **como generar los paquetes.**

Es necesario crear en la raíz del proyecto un archivo de configuración **webpack.config.js** donde podremos configurar los loaders.



webpack.config.js

```
module.exports = {  
  module: {  
    rules: [ //Aquí se cargan los Loaders de Webpack  
      {  
      }  
    ]  
  }  
}
```

Estructura de base del archivo de configuración
/webpack.config.js

Debemos crearlo y configurarlo de forma manual.

Loaders de Webpack

Un Loader indica a Webpack cómo tiene que transformar el código de un módulo concreto.

- Proveen de una forma de manejar los pasos de construcción
- Los Loaders pueden transformar ficheros en otro lenguaje a JavaScript.
- Se pueden cargar imágenes como URLs, generando un string en Base64.
- Por defecto Webpack va a procesar archivos JavaScript y archivos JSON.
- Los Loaders realizan muchas más funciones, porque existe gran cantidad de ellos, y cada uno hace una cosa distinta.



Los **plugins** agregan funcionalidad extra.

Todos los **plugins** dependen de un loader para funcionar.

Por el contrario, los loaders pueden funcionar por sí solos sin plugins.



webpack.config.js

```
module.exports = {  
  module: {  
    rules: [ //Aquí se cargan los Loaders de Webpack  
      {  
  
      }  
    ]  
  },  
  plugins: [  
    // Aquí se cargan los Plugins de WebPack  
  ]  
}
```

Estructura de base del
archivo de configuración
/webpack.config.js
loaders + plugin

```
plugins: [  
  new ExtractTextPlugin({  
    filename: 'style.css',  
    allChunks: false  
  })  
]  
};
```

Plugins de Webpack

- Los **Plugins** son como objetos (con una propiedad de aplicación) que se pueden instanciar. Le permite conectarse a todo el ciclo de vida de eventos de webpack.
- **80% de webpack** está hecho por su propio sistema de plugins .
- **Los plugins agregan funcionalidad adicional** a la compilación (optimized bundles).
- Los loaders solo se aplican según un tipo de archivo en específico, pero su funcionalidad se puede expandir a través de plugins.

En la práctica: Soportando HTML en Webpack...





```
npm i html-webpack-plugin html-loader --save-dev
```

1. Instalar dependencias

Por medio de npm, instalaremos las dependencias del **loader y plugin** para manejo de archivos HTML en webpack:

- **html-loader**
- **html-webpack-plugin**



webpack.config.js

```
module.exports = {  
  module:{  
    rules:[ //Aquí se cargan los Loaders de Webpack  
      {  
  
      }  
    ]  
  },  
  plugins:[  
    // Aquí se cargan los Plugins de WebPack  
  ]  
}
```

2. Crear el archivo **webpack.config.js**

En la raíz del proyecto (/) creamos el archivo **webpack.config.js** y preparamos el archivo de configuración colocando la estructura básica del archivo.

3. Añadir el loader: html-loader en webpack.config.js

```
module.exports = {
  module: {
    rules: [ //Aquí se cargan los Loaders de Webpack
      {
        //test: Significa que tengo que buscar
        //use: de lo que encuentre, que Loader voy a aplicar
        test: /\.html$/, //REGEX → Busca todos los archivos que terminan en .html
        use: [
          {
            loader: "html-loader", //Traduce HTML para que webpack lo entienda
            options: {minimize:true} //Minifica los archivos HTML encontrados
          }
        ]
      }
    ]
  },
  plugins: [
    // Aquí se cargan los Plugins de Webpack
  ]
}
```



webpack.config.js



webpack.config.js

Al inicio del archivo colocar:

const HtmlWebpackPlugin = require('html-webpack-plugin');

y en el apartado de plugins del archivo, colocar:

```
plugins:[
  // Aquí se cargan los Plugins de Webpack
  new HtmlWebpackPlugin({
    template:"./src/index.html", //Que
    archivo HTML va a ser el base de mi proyecto en la
    carpeta src
    filename:"./index.html" // Que único
    archivo de HTML se va a generar en la carpeta dist
    //El archivo de conf. de webpack
    simula que se trabaja desde la carpeta dist, por
    lo que no se necesita especificar.
  })
]
```

[documentación HtmlWebpackPlugin](#)

4. Cargando el plugin HtmlWebpackPlugin

A diferencia de los **loaders**, los **plugins** se mandan a llamar con un **require**.

Y para usarlos, se requiere hacer una instancia por medio de la palabra reservada **new**.

html-webpack-plugin sirve para generar un archivo **HTML 5** que incluye todos los webpack bundles en el **body** a través de la etiqueta **<script>**

5. Hacer un nuevo build del proyecto

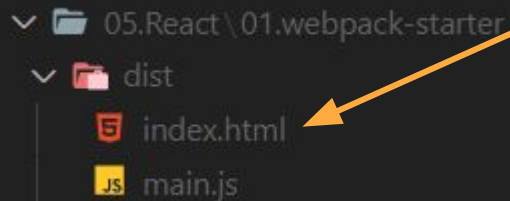
Notaremos que en esta ocasión ya menciona al archivo `index.html`.

Así mismo, debería crear en la carpeta `dist` el archivo `index.html` ya procesado por webpack.

```
//Corremos nuestro script build
//Que a su vez ejecuta webpack
npm run build
```

> webpack

```
asset main.js 609 bytes [compared for emit] [minimized]
asset ./index.html 200 bytes [compared for emit]
runtime modules 670 bytes 3 modules
cacheable modules 233 bytes
  ./src/index.js 74 bytes [built] [code generated]
  ./src/greeting.js 159 bytes [built] [code generated]
```

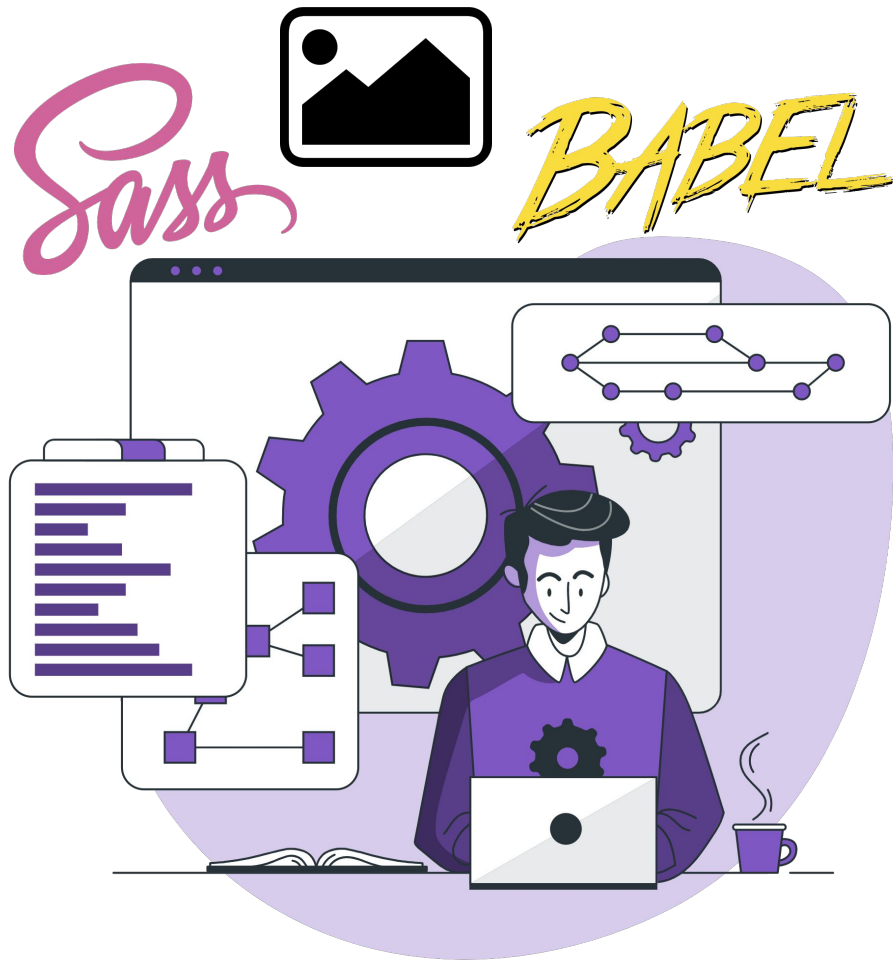


```
05.React\01.webpack-starter
├── dist
│   ├── index.html
│   └── main.js
```

Más funcionalidades con Webpack

dev.*f*
desarrollamos(personas);

dev



¿Qué más puedo necesitar?

Hasta este punto hemos visto:

1. Como instalar Webpack.
2. Como correr Webpack.
3. Como se instalan y configuran loaders y plugins.

Por lo que a partir de este momento se resumirá la forma de añadir más funcionalidades que podemos necesitar.



Soporte de CSS + SASS con Webpack

dev.f
desarrollamos(personas);



```
npm i node-sass style-loader css-loader sass-loader  
mini-css-extract-plugin --save-dev
```

```
npm i node-sass style-loader  
css-loader sass-loader  
mini-css-extract-plugin  
--save-dev
```

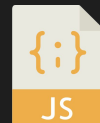
1. Instalar dependencias

Por medio de npm, instalaremos las dependencias del **loader y plugin** para manejo de archivos **CSS** y **SASS** en webpack:

- **node-sass**
- **style-loader**
- **css-loader**
- **sass-loader**
- **mini-css-extract-plugin**

2. Añadir los loaders de CSS en webpack.config.js

```
module.exports = {  
  module: {  
    rules: [ //Aquí se cargan los Loaders de WebPack  
      {  
        test: /\.scss$/,  
        use: [ //El orden de los Loader SI importa  
          "style-loader", // Procesa estilos en línea  
          "css-loader", // Procesa estilos en archivos CSS  
          "sass-loader" // Procesa estilos en archivos SCSS (SASS)  
        ]  
      },  
    ],  
  },  
}
```



webpack.config.js

3. Añadir los plugins de CSS en webpack.config.js

```
const MiniCssExtractPlugin = require('mini-css-extract-plugin');

module.exports = {
  module: {
    rules: [ //Aquí se cargan los Loaders de WebPack
      { }
    ]
  },
  plugins: [
    new MiniCssExtractPlugin({
      filename: "[name].css", //Webpack se encargará de generar un nombre del archivo de CSS por nosotros.
      chunkFilename: "[id].css" //Separa el CSS en pedacitos para que dependiendo de la vista solo cargue el CSS
      necesario. Solo se ejecuta en el caso si hay mucho CSS o es muy grande.
    }),
  ]
}
```



webpack.config.js

BABEL

Soporte de Babel (ES5+) con Webpack

dev.f
desarrollamos(personas);

dev



```
npm i @babel/core babel-loader @babel/preset-env --save-dev
```

```
npm i @babel/core babel-loader  
@babel/preset-env --save-dev
```

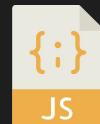
1. Instalar dependencias

Babel nos ayuda a traspilar código y convertir todo el JS moderno en JS vanilla, para que pueda ejecutarse nuestra aplicación en cualquier navegador.

- **babel-core:** Es lo mínimo necesario que necesita babel para funcionar.
- **babel-loader:** Es para que babel se pueda comunicar con webpack.
- **babel/preset-env:** Es para que babel pueda entender Javascript.

2. Añadir los loaders de Babel en webpack.config.js

```
module.exports = {  
  module: {  
    rules: [ //Aquí se cargan los Loaders de WebPack  
      {  
        test: /\.js$/, //Va a buscar todos los archivos JS en mi proyecto  
        exclude: /node_modules/, //Omite la carpeta node_modules  
        use: {  
          loader: "babel-loader" //carga babel  
        }  
      },  
    ],  
  },  
},
```



webpack.config.js



Soporte de Archivos de Imagen

dev.*f*
desarrollamos(personas);

dev



```
npm i file-loader --save-dev
```

```
npm i file-loader --save-dev
```

1. Instalar dependencias

También podemos hacer que babel nos ayude a gestionar y optimizar las imágenes que encuentre.

2. Añadir los loaders de Imagen en webpack.config.js

```
module.exports = {  
  module: {  
    rules: [ //Aquí se cargan los Loaders de WebPack  
      {  
        test: /\..(png|jpg|svg|gif|jpeg)$/,  
        use: [  
          "file-loader" //Se puede colocar de manera implícita sin usar la  
palabra loader:  
        ]  
      }  
    ],  
  },  
}
```



webpack.config.js

Soporte Live Server

dev.*f*
desarrollamos(personas);

dev



```
npm i webpack-dev-server --save-dev
```

```
npm i webpack-dev-server  
--save-dev
```

1. Instalar dependencias

webpack-dev-server nos permite ejecutar nuestro código desde un servidor con función de recargado automático (hot reloading).

Muy útil para cuando nos encontramos en entorno de desarrollo y queramos probar nuestro código.

2. Añadir script start:dev en package.json



package.json

```
package.json X
05.React > 01.webpack-starter > package.json > ...
1  {
2    "name": "01.webpack-starter",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    > Debug
7    "scripts": {
8      "test": "echo \"Error: no test specified\" && exit 1",
9      "build": "webpack",
10     "start:dev": "webpack-dev-server"
11   },
12   "keywords": [],
13   "author": "",
14   "license": "ISC",
15   "devDependencies": {
16     "@babel/core": "^7.15.5",
17     "@babel/preset-env": "^7.15.6",
18     "babel-loader": "^8.2.2",
19     "html-loader": "^2.1.2",
20     "html-webpack-plugin": "^5.3.2",
21     "webpack": "^5.53.0",
22     "webpack-cli": "^4.8.0"
23   }
24 }
```

En Resumen...

dev.*f*
desarrollamos(personas);

dev

Resumen de Webpack Básico

I. Inicializar Proyecto NPM

1. `mkdir proyecto`
2. `cd proyecto`
3. `npm init -y`

II. Instalar dependencias NPM

1. `npm i webpack webpack-cli --save-dev`
2. `npm i webpack-dev-server --save-dev`
3. `npm i html-webpack-plugin html-loader --save-dev`
4. `npm i @babel/core babel-loader @babel/preset-env --save-dev`
5. `npm i file-loader --save-dev`
6. `npm i node-sass style-loader css-loader sass-loader mini-css-extract-plugin --save-dev`

III. Configurar `webpack.config.js`

Archivo copiable/descargable desde:

<https://gist.github.com/4c72aebea32aab94161d283796a54794#file-webpack-config-js>

IV. Crear los scripts en `package.json`

```
"scripts": {  
  "build": "webpack --mode=production",  
  "start:dev": "webpack-dev-server"  
},
```