WebPack

Introducción

Vamos a aprender a configurar algunos script con NPM y WebPack. Para ello necesitamos tener previamente instalado Node.js. Será un ejemplo muy sencillo en el que crearemos una página web, además de un par de archivos css y js, que es la estructura básica que un sitio web puede tener, diferenciando así los modos de desarrollo y de producción.

NPM

Una vez que hayamos configurado nuestro servidor para alojar nuestro proyecto, ya podemos comenzar con el asistente que nos ayudará a crear el archivo `package.json`, que nos servirá para manejar las **dependencias** y los **scripts** que vamos a ejecutar en nuestro proyecto. Navegamos hasta el directorio en el que vamos a trabajar y ejecutamos el siguiente comando:

```
$ npm init
```

Inmediatamente se nos presentará un asistente, que nos ayudará a crear el archivo `package.json`. Una vez terminado este paso, podemos abrir el archivo `package.json` con nuestro editor de código y eliminar el campo `main` y el script `test`, pues no inciden en nada en el funcionamiento del mismo.

package.json

```
{
   "name": "Proyecto Webpack",
   "version": "1.0.0",
   "description": "Diseño de Interfaces Web",
   "main": "index.js",
   "scripts": {
     "test": "echo \"Error: no test specified\" && exit 1"
   },
   "author": "Alumn@",
   "license": ""
 }
```

Dependencias

Ya estamos listos para instalar las **dependencias** que necesitamos para nuestro proyecto. En la siguiente lista podemos detallar que plugins se va a emplear.

- bootstrap
- @popperjs/core
- autoprefixer
- clean-webpack-plugin
- node-sass
- nodemon
- postcss-cli
- webpack
- webpack-cli
- webpack-merge

Entonces procedemos a instalar las dependencias que necesitamos:

```
$ npm install bootstrap @popperjs/core autoprefixer clean-webpack-plugin node-sass nodemon
postcss-cli webpack webpack-cli webpack-merge --save-dev
```

El parámetro `--save-dev` garantiza que se utiliza el paquete para fines de desarrollo y se actualice su package.json. Las versiones instaladas pueden variar según la actualización del repositorio.

package.json

```
{
   "name": "Proyecto Webpack",
   "version": "1.0.0",
   "description": "Diseño de Interfaces Web",
   "scripts": {},
   "author": "Alumn@",
   "license": "",
   "devDependencies": {
     "@popperjs/core": "^2.9.2",
     "autoprefixer": "^10.2.5",
     "bootstrap": "^5.0.1",
     "clean-webpack-plugin": "^4.0.0-alpha.0",
     "node-sass": "^6.0.0",
     "nodemon": "^2.0.7",
     "postcss-cli": "^8.3.1",
     "webpack": "^5.37.1",
     "webpack-cli": "^4.7.0",
     "webpack-merge": "^5.7.3"
   }
 }
```

Directorios

Antes de pasar a crear los scripts que vamos a utilizar, debemos tener la estructura de directorios y archivos que usaremos, pues este paso es determinante para la configuración de los mismos. Tendremos organizado un directorio `assets` y ahí se almacenarán los `scss` y `js`, así como los resultantes compilados, quedando de la siguiente manera:

```
├── assets
│   └── src
│       ├── js
│       └── scss
├── package.json
└── package-lock.json
```

En el directorio `assets/src/scss/` se almacenarán los archivos `scss` y en `assets/src/js/` los archivos `js`.

Estilos

Crea un archivo `styles.scss` en `assets/src/scss/`, desde el que importaremos los archivos precompilados scss de Bootstrap.

styles.scss

```
@import "../../../node_modules/bootstrap/scss/bootstrap";
```

Javascripts

Ahora se cargará la librería de plugins de Bootstrap. Para ello crearemos un archivo llamado `scripts.js` dentro de `assets/src/js/`. Como en el ejemplo anterior, vamos a usar la opción convencional para cargar solo lo que realmente se va a emplear.

scripts.js

```
import "../../../node_modules/bootstrap/dist/js/bootstrap.bundle.min.js";
```

Estructura del proyecto

Ya tenemos nuestro proyecto listo para comenzar a trabajar con los scripts de **NPM** y **WebPack**:

```
├── assets
│   └── src
│       ├── js
│       │   └── scripts.js
│       └── scss
│           └── styles.scss
├── package.json
└── package-lock.json
```

Scripts NPM y WebPack

Ya estamos listos para comenzar a trabajar con **NPM** y **WebPack**. Para ello necesitamos de nuevo editar el archivo package.json que creamos al inicio. Donde añadiremos los scripts necesarios.

WebPack

Antes de comenzar por **WebPack**, que se encargará de compilar nuestros archivos `js`, en este caso solo tenemos el archivo `assets/src/js/scripts.js`, pero dejaremos todo listo para compilar varios archivos a la vez. Existen diferentes modos en que puedes configurar WebPack, como generación para el **desarrollo** y otra para **producción**.

En nuestro ejemplo usaremos tres archivos, `webpack.common.js`, que contendrá la configuración general, común para ambos modos `development` y `production`. Además `webpack.dev.js` y `webpack.prod.js`, para los modos: **development** y **production** respectivamente.

Common

webpack.common.js

```
const path = require('path');
const { CleanWebpackPlugin } = require('clean-webpack-plugin');
module.exports = {
    entry: {
        scripts: './assets/src/js/scripts.js',
    },
    plugins: [
        new CleanWebpackPlugin(),
    ],
    output: {
        path: path.resolve(__dirname, 'assets/dist/js/'),
        filename: '[name].js',
    }
};
```

En este caso debemos notar, que si deseamos agregar otro archivo js, solo debemos agregarlo al campo `entry`. No será necesario añadir más entradas:

webpack.common.js

```
entry: {
     scripts: './assets/src/js/scripts.js',
     plugin1: './assets/src/js/plugin1.js',
     plugin2: './assets/src/js/plugin2.js',
     …
 },
```

El campo `output` se encargará de hacer el resto del trabajo. Nótese que la salida apunta al directorio `assets/dist/js`.

Development

webpack.dev.js

```
const { merge } = require('webpack-merge');
const common = require('./webpack.common.js');
module.exports = merge(common, {
    mode: 'development',
    watch: true,
    watchOptions: {
        ignored: /node_modules/
    }
});
```

Es necesario revisar el valor del campo `mode` que es `development` y sobre todo que hemos ignorado totalmente lo que ocurre en el directorio `node_modules`.

Production

webpack.prod.js

```
const { merge } = require('webpack-merge');
const common = require('./webpack.common.js');
module.exports = merge(common, {
    mode: 'production',
});
```

La principal diferencia con el archivo webpack.dev.js es el valor del campo `mode` que está definido como `production`.

NPM

Ahora solo nos queda el último paso, crear los `scripts` que ejecutaremos, tanto en modo de **producción** como de **desarrollo**, dependiendo del entorno en que se encuentre nuestro proyecto. Es necesario editar en el archivo `package.json`, el campo `scripts`:

package.json

```
"scripts": {
     "build": "npm run css && webpack --config webpack.prod.js",
     "css-compile": "node-sass --include-path node_modules --output-style compressed --source-map true --source-map-contents true --precision 6 assets/src/scss -o assets/dist/css/",
     "css-prefix": "postcss --replace assets/dist/css/styles.css --use autoprefixer --map",
     "css": "npm run css-compile && npm run css-prefix",
     "watch": "nodemon -e scss -x \"npm run css && webpack --config webpack.dev.js\""
   },
```

Recuerda que los comandos scripts `css-compile` y `css-prefix`, el output apunta al directorio `assets/dist/css/`. Realmente en la ejecución podemos ejecutar cada uno de los scripts pero los fundamentales serán los scripts `build` y `watch`, que en definitiva son los que vamos a utilizar continuamente. Ellos son los que se encargarán de llamar a los demás comandos cuando estos sean necesarios.

Build

Este comando lo utilizaremos en un entorno de **producción**, es decir, cuando nuestro proyecto esté visible desde internet. Para ejecutarlo solo debemos navegar hasta la raíz de nuestro proyecto y:

```
$ npm run build
```

Si no ha ocurrido ningún error, entonces obtendremos una respuesta como la siguiente:

```
> project-npm-webpack@1.0.0 build
> npm run css && webpack --config webpack.prod.js
> project-npm-webpack@1.0.0 css
> npm run css-compile && npm run css-prefix
> project-npm-webpack@1.0.0 css-compile
> node-sass --include-path node_modules --output-style compressed --source-map true --source-map-contents true --precision 6 assets/src/scss -o assets/dist/css/
 Rendering Complete, saving .css file…
 Wrote CSS to /var/www/html/project-npm-webpack/assets/dist/css/styles.css
 Wrote Source Map to /var/www/html/project-npm-webpack/assets/dist/css/styles.css.map
 Wrote 1 CSS files to /var/www/html/project-npm-webpack/assets/dist/css/
> project-npm-webpack@1.0.0 css-prefix
> postcss --replace assets/dist/css/styles.css --use autoprefixer --map
 asset scripts.js 77.1 KiB [emitted] [minimized] (name: scripts) 1 related asset
 runtime modules 663 bytes 3 modules
 cacheable modules 77 KiB
    ./assets/src/js/scripts.js 74 bytes [built] [code generated]
    ./node_modules/bootstrap/dist/js/bootstrap.bundle.min.js 76.9 KiB [built] [code generated]
webpack 5.37.1 compiled successfully in 3244 ms
```

Si observamos ahora dentro de nuestro directorio `assets`, ahora tenemos una nueva carpeta `dist` donde se han guardado los archivos `css` y `js` compilados y comprimidos.

```
├── assets
│   ├── dist
│   │   ├── css
│   │   │   ├── styles.css
│   │   │   └── styles.css.map
│   │   └── js
│   │       ├── scripts.js
│   │       └── scripts.js.LICENSE.txt
│   └── src
│       ├── js
│       │   └── scripts.js
│       └── scss
│           └── styles.scss
```

Watch

El comando `watch` lo usaremos en un entorno de **desarrollo**, es útil para que a medida que vamos trabajando, él solo va compilando los archivos `scss` y `js`.

```
$ npm run watch
```

Entonces, mientras editamos los archivos `scss` o `js`, nos irá respondiendo de la siguiente manera, en caso de que no ocurra ningún error:

```
> project-npm-webpack@1.0.0 watch
> nodemon -e scss -x "npm run css && webpack --config webpack.dev.js"
 [nodemon] 2.0.7
 [nodemon] to restart at any time, enter rs
 [nodemon] watching path(s): .
 [nodemon] watching extensions: scss
 [nodemon] starting npm run css && webpack --config webpack.dev.js
> project-npm-webpack@1.0.0 css
> npm run css-compile && npm run css-prefix
```

```
> project-npm-webpack@1.0.0 css-compile
> node-sass --include-path node_modules --output-style compressed --source-map true --source-map-contents true --precision 6 assets/src/scss -o assets/dist/css/
 Rendering Complete, saving .css file…
 Wrote CSS to /var/www/html/project-npm-webpack/assets/dist/css/styles.css
 Wrote Source Map to /var/www/html/project-npm-webpack/assets/dist/css/styles.css.map
 Wrote 1 CSS files to /var/www/html/project-npm-webpack/assets/dist/css/
> project-npm-webpack@1.0.0 css-prefix
> postcss --replace assets/dist/css/styles.css --use autoprefixer --map
 asset scripts.js 83.7 KiB [emitted] (name: scripts)
 runtime modules 937 bytes 4 modules
 cacheable modules 77 KiB
   ./assets/src/js/scripts.js 74 bytes [built] [code generated]
   ./node_modules/bootstrap/dist/js/bootstrap.bundle.min.js 76.9 KiB [built] [code generated]
webpack 5.37.1 compiled successfully in 361 ms
```

Archivo `index.html`

Finalmente necesitamos crear un archivo `index.html` que funcionará como la página del proyecto, donde llamaremos los archivos `css` y `js` dentro del directorio `assets/dist/css/` y `assets/dist/js/` respectivamente.

index.html

```html
<!doctype html>
<html lang="en">
 <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="assets/dist/css/styles.css">
    <title>Proyecto Webpack</title>
 </head>
 <body>
    <h1>Hello World!</h1>
    <script src="assets/dist/js/scripts.js"></script>
 </body>
 </html>
```

Instalación final

Para instalarlo en tu servidor, solo debes clonar el repositorio y ejecutar los mismos scripts que hemos visto aquí, recuerda ya no es necesario subir tu carpeta node_modules dado que al ejecutar `npm install` se descargarán todos los plugins necesarios y según las versiones marcadas en el package.json:

```
$ git clone <dirección url de tu repositorio>
$ npm install
$ npm run build
```