Tema 10

Webpack y otros módulos

Webpack

Webpack es un sistema de agrupación para preparar el desarrollo de una aplicación web para producción. Cuenta con múltiples opciones de configuración.

Se puede considerar como evolución de Grunt y Gulp al permitir automatizar los procesos principales como preprocesar código.

Para poder utilizar Webpack, se debe tener instalado **node.js** en el equipo.

Webpack

Webpack se encarga de transformar y agrupar el código para devolver una nueva versión de este mismo. Se le puede decir cada transformación que tu código necesita realizar de manera que devolverá un paquete de archivos con todos esos cambios.

Hay tres cosas que Webpack necesita saber:

- El punto inicial de la aplicación o el archivo raíz Javascript.
- Las transformaciones a realizar en el código.
- La ruta donde debería guardar el nuevo paquete de código.

Webpack

Para empezar con Webpack se crea un nuevo archivo que contendrá sus configuraciones. Convenientemente, se llamara webpack.config.js y se ubicará en el directorio raíz del proyecto.

Una vez creado el archivo, definiremos el objeto que exportará, el cual representará nuestras configuraciones para Webpack.

El siguiente paso será aplicar las tres condiciones que debe saber Webpack al fichero ya creado.

Webpack - Configuración

Para empezar a utilizar Webpack, vamos a crear un directorio nuevo que será nuestro proyecto. Ejecutaremos desde la consola:

```
mkdir webpack-demo
cd webpack-demo
npm init -y # -y genera el fichero package.json
```

Una vez creado el directorio y el fichero *package.json*, procederemos a instalar Webpack. Ejecutaremos el siguiente comando:

```
npm install webpack --save-dev
```

Webpack - Configuración

Para poder ejecutar Webpack, simplemente deberemos escribir el comando *webpack* en la consola. Si tratamos de ejecutarlo, veremos que nos saldrá lo siguiente:

```
webpack-demo $ node_modules/.bin/webpack
No configuration file found and no output filename configured via CLI option.
A configuration file could be named 'webpack.config.js' in the current
directory.
Use --help to display the CLI options.
```

Esto se debe a que todavía no hemos terminado de crear los archivos necesarios para el proyecto.

Webpack - Configuración

Para que Webpack pueda empezar a trabajar, nuestro proyecto deberá tener la siguiente estructura:

```
- app/
  - index.js
  - component.js
- build/
- package.json
- webpack.config.js
```

Vamos a proceder a crear las carpetas *app* y *build*, así como los ficheros *index.js* y *component.js*.

Webpack - Configuración

Nuestra aplicación consistirá en el básico *Hello World*. Implementaremos los siguientes archivos dentro de app.

app/component.js

```
export default function (text = 'Hello world') {
  const element = document.createElement('div');

  element.innerHTML = text;

  return element;
}
```

app/index.js

```
import component from './component';

document.body.appendChild(component());
```

Webpack - Configuración

Para mantener las cosas simples de mantener, usaremos *html-webpack-plugin* para generar un *index.html* para nuestra aplicación. Se puede instalar de la siguiente manera:

```
npm install html-webpack-plugin --save-dev
```

Es bueno tener en nuestro proyecto campos de entrada y salida en nuestra configuración. Para ilustrar como conectarlos con el plugin anteriormente dicho, modificaremos nuestro archivo webpack.config.js.

Webpack - Configuración

De tal manera que nos quedará:

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

const PATHS = {
  app: path.join(__dirname, 'app'),
  build: path.join(__dirname, 'build'),
};

module.exports = {
   entry: {
    app: PATHS.app,
  },
  output: {
    path: PATHS.build,
    filename: '[name].js',
  },
  plugins: [
    new HtmlWebpackPlugin({
      title: 'Webpack demo',
    }),
  ],
};
```

Webpack - Configuración

Si volvemos a ejecutar el comando webpack, veremos que ahora no dará ningún problema. En la salida de la consola nos mostrará información variada, como por ejemplo la versión actual de Webpack, el tiempo que ha tardado en ejecutarse, etc.

Lo que más nos interesa son los ficheros generados en la carpeta build: app.js y index.html. Si ejecutamos este último con cualquier navegador, tendremos nuestra aplicación funcionando.

Babel

Babel es una herramienta impulsada por la comunidad que ayuda al usuario con la última versión de JavaScript actuando como compilador.

Derivando de Babel, tenemos *babel-loader*, que nos permite compilar archivos JavaScript utilizando Babel y Webpack de forma conjunta.

Una manera de integrar Babel sería asignarlo como *loader* dentro de la propiedad *module*, en el fichero de configuración.

```
module: {
  loaders: [
    {
      test: /\.js$/,
      exclude: /(node_modules|bower_components)/,
      loader: 'babel-loader',
      query: {
        presets: ['es2015']
      }
    } ] }
```

Dropzone

El componente **Dropzone** permite a los usuarios arrastrar y soltar archivos en un área de subida. Se instala con **$ npm install react-dropzone-component**.

```
import React from 'react';
import ReactDOM from 'react-dom';
import DropzoneComponent from 'react-dropzone-component';

var componentConfig = {
    iconFiletypes: ['.jpg', '.png', '.gif'],
    showFiletypeIcon: true,
    postUrl: '/uploadHandler'
};

ReactDOM.render(
    <DropzoneComponent config={componentConfig}
                       eventHandlers={eventHandlers}
                       djsConfig={djsConfig} />,
    document.getElementById('content')
);
```

Dropzone

Hay muchas operaciones que requerirán acceder al objeto de la dropzone. Para coger dicho objeto, se usa el evento **init**, cuyo callback recibirá una referencia al objeto de la dropzone como parámetro.

```
var myDropzone;

function initCallback (dropzone) {
    myDropzone = dropzone;
}

function removeFile () {
    if (myDropzone) {
        myDropzone.removeFile();
    }
}
```