{desafío}
latam_

Introducción a ES6+ _

Parte I

Conceptos Claves

- Librería
- Framework
- Servidor
- HTML
- CSS
- JavaScript
- Clase
- Objeto
- Callback
- Layout

¿Qué aprenderemos?

webpack

ES6

Motivación

Utilidad y Versatilidad

ES6 es la versión más utilizada actualmente por los desarrolladores. Su capacidad de adaptarse a trabajar con diversos frameworks de gran utilidad.

La nueva versión es útil tanto en frontend como en backend. La cantidad de nuevas posibilidades de trabajo abre un mundo para trabajar de mejor forma con paradigmas funcionales y orientación a eventos.

ES6

Estructuración de un Proyecto ES6+ con Babel/Webpack

Objetivos

El objetivo de esta sección es enseñarles algunos elementos básicos para configurar un entorno de trabajo de modo que permita transpilar código ES6+ a alguna versión más antigua y de esa forma, volverlo compatible para un mayor espectro de navegadores.

Node Package Manager (NPM)

Node Package Manager o simplemente npm es un gestor de paquetes, el cual hará más fáciles nuestras vidas al momento de trabajar con Node, ya que gracias a él podremos tener cualquier librería disponible con solo una línea de código, npm nos ayudará a administrar nuestros módulos, distribuir paquetes y agregar dependencias de una manera sencilla.

Dependencia

Particularmente, hablar de dependencia es similar a un package en Java o un módulo en python, una dependencia como tal es una librería que se puede componer de otras librerías, tanto internas como utilizar recursos externos a ésta.

Las librerías están hechas para facilitarnos la vida como desarrolladores.

NPM

Cuando instalamos nuevos paquetes lo que hace npm es instalarlo de manera local en nuestro proyecto dentro de la carpeta node_modules. Esta carpeta contendrá todas las librerías y dependencias necesarias para iniciar el proyecto, según lo que se defina en el archivo package.json.

```
npm install **nombre_dependencia** --save
```

Para instalar npm en nuestro ordenador es necesario, primero, tener Node.js. Así que verificaremos que se encuentre instalado:

```
nodejs -v
```

NPM

Es importante la instalación de node puesto que para usar ES6+ es necesario correr ciertos comandos que proveen librerías de node para poder convertir el código en una forma legible para los navegadores.

Babel y Webpack

En esta sección abordaremos las principales características de Babel y webpack, que van de la mano puesto que en su conjunto permiten configurar un entorno de trabajo.

Transpilar código ES6+ para volverlo interpretable y generar una configuración para ejecutar los comandos necesarios para que la aplicación o el servicio como tal funcione correctamente.

Babel

Babel es un compilador que convierte un estándar nuevo en una versión totalmente compatible de JavaScript. Así, tenemos la ventaja de poder programar en un estándar nuevo sin renunciar a la compatibilidad entre navegadores.

Webpack

Webpack se define como un empaquetador de módulos (un bundler en la jerga habitual), pero que hace muchísimas cosas más:

- Gestión de dependencias
- Ejecución de tareas
- Conversión de formatos
- Servidor de desarrollo
- Carga y uso de módulos de todo tipo (AMD, CommonJS o ES6)

Webpack

- Es una herramienta extremadamente útil cuando desarrollamos aplicaciones web diseñadas con filosofía modular, es decir, separando el código en módulos que luego se utilizan como dependencias en otros módulos.
- Una de las cosas que hace realmente bien Webpack es la gestión de esos módulos y de sus dependencias,
- También puede usarse para cuestiones como concatenación de código, minimización y ofuscación, verificación de buenas prácticas (linting), carga bajo demanda de módulos, etc.
- No solo el código JavaScript se considera un módulo. Las hojas de estilo, las páginas HTML e incluso las imágenes se pueden utilizar también como tales, lo cual da un extra de potencia muy interesante.

Estructura de un Proyecto Node JS Básico

Los ejemplos presentados a continuación serán básicos orientados a proyectos personales.

```
proyecto
|   README.md
|   index.html
|
└──css
|   |   style.css
|   |   otherStyle.css
|
|
|
└──js
|   |   main.js
|
|
|
|
|___img
|   |   img1.png

...
```

{desafío} latam_

Ejemplo Proyecto Básico

Como se observa en el ejemplo, tenemos 3 carpetas principales:

Css: Carpeta en donde se guardan todos los archivos de estilos del proyecto.

Js: Carpeta en donde se guardan todos los archivos de tipo del proyecto. En este caso escribiremos en estos archivos con el formato de ES6. Explicaremos el lenguaje más adelante, por ahora nos enfocaremos en entender que dichos archivos estarán escritos en ES6.

Img: Carpeta en donde se guardan las imágenes del proyecto.

Ejemplo Básico de Proyecto

- Correr el comando: npm init en la carpeta raiz de tu proyecto.
  - De esta forma se iniciarán los archivos necesarios para transformar tu proyecto en uno que pueda ser compilado por babel.
  - Al correr el comando el sistema te hará unas preguntas, presiona enter para crear la configuración con los valores predeterminados.

- Luego, debes correr la siguiente secuencia de comandos para instalar las dependencias necesarias:

```
npm i webpack webpack-cli @babel/core @babel/plugin-proposal-object-rest-spread @babel/preset-env babel-loader -D
```

Ejemplo Básico de Proyecto

Básicamente lo que hace este comando es instalar las dependencias de webpack y babel para configurar tu proyecto como uno compilable.

Vas a observar que en tu carpeta raíz se crearon diversos archivos nuevos.

- Uno de ellos se llama package.json para proyectos Node.
- Por defecto, siempre estará configurado el script: npm start.

Ejemplo Básico de Proyecto

Abriremos el archivo package.json y reemplazamos su contenido por lo siguiente:

```
{
    "name": "proyecto",
    "version": "1.0.0",
    "description": "",
    "main": "webpack.config.js",
    "scripts": {
        "build": "webpack --mode production"
    },
    "repository": {
        "type": "git", // opcional
        "url": ""
    },
    "keywords": [],
    "author": "",
    "license": "ISC",
    "bugs": {
        "url": "" //opcional
    },
    "homepage": "",
    "devDependencies": {
        "@babel/core": "^7.0.0-beta.42",
        "@babel/plugin-proposal-object-rest-spread": "^7.0.0-beta.42",
        "@babel/preset-env": "^7.0.0-beta.42",
        "babel-loader": "^8.0.0-beta.2",
        "webpack": "^4.2.0",
        "webpack-cli": "^2.0.12"
    }
}
```

Webpack - Configuración

Etiquetas a considerar:

- Esta etiqueta provee comandos para ejecutarse y generar ciertas acciones:

```
"Scripts": {...}
```

- La siguiente etiqueta son las dependencias que estarán instaladas para el entorno de desarrollo de un producto:

```
"devDependencies": {...}
```

Webpack - Configuración

Ahora, en nuestro archivo ubicado en /js/main.js colocaremos el siguiente bloque de código escrito en ES6+:

```
let { x, y, ...z } = { x: 1, y: 2, a: 3, b: 4 };
console.log(x); // 1
console.log(y); // 2
console.log(z); // { a: 3, b: 4 }
[5, 6].map(n => console.log(n));
```

Webpack - Configuración

Ahora, iremos al archivo de configuración de babel y colocaremos el siguiente bloque de código:

```
{
 test: /\.js$/, //Regular expression
 exclude: /(node_modules|bower_components)/,//excluded node_modules
 use: {
 loader: "babel-loader",
 options: {
   presets: ["@babel/preset-env"]  //Preset used for env setup
  }
 }
```

Este es un archivo en formato JSON que toma los archivos de tipo .js , con ciertas excepciones y los transforma en archivos legibles por un navegador.

Webpack - Configuración

Por otra parte, es necesario hacer una configuración de webpack cambiando y reemplazando lo escrito en el archivo:

- **webpack.config.js**

Por lo siguiente:

```
const path = require("path");
const webpack = require("webpack");
const webpack_rules = [];
const webpackOption = {
    entry: "./app.js",
    output: {
        path: path.resolve(__dirname, "dist"),
        filename: "bundle.js",
    },
    module: {
        rules: webpack_rules
    }
};
let babelLoader = {
    test: /\.js$/,
    exclude: /(node_modules|bower_components)/,
    use: {
        loader: "babel-loader",
        options: {
            presets: ["@babel/preset-env"]
        }
    }
};
webpack_rules.push(babelLoader);
module.exports = webpackOption;
```

Webpack - Configuración

El archivo webpack.config.js es Javascript, define un conjunto de objetos asignados a diversas variables con el objetivo de generar una configuración de webpack ordenada y legible.

Webpack - Babel Configuración

Ahora, configuraremos el archivo final de babelrc :

```
{
    "presets": [
        [
            "@babel/preset-env",
            {
                "targets": {
                    "node": "8.9.9", // debes colocar acá la versión de node
que tengas
                    "esmodules": true
                }
            }
        ]
    ],
    "plugins": ["@babel/plugin-proposal-object-rest-spread"]
}
```

Webpack - Babel Configuración

Para validar que todo funciona correctamente, es necesario correr el comando:

```
npm run build
```

Ahora nos vamos al index.html, en este archivo ya no haremos referencia al archivo real que trabajamos en ES6, sino que a su compilado ubicado en dist, de esta forma:

```
<script type="text/javascript" src="./dist/bundle.js"></script>
```

Webpack Configuración

Y listo, ahora cada vez que hagamos cambios, podemos hacer el build para reflejar dichos cambios en la compilación y automáticamente index.html apuntará al archivo compilado.

Proyecto final

Por último, te presento como se debe ver tu carpeta final de proyecto en base al ejemplo que realizamos:

{desafío} latam_

Academia de talentos digitales

www.desafiolatam.com