“DE JAVASCRIPT CLÁSICO A JAVASCRIPT MODERNO”

Talleres de informática
https://t.me/TalleresInformatica

Sergio García

4 de mayo de 2020

Versión:200505.1149

Licencia

ÍNDICE DE CONTENIDO

TALLER "DE JAVASCRIPT CLÁSICO A JAVASCRIPT MODERNO"

1. INTRODUCCIÓN

Taller en el que de la mano de un sencillo Chat aprovechando Firebase y Firestore, aprenderemos como pasar del Javascript clásico al Javascript Moderno.

Todo el material del taller está disponible en https://github.com/sergarb1/TallerJavascriptModerno

Para realizar el taller necesitamos NodeJS https://nodejs.org/es/.

Asimismo, para algunas funciones utilizaremos Firefox Developer Edition. Lo podéis descargar en https://www.mozilla.org/es-ES/firefox/developer/

También puede usarse Google Chrome y DevTools.

Como editor recomendamos Visual Studio Code https://code.visualstudio.com/

2. PARTE 1: APLICACIÓN SENCILLA USANDO FIREBASE Y FIRESTORE

2.1 ¿Que son Firebase y Firestore?

Firebase https://firebase.google.com/ es un servicio que nos permite alojar y desplegar aplicaciones y todo lo necesario para que se ejecuten en la nube. Entre otras opciones, nos permite una base de datos textual en tiempo real y sin necesidad de interacción en servidor usando **Firestore**. Esta base de datos tiene como particularidad que no solo nos permitirá almacenar información sino que nos avisará cuando se modifique para poder actualizar los datos en tiempo real sin tener que lanzar consultas explicitas a un servidor.

2.2 Como empezar

Para empezar, debemos crear un proyecto en **Firebase** y guardar sus credenciales. Con ese proyecto creado, podemos acceder a su base de datos **Firestore**.

Para hacer pruebas como en este pequeño ejemplo debemos modificar las reglas de autorización de **Firestore**, para que cualquiera pueda escribir. Esto es así porque es un ejemplo y estamos entrenando, en producción deberíamos limitar el acceso ya que sino cualquiera podría acceder y modificar nuestra base de datos.

Las reglas que permite acceso completo a todo el mundo son:

```
service cloud.firestore {
      match /databases/{database}/documents {
            match /{document=**} {
                  allow read, write: if true;
            }
      }
}
```

2.3 Enlaces a visitar para saber más de Firebase y Firestore

Aquí algunos enlaces interesantes para quienes desconozcáis Firebase y Firestore.

- Primeros pasos con Firebase: https://firebase.google.com/docs/functions/get-started?hl=es
- Primeros pasos con Firestore: https://firebase.google.com/docs/firestore/quickstart?hl=es-419
- Modelo de datos con Firestore https://firebase.google.com/docs/firestore/data-model?hl=es-419#documents
- Obtener datos con Firestore https://firebase.google.com/docs/firestore/query-data/get-data?hl=es
- Obtener actualizaciones en tiempo real con Firestore https://firebase.google.com/docs/firestore/query-data/listen?hl=es-419

2.4 Ejemplo utilizado en el taller

En la parte 2 de este taller trabajaremos con un ejemplo de Chat en tiempo real hecho con estas tecnologías. En el repositorio del taller se incluye código comentado:

https://github.com/sergarb1/TallerJavascriptModerno

Dicho ejemplo consiste en una aplicación de Javascript que hacer lo siguiente:

1. Incluye las bibliotecas para trabajar con Firebase y Firestore.
2. Se conecta mediante credenciales a Firebase y Firestore.
    - **IMPORTANTE**: en los ejemplos las credenciales se proporcionan vacías, deben rellenarse con lo que proceda.
3. Al cargarse el DOM, obtiene el estado del chat en Firestore y lo carga en un DIV.
4. Hay un formulario donde se puede enviar chats, que se traducen en nuevas entradas en Firestore.
5. Cada vez que se añada una entrada en Firestore al chat desde cualquier fuente, Firestore avisa a todos los clientes y estos lo añade en la parte superior del div del chat.

Por seguridad y para evitar ataques XSS, todo lo enviado y mostrado en el chat se le aplica un filtro para evitar código HTML y Javascript malicioso.

Mientras dure el taller la aplicación del chat estará disponible para ser probada en https://apuntesfpinformatica.es/talleres/chat/

3. PARTE 2 – PASANDO DE JAVASCRIPT CLÁSICO A MODERNO A TRAVÉS DE EJEMPLOS

3.1 Chat simple – Versión 0 (Todo junto!)

Esta es la versión inicial de nuestro Chat simple. Consiste en un **"index.html"** donde tenemos todo el Javascript dentro de una etiqueta <script> y como externo únicamente un fichero **"styles.css"** donde guardamos la hoja de estilo CSS.

Las bibliotecas externas se incluyen mediante <script src>

Como funciones destacadas:

- **escapeHTML**: función auxiliar que recibe un texto y lo devuelve "escapando" el HTML para evitar ataques de XSS. Es llamada tanto al enviar datos como la mostrarlos.
- **inicio**: función llamada al cargarse el body para asegurarnos el DOM este cargado. Esta se encarga de inicializar el div del chat. Asimismo, también mete un manejador, para que cuando Firestore le avise de un cambio en la base de datos, el actualice el chat.
- **enviarMensaje**: función encargada de enviar una nueva entrada a nuestra bases de datos en Firestore. Esta asociada al click en el botón "Enviar".

3.2 Chat simple – Versión 1 (Separamos un poco con <script src>)

Similar al primero, solo que separamos utilizando <script src> y dividimos la funcionalidad en dos ficheros, uno donde están tanto "inicio" como "escapeHTML· y otra donde esta "enviarMensaje". Asimismo del código HTML se han eliminado los manejadores de eventos y se han introducido dentro del código Javascript.

3.3 Chat simple – Versión 2 (Comienza la revolución: empezamos con Webpack)

Comenzamos a usar NodeJS, su gestor de paquetes NPM y según vuestra configuración o sistema operativo NPX para la ejecución de paquetes.

Pasos para empezar:

1. Creamos un directorio para nuestro proyecto
2. Usando en el directorio del proyecto ***"npm init"*** (nos pide información) o ***"npm init -y"*** (usa información por defecto) inicializa el directorio como proyecto Node con el fichero de configuración **"package.json"**
    - Para saber más de **"package.json"** puedes visitar https://medium.com/noders/t%C3%BA-yo-y-package-json-9553929fb2e3
    - Si tenemos un directorio con **"package.json"** y ejecutamos ***"npm install"***, NPM descargará automáticamente todas las dependencias de nuestro proyecto.

3. Con ***"npm install webpack webpack-cli –save-dev"*** instalamos los paquetes webpack y webpack-cli como paquetes utilizados por nuestro proyecto cuando estemos en desarrollo y no en producción (por eso **–save-dev** y no **–save** a secas).
    - Recordamos que las siglas CLI suelen indicar que es "Comand Line Interface". En este caso webpack-cli nos ayuda a realizar ciertas operaciones con Webpack.

Webpack es una herramienta que nos permite muchas opciones. En esta parte del taller, simplemente la utilizaremos para que automáticamente analice nuestros ficheros ".js" y genere un único fichero ".js" para su distribución.

Si queréis saber más sobre Webpack https://webpack.js.org/guides/getting-started/

Aunque Webpack puede usarse vía consola, es más útil disponer de un fichero de configuración, este suele llamarse por defecto **"webpack.config.json"** aunque puede tener otros nombres sobretodo para tener distintas opciones de configuración.

En el ejemplo nuestro fichero contiene:

```
const path = require('path');
module.exports = {
  entry: './src/main.js',
  output: {
    filename: 'main.js',
    path: path.resolve(__dirname, 'dist'),
  },
};
```

Con esta configuración, Webpack procesa el fichero **"/src/main.js"** de nuestro proyecto y guarda el nuevo fichero generado con todas las inclusiones como **"/dist/main.js"**.

En el directorio **"/dist"** hemos dejado a mano nuestro HTML y nuestra hoja de estilo CSS.

Para conseguir el empaquetado, procedemos a ejecutar Webpack usando ***"npx webpack"*** o ***"npx webpack –config webpack.config.json"***. En ambos casos tomara como fichero de configuración "webpack.config.json."

En el código Javascript, usamos **"export"** para indicar que una función es exportable

```
 export function escapeHtml(text) {
   return text.replace(/&/g, "&amp;")
     .replace(/</g, "&lt;")
     .replace(/>/g, "&gt;")
     .replace(/"/g, "&quot;")
     .replace(/'/g, "&#039;");
 }
```

Asimismo usamos import con {} para importar funciones:

```
import  { enviarMensaje } from './eventos.js';
import  { escapeHtml } from './eventos.js';
```

Para saber mas sobre export y export default

https://stackoverflow.com/questions/33611812/export-const-vs-export-default-in-es6

3.4 Chat simple – Versión 3 (Webpack se alía con Webpack-dev-server)

3.4.1 Webpack en modo "watch"

En primer lugar, modificando "scripts" dentro del fichero "package.json" podemos poner nombres a scripts y que se ejecuten con ***"npm run nombrescript"***.

Para este ejemplo creamos dos scripts:

```
"build": "webpack –progress -p",
"watch":"webpack –progress --watch"
```

El primero básicamente ejecuta en modo producción Webpack tal como vimos en el primer ejemplo. El segundo, lanzado con ***"npm run watch"*** ejecuta Webpack en un modo muy interesante: el modo "watch". Este modo básicamente si detecta que un fichero de los que procesa Webpack recibe algún cambio, es que debe procesarlo automáticamente. Es decir, si cambiamos alguno de los ".js" de nuestro proyecto, Webpack lo detecta y genera el fichero **"main.js"**.

3.4.2 Copy-webpack-plugin

Para que el proyecto tenga una estructura más amigable y que los ficheros **"index.html"** y **"styles.css"** no tengan que estar copiados directamente a "**/dist**", nos hemos apoyado en el plugin "**copy-webpack-plugin**" hemos definido la siguiente estructura: **"src/js"**, **"src/html"** y **"src/css"**. Configuraremos el plugin para que se detecten cambios en la carpeta "**src/html**" y "**src/css"** (que no son procesadas directamente por Webpack") y sus contenidos se copien a "**/dist"**.

Si queréis saber mas de este plugin podéis visitar https://webpack.js.org/plugins/copy-webpack-plugin/

Para poner en marcha nuestra configuración daremos los siguientes pasos:

1. Instalaremos el plugin con ***"npm install copy-webpack-plugin –save-dev"***
2. Añadiremos arriba del todo del **"webpack.config.js"** la linea ***"const CopyPlugin = require('copy-webpack-plugin');"***

3. Añadiremos el siguiente código a **"webpack.config.js"** para el copiado automático de los directorios especificados:

```
plugins: [
       new CopyPlugin(
           [
               { from: 'src/html', to: './', force: true },
               { from: 'src/css', to: './', force: true },
           ]
       ),
 ]
```

3.4.3 devtool:source-maps

Cuando Webpack nos empaquete varios ficheros, nos genera una batiburrillo de código difícil de depurar. Para facilitar la depuración existe generación de "source-maps" en el código, que en modo "development" incluyen información para enlazar el código generado con el código original .

Mediante herramientas expertas, como por ejemplo el depurador de "Firefox developers" se puede depurar respecto al código original y no al código empaquetado.

Para ver las distintas opciones de configuración de "source-maps" https://webpack.js.org/configuration/devtool/

En el ejemplo, mediante "webpack.config.json" nosotros usamos la siguiente configuración:

```
devtool: 'eval-source-map',
```

3.4.4 Webpack-dev-server

La característica "watch" es muy útil, pero se hace mucho más útil cuando entra en juego el plugin **"webpack-dev-server"**. Este es un plugin configurable que crea un pequeño servidor web donde se cargará la página para facilitarnos su depuración.

En esta sección hablaremos sobre como configurarlo para que cuando se haga un cambio no solo nos genere de nuevo los fichero empaquetados, sino para que además de forma automática nos recargue el navegador y veamos necesario.

Si queréis saber más sobre este plugin https://webpack.js.org/configuration/dev-server/

Para poner en marcha esta gran utilidad realizamos los siguientes pasos:

1. Instalaremos el plugin con ***"npm install webpack-dev-server –save-dev"***
2. Añadiremos un script a **"package.json"** para llamar al servidor.
   - ***"server": "webpack-dev-server –open"*** que será llamado con **"npm run server"**.
   - También podemos llamar a mano al plugin con ***"npx webpack-dev-server –open"***

3. Añadiremos la siguiente configuración a **"webpack.config.json"**

```
devServer: {
publicPath:"/",
hot:true,
contentBase: path.resolve(__dirname, 'dist'),
watchContentBase: true, // Mira cambios en /dist
writeToDisk: true,
}
```

Esta configuración indica que el servidor recargue el navegador automáticamente (hot), que su carpeta inicio sea **"/dist"**, que vigile los cambios en ***"/dist"*** *y que al ejecutarlo los ficheros generados no estén en memoria, sino que los copie a disco en* **"/dist"** (necesario para que también actualice automáticamente cambios en **"src/html"** y **"src/css"**).

3.4.5 Incluyendo al proyecto para producción las biliotecas de Firebase

Con un fin puramente didáctico (abajo explicamos porque en este contexto no es lo más útil), podemos guardar nuestras bibliotecas para producción y que Webpack las incluya en el fichero ".js" generado. Podemos instalar Firebase con ***"npm install --save firebase".*** Un detalle importante es que al guardarse la biblioteca como parte del proyecto y no solo para modo "development", se utiliza en el guardado **--save** lugar de **–save-dev**.

Para incluir estas bibliotecas guardadas, en el código usamos

```
global.firebase = require("firebase");
require("firebase/firestore");
```

¿Porque en este contexto no es útil hacer esto? A efectos prácticos, al usar bibliotecas tipo Firebase (o otras muy comunes, como jQuery, Vue, etc.) nos puede interesar que en vez de obtenerlas de nuestro sitio como hemos hecho en este ejemplo, la obtengas de los CDN optimizados para ello como estamos haciendo en ejemplos anteriores con este código de "**index.html**".

```
<script src="https://www.gstatic.com/firebasejs/7.2.3/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/7.2.3/firebase-firestore.js"></
script>
```

3.5 Ejemplo 4 – Babel como transpilador

Babel es un transpilador (un compilador pasa de código fuente a código máquina. Un transpilador transpila de un código fuente a otro código fuente en otro lenguaje).

A efectos prácticos, nos permite escribir código Javascript en el estándar que queramos (ES6, Typescript, etc.) y traducir ese código a otras versiones del estándar de Javascript diferentes. Generalmente buscando que sean soportadas por navegadores más antiguos.

Para saber más sobre Babel https://medium.com/@renzocastro/webpack-babel-transpilando-tu-js-502244a61f5b

Para instalarlo utilizamos ***"npm install babel-core babel-preset-env babel-loader --save-dev"***

Para configurarlo y que funcione de forma automática y transparente con Webpack modificamos **"webpack.config.json"** añadiendo:

```
module:{
      rules: [
            {
                  test: /\.js$/,
                  exclude: /node_modules/,
                  use: {
                        loader: 'babel-loader'
                  }
            }
      ]
},
```

Esto indica que cada fichero ".js" que pase por "Webpack" (excluyendo los paquetes descargados en **"/node_modules"**) debe ser procesado por Babel.

Para configurar Babel, usaremos el fichero "**.babelrc"**. Aquí un ejemplo donde se pide que transpile según los navegadores que queremos soporte nuestro código. Para indicar navegadores se usa la sintaxis de https://browserl.ist

En el ejemplo esta activo Firefox 71, pero están comentado para "jugar" con ellos NodeJS y IE 6.

```
{     "presets": [
            ["@babel/preset-env", {
                  "targets": {
                        "firefox": "71" //"ie": "6"
                  }
            }]
      ]
}
```

3.6 Ejemplo 5 – ESLint como linter

ESLint es un linter. Este nos permite descubrir errores de código así como puede asesorarnos e incluso obligarnos a seguir un estilo de código.

Entornos como Visual Studio Code y su plugin para ESLint, no solo integran el linter dándote información mientras desarrollas, sino que además te permiten configurar el formateador de documentos para que automáticamente lo formatee al formato deseado por el linter.

- Mas información en https://scotch.io/tutorials/linting-and-formatting-with-eslint-in-vs-code

Para instalar ESLint en nuestro proyecto podemos usar ***"npm install eslint –save-dev".***

Para generar un fichero de configuración **".eslint.js"** podéis usar ***"npx eslint –init"*** y mediante unas preguntas os generará el fichero "**.eslint.js**" que luego podéis adaptar al gusto.

El fichero usado en el ejemplo es:

```
module.exports = {
  env: {
    browser: true,
    es6: true
  },
  extends: [
    'standard'
  ],
  globals: {
    Atomics: 'readonly',
    SharedArrayBuffer: 'readonly'
  },
  parserOptions: {
    ecmaVersion: 2018,
    sourceType: 'module'
  },
  rules: {
    "no-tabs": 0
  }
}
```

De forma resumida, indicar que en este fichero me baso en las reglas "**standard**" para estilo de código y que he añadido ***"no-tabs": 0*** para amplias las reglas y me permita el uso de tabuladores.

4. ¿DEBO REPETIR ESTA CONFIGURACIÓN PARA CADA PROYECTO?

Ya sabemos que es Webpack, Webpack-dev-server, Babel y ESLint. ¿Debo realizar estos pasos para cada proyecto que inicie? La respuesta en NO. Basta con hacerlo una vez y guardar el proyecto con la configuración (por ejemplo en un repositorio de GitHub o GitLab) y simplemente clonarlo cada vez que empecemos un proyecto.

La base con la configuración aquí descrita esta disponible en el repositorio https://github.com/sergarb1/BaseJavascriptModerno

Simplemente debéis bajar el repositorio y tras ello ejecutar ***"npm install"*** para que instale los módulos que necesita el proyecto base. (Webpack, Babel, ESLint).

5. ENTORNOS WEB PARA TRABAJAREMOS

Aparte de esta configuración para trabajar en local, hay servicios que te permiten tener un entorno online similar. Suelen ser gratuitos para proyectos públicos y de un desarrollado y de pago para proyectos privados y múltiples desarrolladores. Algunos de los más conocidos son:

- CodePen: https://codepen.io/
- StackBlitz: https://stackblitz.com/
- JS Fiddle: https://jsfiddle.net/

6. TECNOLOGÍAS MODERNAS

Aunque este taller iba orientado a proporcionar las bases de como funciona el desarrollo moderno del Javascript, aquí os dejo enlaces a algunas tecnologías punteras en desarrollo Javascript.

- Programación reactiva
  - Vue: https://vuejs.org/
  - React: https://es.reactjs.org/
  - Angular: https://angular.io/
- Aplicaciones móviles
  - Cordova: https://cordova.apache.org/
- Aplicaciones de escritorio
  - Electron: https://www.electronjs.org/
- Generador de aplicaciones multiplataforma:
  - Quasar: https://quasar.dev/
    - Framework que soporta Vue y usa Webpack, Babel, EsLint, Cordova y Electron.