FULL STACK DEVELOPER

Módulo 2 – ECMAScript 2015

Jorge Borrego Marqués

Prueba Evaluable

1. Crea una estructura de proyecto HTML y JavaScript.

Creamos un proyecto para la práctica en la carpeta que uso al respecto en el path:

C:\Users\jorge\Desktop\Master FullStack\proyectos\M2_ECMAScript2015

- Consola y comienzo del repositorio en local con el comando “git init”.
- Abro la carpeta en Visual Studio Code y en la raíz abrimos un archivo “index.html”.

Index.html

En la misma raíz del proyecto abrimos carptea “js” y dentro de ella un archivo “main.js”.

En el index linkamos con la etiqueta <script src=”./js/main.js” el archivo javascript.

```
<script src="./js/main.js"></script>
```

****Código del index.html:**

```
<!DOCTYPE html>
<html lang="es">
  <head>
```

```
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Práctica de Evaluación</title>
  </head>
  <body>
    <script src="./js/main.js"></script>
  </body>
</html>
```

2. En el archivo index.html, usa el siguiente código para crear un sencillo formulario.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Práctica de Evaluación</title>
  </head>
  <body>
    <input type="text" id="name" placeholder="Nombre" />
    <input type="text" id="surname" placeholder="Apellidos" />
    <input type="number" id="points" placeholder="Puntuación entre 0 y 12" />
    <button onclick="showUserResult()">Ver listado por consola</button>
    <script src="./js/main.js"></script>
  </body>
</html>
```

En el código HTML cabe destacar la definición de los campos con atributos:

- **type** -> Indica el tipo de dato que se recoge en el campo.
- **Id** -> Identificador de la variable.
- **Placeholder** –> Texto que se muestra en el campo para indicar al usuario que se debe rellenar en cada campo.

Y la definición del botón con la propiedad "***onClick***" dónde invocaremos a la función que hará que los valores de los campos "viajen" a nuestro JavaScript.

3. Desarrolla una clase para alumnos que contenga las propiedades de nombre, apellidos y puntos.

Desarrollo de la clase Alumno(nombre, apellidos, puntos)

```
class Alumno {
    nombre;
    apellidos;
    puntos;

    constructor (nombre, apellidos, puntos){
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.puntos = puntos;
    }
}
```

- En la clase Alumno declaramos las variables requeridas de: nombre, apellidos y puntos.
- Seguidamente método constructor que recibe como parámetros nombre, apellidos y puntos.
- En el constructor iniciamos las variables de clase indicando con la palabra reservada “this” que las variables toman el valor de los parámetros que recibe el método constructor.

4. En la clase anterior, añade un método “set” para establecer el valor de los puntos y otro método “get” que devuelva el string “Apto” si los puntos son iguales o superiores a 5 y “No apto” en caso contrario.

En la clase Alumno añadimos los métodos setPuntos() y getPuntos(), que nos permiten darle valor a la variable puntos y recibir su valor.

```
setPuntos(puntos) {
    this.puntos = puntos;
  }

  getPuntos() {
    if (this.puntos >= 5)
      return `Apto`;
    return `No apto`;
  }
```

- **setPuntos(puntos)**: Recibe como parámetro un number y tomando el valor de este parámetro asignamos a la propiedad de la clase this.puntos.
- **getPuntos():** Desde la función tomamos el valor la propiedad puntos y condicinamos el retorno con un bloque if-else.

5. . Declara una función que, pasados 2 segundos, devuelva una promesa que instancie a partir de los datos del formulario un objeto “alumn” y devuelva un mensaje con su nombre, sus apellidos y si ha resultado apto o no. Para obtener los valores del formulario, usa las siguientes líneas.

Función Promesa:

```
function getAlumno (name,surname,points) {
   return new Promise((resolve, reject) => {

      setTimeout(() => {
         resolve(new Alumno(name,surname,points))
      },2000)
   })
}
```

En este punto introducimos una function para conseguir los datos de un alumno perteneciente a la clase Alumno pasándole como parámetros los campos recogidos del formulario e inicializando al mismo tiempo variables con el mismo nombre, que posteriormente nos servirán de parámetros para la función:

```
let name = document.getElementById("name").value;
let surname =
document.getElementById("surname").value;
let points = document.getElementById("points").value;
```

Una vez pasados los argumentos retornamos el Objeto Promise argumentando dos de sus métodos: reject, resolve.

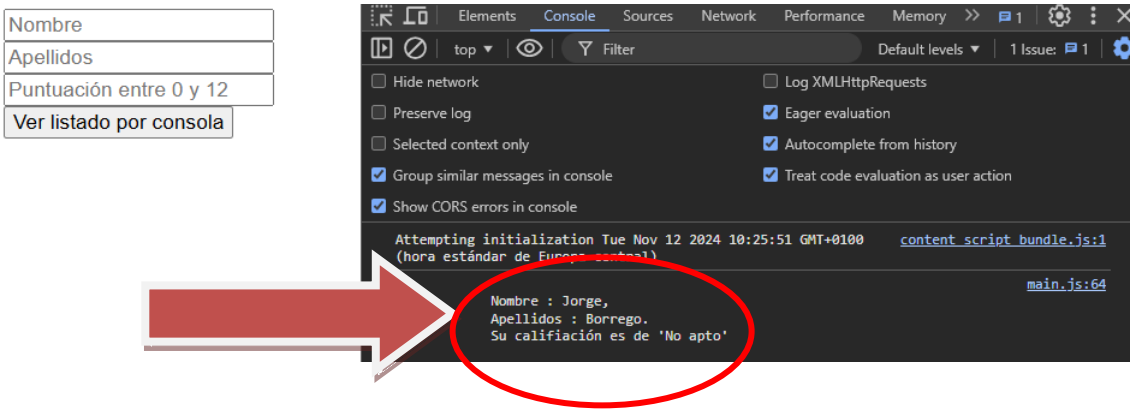
Para resolver la Promesa declaramos (a forma de simulación) una función setInterval(), donde instanciaremos un Objeto Alumno con las propiedades que pasamos como argumentos en la función: name,surname,points.

Al invocar la función definida con la Promesa debemos recoger los datos y mostrarlos, en este caso por consola. Para invocar a la función tomamos de Ejemplo unos valores cualquiera con la simple finalidad de comprobar el funcionamiento del código ya que aún no hemos activado el botón del HTML.

Para recoger los datos usamos la sentencia .then y mostramos la data recogida por consola y añadimos un .cath para recoger el error en caso de producirse.

```
getAlumno('Jorge','Borrego',4) // Los parámetros usados en la function son de Ejemplo
    .then( data => {
        console.log(`
            Nombre : ${data.nombre},
            Apellidos : ${data.apellidos}.
            Su califiación es de '${data.puntos}'
            `);
    })
    .catch(error =>
        console.error(error)
    );
```

Recibimos por Consola el siguiente resultado:



Por consola se muestran los valores de las propiedades del Objeto que pasamos como parámetros de Ejemplo a la hora de invocar a la función “getAlumno()”.

En este caso:

Name: Jorge

Surname: Borrego Marqués

Points: 7

6. Declara la función que es invocada desde el botón index.html con el nombre “showUserResult()” de forma que implemente el patrón “async-await” e invoque la función anterior para recibir el mensaje de la promesa e imprimirlo por consola.

Ahora usando las variables que recogen la información de los campos del formulario en el HTML,

```
let name = document.getElementById("name").value;
let surname = document.getElementById("surname").value;
let points = document.getElementById("points").value;
```

podemos usarlos para declarar la función que activa el botón y que nos pasará la información de los campos . Declaramos el asyc-await para recibir las respuesta del formulario y declarar el Objeto alumno que será recibido por la Promesa anteriormente declarada. La lógica de este proceso radica en declarar una función asíncrona con instrucciones síncronas que permiten al programa seguir con otras tareas mientras la Promesa no se ha resuelto todavía.

```
async function showUserResult(name,surname,points) {
        let alumn = await getAlumno(name,surname,points);
        console.log(`
                Nombre : ${alumn.nombre},
                Apellidos : ${alumn.apellidos}.
                Su califiación es de '${alumn.puntos}'
              `);
}
```

7. Implementa en la función que devuelve la promesa la gestión de errores para que, en caso de que el nombre, los apellidos o los puntos recibidos desde HTML sean strings vacíos, devuelva un mensaje de error con el valor “Datos no válidos”.

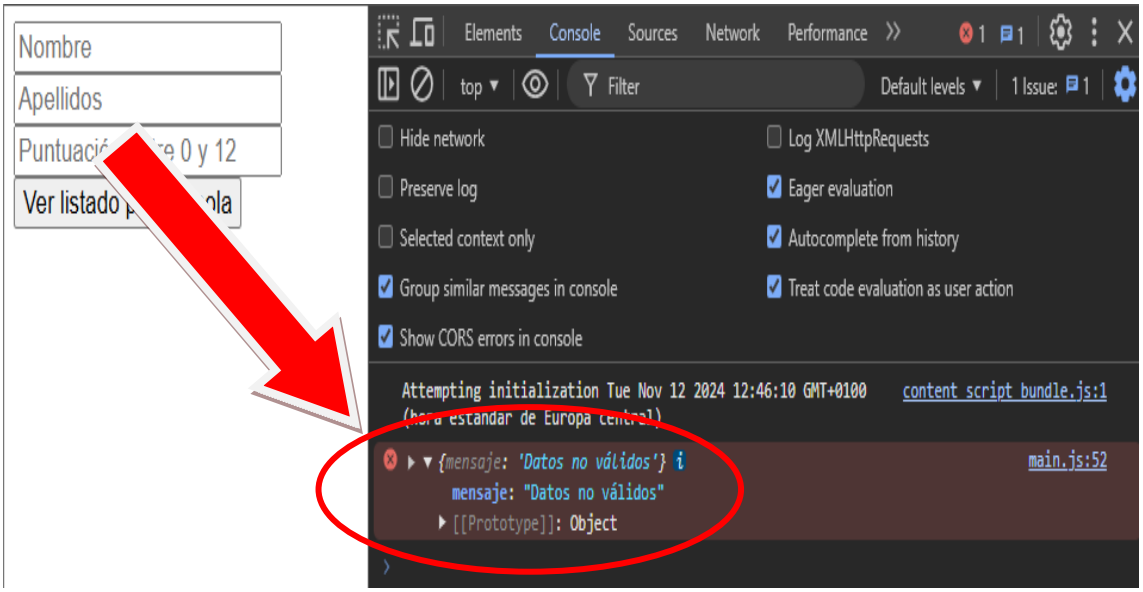
```
function getAlumno (name,surname,points) {
    return new Promise((resolve, reject) => {
        if (
            (name === '') || (surname === '') || (points === '')
        ) {
            reject({mensaje: `Datos no válidos`})
        }
        setTimeout(() => {
            resolve(new Alumno(name,surname,points))
        },2000)
    })
}
```

Anteriormente cuando declaramos la Promesa no hicimos uso del método reject y directamente pasamos a resolver dicha Promesa por medio de la instancia del objeto en un tiempo determinado.

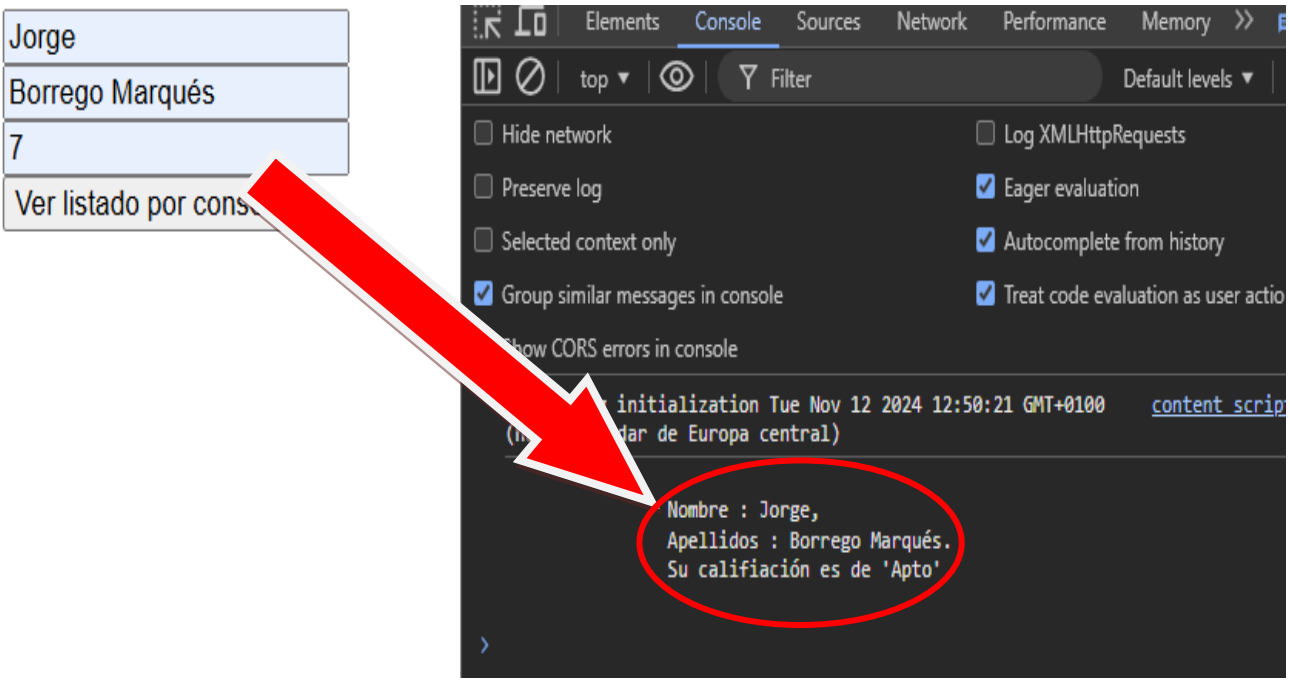
Pues ahora daremos uso del método reject para verificar que los datos enviados para resolver la Promesa cumplen una condición específica(Los campos no pueden ser devueltos como campos vacíos) y si ésta no se cumple entonces retornaremos un mensaje de error con la leyenda de ‘Datos no válidos’

Respuesta por Consola:

- Campos vacíos
-



- Campos rellenos OK.



8. Usa el patrón “async-await” con un “try-catch” para imprimir por consola el error del paso anterior cuando se produzca.

```
async function showUserResult() {
    try {
        let alumn = await getAlumno(
            document.getElementById("name").value,
            document.getElementById("surname").value,
            document.getElementById("points").value

        );
        console.log(`
                Nombre : ${alumn.nombre},
                Apellidos : ${alumn.apellidos}.
                Su califiación es de '${alumn.puntos}'
               `);
    } catch (error) {
        console.error(error);
    }
}
```

Por último en nuestra función asíncrona invocada desde el botón del HTML, es de buena práctica introducir un bloque “try-catch” que recoge los posibles errores a la hora de la recopilación de los datos y evitar la “rotura” del programa donde podremos seguir dando instrucciones o posibles soluciones al cliente si esto ocurriera.

9. Código completo de la tarea.

Como resumen:

Por medio de la práctica hemos conseguido “enlazar” simulando una sincronía entre nuestro formulario HML y las respuestas en JavaScript reaccionando a dicha información. Para ellos hemos declarado clases, instanciado Objetos de esa misma clase, Usado recursos de sincronía como son el Objeto Promise con sus métodos y hemos declarado un protocolo async-await con gestión de error para dar solución a futuros problemas pero de una manera estable y robusta.

```
// ** Desarrollo de la práctica Evaluable en JavaScript **

// * Clase Alumno *
class Alumno {
    nombre;
    apellidos;
    puntos;

    constructor (nombre, apellidos, puntos){
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.puntos = this.getPuntos(puntos);
    }

    setPuntos(puntos) {
       this.puntos = puntos;
    }

    getPuntos() {
        if (this.puntos >= 5)
            return `Apto`;
```

```
        return `No apto`;
    }
}

function getAlumno (name,surname,points) {
    return new Promise((resolve, reject) => {
        if (
            (name === '') || (surname === '') || (points === '')
        ) {
            reject({mensaje: `Datos no válidos`})
        }
        setTimeout(() => {
            resolve(new Alumno(name,surname,points))
        },2000)
    })
}

async function showUserResult() {
    try {
        let alumn = await getAlumno(
            document.getElementById("name").value,
            document.getElementById("surname").value,
            document.getElementById("points").value
        );
        console.log(`
                Nombre : ${alumn.nombre},
                Apellidos : ${alumn.apellidos}.
                Su califiación es de '${alumn.puntos}'
               `);
    } catch (error) {
        console.error(error);
    }
}
```