CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Otras herramientas para el manejo de errores (par. 2)

Alumno: Oswaldo Gael Cervantes Castoño

Código: 219747468

Computación Tolerante a Fallas

Sección D06

Profesor: Michel Emanuel López Franco

Herramientas para el manejo de errores en programación

Objetivo: Genera un ejemplo en el lenguaje de tu preferencia utilizando las herramientas que encontraste.

Desarrollo:

Para el desarrollo de esta actividad decidí hacer uso de la técnica de manejo de “promesas” la cual es una clase de JavaScript que permite gestionar el manejo de operaciones asíncronas. El objeto promesa funciona como un contenedor para un valor que no necesariamente conocemos cuando la promesa es creada. En una operación asíncrona, nos permite asociar distintos controladores al resultado si se cumple con éxito, o a los motivos del error si ha fallado.

JavaScript es “single threaded” (un solo hilo), lo que significa que solo puede ejecutar una acción al mismo tiempo, por lo que utilizar promesas facilita, en buena medida, el control de flujos de datos asíncronos en una aplicación.

Ejemplo codificado

Ya que una de las funciones principales de JavaScript es modificar de forma dinámica el contenido de archivos HTML, en este caso para ejemplificar el uso de promesas utilizaré dos archivos con código HTML: uno que contendrá el script principal en el cual estará la promesa (index.html) y otro con contenido estándar el cual se invocará por medio del método GET de un objeto de tipo XMLHttpRequest() (secondFile.html). En caso de que el archivo sea encontrado, la promesa mostrará el contenido del segundo archivo vaciando el contenido de este en el div que se encuentra en el archivo index.html.

El primer archivo antes de agregar el script de JavaScript (index.html), como ya se mencionó contiene un div con el id “container” sobre el cual se pretende vaciar el contenido del segundo archivo en caso de que la promesa lo encuentre; la estructura de dicho archivo es la siguiente aunque en pasos posteriores se va a agregar y explicar el contenido de JavaScript:

```html
<!DOCTYPE html>
<html lang="en">
<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>Promesas JS</title>
</head>
<body>
    <h2>Promesas en JavaScript</h2>
    <div id="container"></div>
</body>
</html>
```

El segundo archivo tendrá la misma estructura todo el tiempo ya que solamente se utilizará como recurso a consumir desde el archivo index.html para demostrar el uso de promesas. La estructura de dicho archivo es la siguiente:

```html
<!DOCTYPE html>
<html lang="en">
<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>Segundo archivo</title>
</head>
<body>
    <p>
        Este archivo se ha invocado desde index.html
    </p>
</body>
</html>
```

El primer paso es agregar en el archivo index.html una función que nos permita hacer el vaciado del archivo requerido por medio de GET en el div, para lo cual agregamos las etiquetas de script justo antes de cerrar la etiqueta de encabezado, justo como se muestra en la siguiente imagen:

```html
<script>
    function myDisplayer(content) {
        document.getElementById("container").innerHTML = content;
    }
</script>
```

El siguiente paso es crear nuestro objeto promesa de la siguiente forma:

```
<script>
    function myDisplayer(content) {
        document.getElementById("container").innerHTML = content;
    }

    let myPromise = new Promise(function(myResolve, myReject) {

    });
</script>
```

Lo que sigue es crear nuestro objeto de la clase XMLHttpRequest() para posteriormente inicializar los parámetros de la petición que queremos realizar o “abrimos” la petición sobre el archivo que queremos obtener por medio de GET:

```
<script>
    function myDisplayer(content) {
        document.getElementById("container").innerHTML = content;
    }

    let myPromise = new Promise(function(myResolve, myReject) {
        let req = new XMLHttpRequest();
        req.open('GET', "secondFile.html");
    });
</script>
```

A continuación tenemos que hacer dos cosas para completar la petición: primero debemos crear una función para tratar el contenido devuelto después de la petición por medio del método “onload”; en este caso vamos a evaluar el estado de nuestra petición por medio del método “status”, en caso de que el estado de la petición sea 200, significa que la petición se completó de forma exitosa y tenemos la información requerida así que procedemos definir la función de “myResolve” la cual pasará como parámetro al método “then” del objeto promesa la información del archivo solicitado por medio del método “response” y ser tratado como el callback resolve; en caso de no ser 200, significa que no se encontró el recurso solicitado y el estado en este caso sería 404, mismo que se retorna por medio de la función “myReject” al método “then” del objeto promesa para ser tratado como el callback reject. La segunda cosa que debemos hacer para finalizar con éxito la petición es ejecutar el método “send” de nuestra petición para obtener una

respuesta al recurso solicitado y poder procesarla por medio de la función mencionada anteriormente:

```
<script>
    function myDisplayer(content) {
        document.getElementById("container").innerHTML = content;
    }

    let myPromise = new Promise(function(myResolve, myReject) {
        let req = new XMLHttpRequest();
        req.open('GET', "secondFile.html");
        req.onload = function() {
            if (req.status == 200) {
                myResolve(req.response);
            } else {
                myReject("Archivo no encontrado" + ", Error: " + req.status);
            }
        };
        req.send();
    });
</script>
```

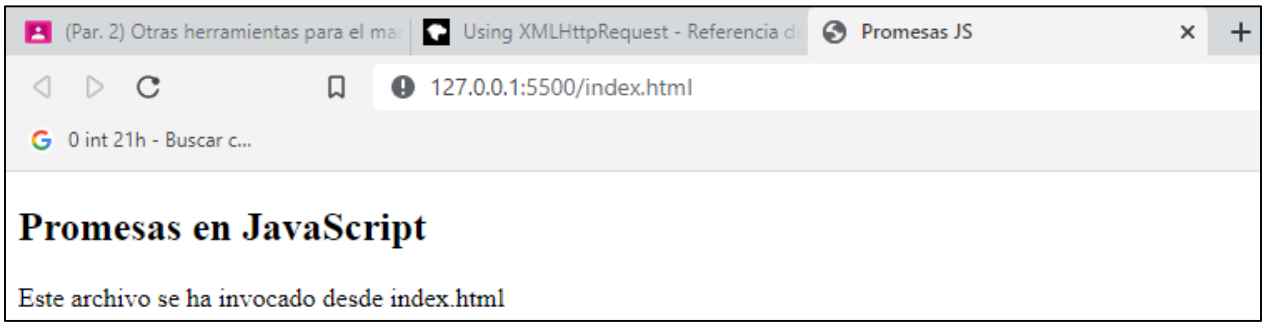
Finalmente tenemos que ejecutar el método “then” del objeto promesa de la siguiente manera:

```
<script>
    function myDisplayer(content) {
        document.getElementById("container").innerHTML = content;
    }

    let myPromise = new Promise(function(myResolve, myReject) {
        let req = new XMLHttpRequest();
        req.open('GET', "secondFile.html");
        req.onload = function() {
            if (req.status == 200) {
                myResolve(req.response);
            } else {
                myReject("Archivo no encontrado" + ", Error: " + req.status);
            }
        };
        req.send();
    });

    myPromise.then(
        function(value) {myDisplayer(value);},
        function(error) {myDisplayer(error);}
    );
</script>
```

En este caso “then” es el que se encarga de gestionar el flujo de la promesa según la función que se ha invocado en líneas previas. De forma general, una promesa puede tener solo un estado a la vez una vez que se ejecuta: cumplida o rechazada y en cualquier caso se llamará solamente una función “myResolve” o “myReject” para que “then” la pueda procesar y mostrar al usuario el resultado que en este caso es el vaciado de la información de la petición en el div de nuestro archivo index.html.

Pruebas del código

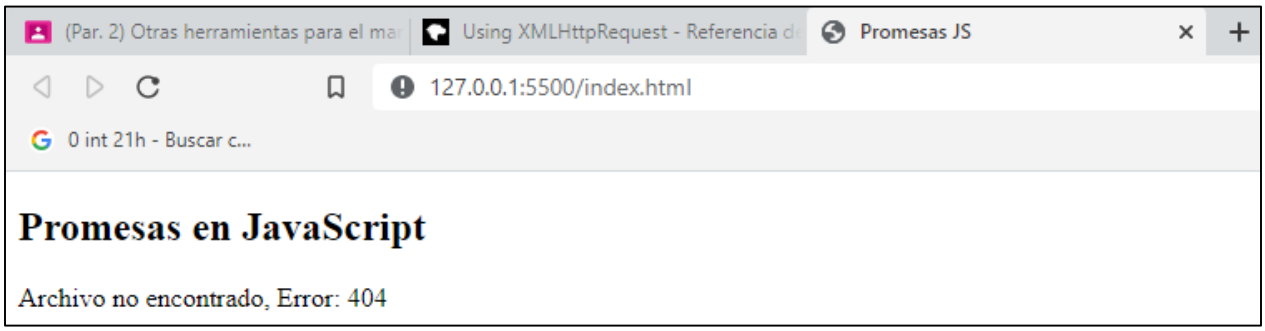
La ejecución del código anterior tal y como quedó en la última imagen no tendrá problemas al ejecutarse ya que el nombre del archivo es correcto y este se encuentra en la misma carpeta que el archivo index.html, así que el resultado al abrir en un navegador dicho archivo es el siguiente:



Nuestra prueba para ver el error que se gestionar por medio de la promesa será modificar el nombre del archivo en el código JavaScript de la siguiente manera:

```
req.open('GET', "secondFilehtml");
```

Y el resultado que la promesa arrojará al no encontrar dicho archivo será el siguiente:



Cervantes Castoño Oswaldo Gael. Computación Tolerante a Fallas.

Link al repositorio con el código fuente: https://github.com/gaelcervantes65/Otras-herramientas-para-el-manejar-errores-par.-2-

Conclusiones

Como conclusiones para la realización de esta actividad puedo mencionar que considero que la principal ventaja de utilizar este tipo de herramientas en el contexto del desarrollo web es que podemos saber de forma instantánea la razón de que no podamos visualizar algún contenido que se consume ya sea desde un archivo local o desde una API remota, ya que sin utilizar una técnica como la del manejo de promesas solo tendríamos una pantalla en blanco sin información asociada con algún tipo de error, por lo que es necesario destacar su importancia así como la de muchos otros tipos de herramientas o técnicas para el manejo de errores y poder desarrollar software que sea muy robusto ante fallos y que este tenga las herramientas necesarias para gestionarlos en caso de que sea inminente su aparición. En general estoy conforme con lo aprendido con este par de actividades ya que considero que son parte fundamental del desarrollo de software que forma parte de nuestro día a día como programadores.

Fuentes de información

- Using XMLHttpRequest. (2022, 6 febrero). MDN Web Docs. Recuperado 8 de febrero de 2022, de https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest
- JavaScript Promises. (s. f.). W3Schools. Recuperado 8 de febrero de 2022, de https://www.w3schools.com/js/js_promise.asp