Página Web: Buscador de GIFs

Alumno: Daniel Llanes
Profesor: Luis Romano

Tecnicatura Superior en Desarrollo de Software

Introducción

El presente proyecto consiste en una aplicación web que permite a los usuarios buscar GIFs utilizando la API de Giphy. La aplicación cuenta con una interfaz simple que incluye un campo de texto para ingresar el término de búsqueda y un botón para iniciar la consulta. Además, la funcionalidad está diseñada para que el usuario también pueda realizar la búsqueda presionando la tecla Enter.

El objetivo de la aplicación es ofrecer una experiencia fluida para buscar y mostrar hasta 9 GIFs que coincidan con el término de búsqueda ingresado. Se hace uso de las tecnologías HTML, CSS y JavaScript, con una llamada a una API externa para obtener los datos.

Descripción del problema

El problema a resolver consiste en implementar una aplicación web que permita a los usuarios buscar y visualizar GIFs relacionados con un término específico. Además, los resultados deben mostrarse de forma organizada en una cuadrícula y actualizarse dinámicamente, eliminando cualquier GIF de búsquedas anteriores.

El desafío principal consiste en gestionar la interacción con una API externa, procesar los datos recibidos, y actualizar el contenido de la página web sin recargarla. También es necesario manejar adecuadamente los errores en caso de que la solicitud a la API falle o no retorne datos válidos.

Explicación Técnica

Estructura:

La estructura del proyecto consta de tres partes:

- Archivo HTML que consta de los componentes que conforman la estructura de la página web
- Archivo CSS que estiliza la página
- Archivo JS en el cual se almacena la lógica y la interacción con la API

HTML:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Buscador de gifs</title>
    <link rel="stylesheet" href="style.css">
    <script src="JS/main.js" defer></script>
</head>
<body>
    <form action=""> <!-- formulario -->
        <div class="div">  <!-- contenedor que contiene el buscador, el botón de
            <input id="busqueda" type="text">
            <button id="boton">Buscar Gifs</button>

            <div id="resultados"></div> <!--contenedor en el cual se mostrarán l
        </div>
    </form>
</body>
</html>
```

En el HTML tenemos los links hacia los estilos (CSS) y hacia la lógica de la API (JS). En el body tenemos un formulario, el cual contiene una etiqueta “div” que a su vez contiene el input de búsqueda, el botón para realizar la acción de buscar y otro contenedor más para guardar los resultados obtenidos.
El input de campo de texto, el botón y el contenedor de resultados son identificados gracias a un id, mientras que el contenedor principal del formulario consta de una clase “div”

CSS:

```css
body{
    background-image: radial-gradient(circle, #7072bb, #0c00f7);
}
/*estilización del campo de texto*/
input{
    margin-bottom: 10px;
    width: 80%;
    padding: 10px;
    font-size: 30px;
    border: 2px, solid, black;
    border-radius: 5px;
    outline: none; /*eliminación del borde por defecto*/
}
```

En el CSS tenemos el body con el color de fondo modificado, para generar un degradé que va desde el centro con un azul claro a los bordes de la página con un azul más brillante y vivo

En el input de búsqueda se utilizó un margen inferior para separar el campo de texto del botón de búsqueda, un ancho del 80% de la pantalla, un relleno de 10 píxeles alrededor del texto, un tamaño de fuente de 30 píxeles, borde básico y redondeado y se quitó el borde por defecto que viene con la etiqueta.

```css
button{
    width: 300px;
    height: 50px;
    display: block; /*se muestra como bloque para qu
    margin: auto; /*se establecen margenes auto para
    background-color: rgb(5, 5, 83);
    color: white;
    padding: 40px, 40px;
    font-size: 30px;
    border: none; /*se elimina el borde por defecto
    border-radius: 5px;
    cursor: pointer; /*se indica que el cursor se mu
    transition: background-color 0.3s ease;; /*trans
    margin-bottom: 10px;
}

/*estilización del botón al pasar el cursor por enci
button:hover{
    background-color: #3c3f97;
}
```

Para el botón se utilizó un ancho de 300 pixeles y un alto de 50. Se muestra como elemento de bloque para aparecer por debajo del campo de texto, se estableció margen automático para que se centre en el medio horizontalmente, un color azul oscuro de fondo y blanco para las letras, un relleno alrededor de 40 píxeles y tamaño de fuente de 30. Se eliminó el borde que trae por defecto y se redondeó, el cursor se cambió para que aparezca como una mano cuando se pase por encima de él, se aplicó una transición para el cambio de color y se le dió un margen inferior de 10 píxeles.

```
.div{
    width: auto;
    text-align: center;
}
/*estilización del contenedor de resultados*/
#resultados{
    display: grid; /*se muestra como una grilla
    grid-template-columns: repeat(3, auto); /*s
}
```

Para el contenedor principal se establece un margen automático para que los elementos se centraran horizontalmente al igual que el texto.
En el contenedor de los resultados se asignó una disposición de grilla para mostrar los elementos en una cuadrícula de 3x3.

Javascript:

```
const api_key = "PAEsLG5YrRyvXFhbOM5uCgc08fMqIxmF";


// Función que realiza la llamada a la API de Giphy
// Recibe como parámetros la clave de la API (ak) y el término de búsqueda (termino)
const llamada = (ak, termino) =>{
    // se construye la URL con los valores asignados a los parámetros, aquí también definim
    const url = `https://api.giphy.com/v1/gifs/search?api_key=${ak}&q=${termino}&limit=9&of

    // Se devuelve la promesa resultante de la función fetch, que hace la petición a la API
    return fetch(url)
}
```

Primero se guarda en una constante la clave proporcionada por la API. Luego se define la función de llamada que construye la url con los parámetros (clave de la API y término de búsqueda). Luego retorna fetch de la url construida en la función para manejar la promesa.

```
function buscarGifs(){
    let valor = document.getElementById("busqueda").value

    //Eliminar imagenes anteriores para no acumular los resultados
    const gif_anteriores = document.querySelectorAll("img");
    gif_anteriores.forEach(img => img.remove());

    //se llama a la API con la clave de la API y el valor buscado
    llamada(api_key, valor)
    .then((response) => {

        //si la respuesta no es la esperada, manda un error
        if (!response.ok) {
            throw new Error("Error en la respuesta de red")
        }
        //se convierte la respuesta en fomati JSON
        return response.json();
    })
    .then((results) => {
        // caso contrario itera sobre los elementos devueltos por la API (hasta 9
        results.data.forEach(element => {
            //En cada iteración:

            // muestra la url de cada imagen por la consola
            console.log(element.images.original.url)

            // se guarda en una constante un elemento img
            const img = document.createElement("img");
            // se modifica el src, haciendo que la imagen sea la url devuelta por
            img.setAttribute("src", element.images.original.url);
            // se guarda el resultado como un hijo del contenedor de resultados en
            resultados.appendChild(img);
        });
    })
    .catch((error) => {
        // Captura y muestra errores en la consola
        console.error("Error en la consulta: " + error.message);
```

Se crea la función buscarGifs, que guarda en una constante el valor ingresado en el campo de texto. Luego elimina las imágenes anteriores seleccionando todos los elementos img que se encuentren en el DOM y eliminandolos uno por uno a través de un forEach

Después de esto se hace la llamada a la API con la función anteriormente mencionada, pasándole la clave de la API y el valor guardado que se introdujo en el campo de texto. Si la respuesta no es la deseada lanza un error, de lo contrario captura todos los elementos e itera sobre ellos, en cada iteración muestra por consola la URL de la imágen, crea un nuevo elemento y configura su URL para que la misma sea la misma de la devuelta por la API, finalmente la añade al DOM específicamente en el contenedor de resultados.

```
}
// se agrega un escuchador de eventos al botón de buscarGifs para que ejecute la funci
boton.addEventListener("click", buscarGifs)

// se agrega un escuchador de eventos al campo de texto para que al presionar la tecla
document.getElementById("busqueda").addEventListener("keydown", function(event) {
    if (event.key === "Enter") { //si la letra presionada es Enter:
        // esta linea previene el comportamiento predeterminado del fomulario al presi
        event.preventDefault();
        buscarGifs();  // Llama a la función de búsqueda
    }
});
```

Finalmente, se asigna la función “buscarGifs” a un manejador de eventos en el elemento botón al ser presionado con un click y también al elemento input del campo de búsqueda, para que cuando se presione la tecla enter llame a la misma función

Conclusión

En este proyecto se han reforzado las habilidades para la construcción de páginas web utilizando HTML, CSS y Javascript. Se consumió una API y aprendimos a manipular el DOM, utilizando promesas manejandolas con la función fetch y logrando cumplir con los requerimientos. El buscador de GIFs presenta una interfaz simple e intuitiva que devuelve hasta 9 imágenes GIF según el criterio de búsqueda. Ordenadas en una grilla de 3x3 gracias a la estilización de CSS. Su repositorio en GitHub cuenta con todos los archivos para su funcionamiento, un archivo README.md para su utilización y el presente informe.