GUÍAS

para hacer “gifOS”

Nota: Para que un proyecto apruebe, tiene que cumplir con todas las condiciones del Checklist.

Guía 1: Estructura de proyecto y themes _

Este proyecto tiene como objetivo integrar los conocimientos previos de maquetado con los fundamentos de la programación aprendidos en este módulo.

El desafío será desarrollar un catálogo de GIFs que contará con funcionalidad de búsqueda, interacción con una API externa y captura de video.

Podemos comenzar creando una estructura básica de carpetas para mantener todos nuestros archivos ordenados. A continuación debemos crear la estructura de nuestros html, sin olvidar de vincular todos los recursos importantes, incluyendo scripts, fuentes y hojas de estilo.

En los requerimientos del proyecto se nos pide además implementar dos estilos distintos que el usuario podrá seleccionar desde un menú, es importante que tengamos en cuenta esto a la hora de estilar.

Teniendo esta estructura inicial podemos proceder con el maquetado del sitio, dejando preparadas las bases para agregarle funcionalidad más adelante.

El objetivo del proyecto es replicar un esquema laboral de un Desarrollador Front-End, por lo que el diseño debe ser igual al de referencia.

Checkpoint

Para comprobar que todo está funcionando como esperamos podemos probar agregando un estilo global en el css, por ejemplo un color de fondo a la etiqueta <body>. También sería ideal agregar texto para comprobar que la fuente funciona y un console.log dentro del archivo js para verificar que nuestro script está cargando.

Tip: para verificar que no haya errores en el proceso podemos abrir las herramientas de desarrollo de nuestro navegador y comprobar que no haya errores de ningún tipo en la consola.

Guía 2: Integración con una API externa _

Antes de continuar con el desarrollo, Giphy nos pide como requisito para utilizar su API que creemos una “aplicación” y generemos nuestra key.

Para generar esta API Key

1. Visitar la página de Giphy developers https://developers.giphy.com
2. Crear una cuenta
3. Seguir los pasos para generar la key

Al terminar estos pasos deberíamos tener nuestra API Key, es un string de 32 caracteres conformado por números y letras mayúsculas y minúsculas.

Teniendo esta información ya podemos hacer requests a la API, vamos a comenzar con la búsqueda. Para hacer todas nuestras requests usaremos Fetch y será importante tener la documentación oficial como referencia.

Empezaremos por la barra de búsqueda, para lo cual debemos realizar una request de tipo GET al endpoint de búsqueda.

Para eso debemos declarar una función que se llame cada vez que presionemos el botón “Buscar” y que reciba como parámetro el string

que deseamos buscar, en este caso el valor actual de la etiqueta <input> que compone a nuestra barra de búsqueda. Una vez que contamos con este valor el resto de la función debería contener nuestro GET.

Si seguimos las instrucciones de la documentación nuestro código debería ser similar a este:

```
function getSearchResults(search) {
   const found =
fetch('http://api.giphy.com/v1/gifs/search?q=' + search +
'&api_key=' + apiKey)
       .then((response) => {
          return response.json()
       }).then(data => {
           return data
       })
       .catch((error) => {
           return error
       })
   return found
}
```

Una vez que el resultado de nuestra búsqueda “vuelve” lo recibimos nos ocuparemos de darle un formato que nos sirva, en este caso JSON.

Teniendo esta respuesta ya deberíamos contar con todas las herramientas para generar nuestra galería de búsquedas y finalizar la funcionalidad esperada para index.html. Se deberán resolver los cambios requeridos en el DOM.

Checkpoint

Habiendo realizado nuestro primer fetch, podemos comprobar que efectivamente nos estamos comunicando con la API de Giphy realizando un console.log(data) dentro del scope del segundo .then().

Al llamar a nuestra función deberíamos ver en consola una respuesta parecida a esta:

```
{data: Array(25), pagination: {…}, meta: {…}}
        data: (25) [{…}, {…}, {…}, {…}, {…}, {…}, {…},
        {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
        {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
        meta: {status: 200, msg: "OK", response_id:
        "5d217f854b4567716b5626d4"}
        pagination: {total_count: 26015, count: 25, offset: 0}
        __proto__: Object
```

Tip: Dedicá un tiempo a explorar el objeto GIF en consola e interiorizarte con su contenido. Dentro de Data se puede encontrar un montón de información útil que te va a servir a la hora de armar la galería de imágenes.

Guía 3: Obteniendo video desde el browser _

Para capturar video primero debemos obtenerlo por algún medio, y para eso vamos a utilizar el método navigator.mediaDevices.getUserMedia. Podemos encontrar más información al respecto en la documentación oficial.

El segundo requisito para obtener video en nuestra página es tener un contenedor que pueda reproducirlo, en este caso una etiqueta <video>. Agreguemos entonces (si no tenemos ya) una etiqueta <video> dentro del layout de upload.html.

Podemos entonces declarar una función que inicie las funciones de captura.

Dentro de la función llamaremos al método navigator.mediaDevices.getUserMedia y le pasaremos las opciones de video que deseamos.

```
function getStreamAndRecord () {
```

```
  navigator.mediaDevices.getUserMedia({
     audio: false,
     video: {
     height: { max: 480 }
    }
 })
    .then(function(stream) {
      video.srcObject = stream;
      video.play()
}
```

El método nos devuelve una promesa, por lo cual debemos manejar la respuesta de manera asíncrona. Una vez que tenemos nuestro stream podemos utilizarlo como src de la tag <video> y llamar al método play() para que comience a reproducirse.

Ahora debemos implementar que la función getStreamAndRecord se llame cada vez que presionamos el botón “Grabar” y que el video aparezca o desaparezca según haga falta, esto se puede resolver con listeners e interactuando con el DOM.

Checkpoint

Si seguimos las instrucciones correctamente deberíamos poder lograr que la cámara empiece a funcionar cada vez que presionamos nuestro botón “Grabar”.

Guía 4: Grabando video _

Ahora que somos capaces de ver el stream de nuestra webcam el siguiente paso es capturarlo. Para este paso vamos a utilizar una librería externa llamada recordRTC.

Para evitar descargar la librería a nuestras computadoras, usaremos un CDN, es decir, una versión de la librería que se encuentra online.

Podemos utilizarla de la siguiente manera:

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/RecordRTC/5.5.8/R
ecordRTC.js" type="text/javascript"></script>
```

Para comenzar a interactuar con la librería, debemos crear un objeto recorder. Este objeto recibirá opciones y cuenta con varios métodos que podemos utilizar para grabar, por ejemplo:

```
recorder = RecordRTC(stream, {
       type: 'gif',
       frameRate: 1,
       quality: 10,
       width: 360,
       hidden: 240,
       onGifRecordingStarted: function() {
         console.log('started')
       },
     });
```

Para entender qué hace cada una de las opciones siempre podemos volver a revisar la documentación.

El recorder cuenta con muchos métodos, pero en este momento solo nos importan startRecording y stopRecording.

StartRecording, como su nombre lo indica comienza la grabación y no requiere de ningún parámetro extra.

stopRecording recibe como parámetro un callback, donde le indicamos que hacer con la información grabada una vez que paramos la grabación.

Entonces, sabiendo esto podemos continuar con los siguientes pasos, al apretar el botón ‘grabar’ creamos un nuevo recorder y le decimos que inicie la grabación con startRecording y cuando apretamos el botón ‘stop’ llamamos a stopRecording pasándole como callback nuestra función anteriormente definida.

Es importante en este paso agregar todos los cambios del DOM necesarios para que los botones nos muestran cuando estamos grabando y cuando no.

Checkpoint

Al finalizar estos pasos deberíamos ser capaces de iniciar una grabación y pararla, esto se puede confirmar a través del console.log que colocamos al finalizar la grabación.

Guía 5: Generando un archivo para subir (upload) _

Ya tenemos nuestro proceso de grabación funcionando, pero ¿cómo accedemos al archivo para subirlo a Giphy?

Para realizar esto debemos primero revisar dos conceptos, el método getBlob y FormData

getBlob() es un método del objeto recorder creado con RecordRTC que nos permite acceder a los datos grabados.

Un blob es una manera de guardar datos que eventualmente se puede transformar en un archivo para ser leído por el sistema operativo. Podemos profundizar más sobre ellos aquí.

FormData es un objeto que nos permite darle formato clave valor a nuestra información para enviarla de manera ordenada a través de body de un POST. Para agregar información utilizamos un método del objeto llamado .append que recibe dos parámetros clave y valor. Podemos revisar su documentación aquí.

Teniendo estos conceptos, ahora solo nos queda crear nuestro archivo y organizar la información para enviar.

Podemos crear nuestro objeto FormData de la siguiente manera:

```
let form = new FormData();
```

E incluir nuestra grabación al form usando el método .append

```
 form.append('file', recorder.getBlob(), 'myGif.gif');
```

Como podemos ver, append recibe “file” como clave, nuestro blob como segundo parámetro y en tercer lugar un nombre de archivo, en este caso elegimos myGif.gif Con estos pasos ya tenemos lista la información para subir a la API de Giphy.

Tip: Para mostrar previews de nuestro archivo podemos utilizar el método getBlob combinado con createObjectURL para crear una url que puede ser usada como atributo src de una etiqueta

Habiendo seguido estos pasos ya estamos listos para subir nuestro archivo a través de la API utilizando el método POST.

Checkpoint

Podemos confirmar que nuestra FormData se está creando de manera correcta utilizando otro método de FormData, .get(). Este método recibe como parámetro la clave de la información que estamos buscando, como en nuestro caso la clave es “file”.

Podemos hacer un console.log() del método y revisar el contenido en la consola del navegador.

Para este código:

```
let form = new FormData();
 form.append('file', recorder.getBlob(), 'myGif.gif');
 console.log(form.get('file'))
```

Deberíamos obtener una respuesta parecida a:

File {name: "myGif.gif", lastModified: 1562549678745, lastModifiedDate: Sun Jul 07 2019 22:34:38 GMT-0300

(Argentina Standard Time), webkitRelativePath: "", size: 1973490, ...}

Guía 6: Guardando y obteniendo información de Local Storage _

Nuestro proyecto está casi listo, solo nos falta guardar y mostrar los GIFs que capturamos.
Para esto vamos a utilizar una funcionalidad del browser llamada local storage.

Local storage, como su nombre lo indica, es una sección donde el navegador puede guardar información sobre un sitio web si su desarrollador así lo desea. Esto permite persistir información sin la necesidad de una base de datos. Podemos encontrar la documentación oficial aquí.

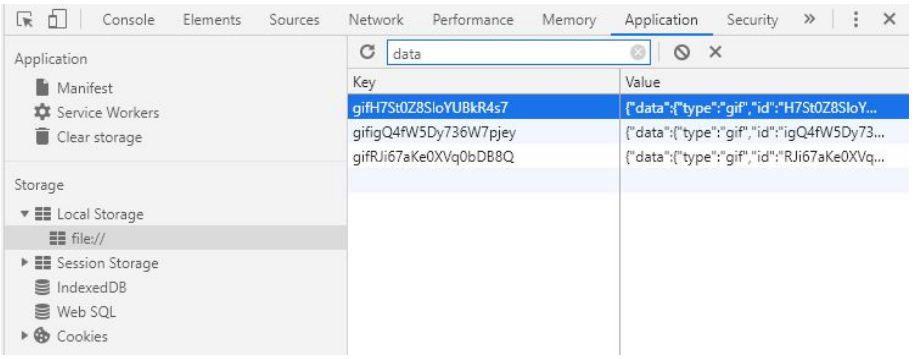
Para esto utilizaremos un método llamado localStorage.setItem que recibe dos parámetros, el primero es el nombre de lo que queremos guardar, y el segundo la información. Es importante destacar que debido a limitaciones del browser la información deberá ser guardada en formato string.

Teniendo nuestra información guardada, podemos mostrarla la próxima vez que ingresamos a la página. Para esto utilizaremos otro método de local storage llamado localStorage.getItem() que recibe un parámetro, en este caso las keys de los items que queremos obtener.

Local Storage es una lista, y como tal puede ser recorrida con bucles.

Checkpoint

Podemos comprobar el funcionamiento correcto de nuestra función haciendo un upload y abriendo las herramientas de desarrollo del navegador en la sección Application -> localstorage. Si nuestro GIF se termina de subir correctamente y la información se salva a Local Storage deberíamos ver algo similar a esto:


Console
Elements
Sources
Network
Performance
Memory
Application
Security
Application
Manifest
Service Workers
Clear storage
Storage
Local Storage
file://
Session Storage
IndexedDB
Web SQL
Cookies
data
Key
Value
gifH7St0Z8SloYUBkR4s7
{"data":{"type":"gif","id":"H7St0Z8SloY...
gifigQ4fW5Dy736W7pjey
{"data":{"type":"gif","id":"igQ4fW5Dy73...
gifRJi67aKe0XVq0bDB8Q
{"data":{"type":"gif","id":"RJi67aKe0XVq...