

UNIVERSIDAD PRIVADA DEL VALLE
FACULTAD DE INFORMÁTICA Y ELECTRÓNICA
INGENIERÍA DE SISTEMAS INFORMÁTICOS



PROGRAMACION WEB I

TRABAJO FINAL DE SEMESTRE: GALERIA DE ARTE: MANUAL TECNICO

ESTUDIANTES: CHRIS JEREMY ROJAS CONDORI

DOCENTE: ING. HENRY MIRANDA ORDONEZ

GRUPO: C

GESTIÓN: 2025

LA PAZ – BOLIVIA

INDICE

MANUAL TECNICO – MORPHO ETHER	2
1. Descripción General del Sistema.....	3
2. Arquitectura.....	3
3. Logica detras del sistema.....	4
3.1 Variables Generales.	4
3.2 Inicialización:.....	4
3.3 Funciones obtenerObrasIDs(), cargarBloque() y obtenerDatosObra()	4
4. Funciones para que se muestren as obras.....	6
4.1 Función mostrarObra(obra, i)	6
4.2 Función abrirModal(obra, idObra).	7
4.3 Función generarContenidoModal(obra, idObra)	7
4.4 Función cargarCalificacionesPrevias(idObra)	8
4.5 Función guardarCalificacion(idObra)	9
4.6 Función generarEstrellas(p).....	10
4.7 Función actualizarPromedio(idObra).	10
5. Responsividad y orden por columnas.....	10

MANUAL TECNICO – MORPHO ETHER

1. Descripción General del Sistema

Morpho Ether es una aplicación web que usa como fuente de datos a la API pública del The Metropolitan Museum of Art (MET) para mostrar obras de arte, desplegarlas en una interfaz gráfica en una página web y permite dar calificaciones a las obras de arte que se muestran en la pantalla de inicio mediante almacenamiento local (LocalStorage).

La tecnología usada en este programa es:

- HTML - estructura de la interfaz.
- CSS/ Flex/Grid - para un diseño responsive y ordenado por columnas.
- JavaScript - lógica, donde están los links del Api, modal, lógica de las calificaciones.
- API The Metropolitan Museum of Art (MET) - fuente de datos para las imágenes y datos de las obras de arte.
- LocalStorage - para que se guarde en el navegador local las calificaciones que se hacen.
- VisualStudio – Es la IDE que se uso para desarrollar y conectar todo para la pagina web.

2. Arquitectura

La aplicación sigue una arquitectura modular simple:

```
/ProyectoWebRepositorio  
 /Proyecto1erSemWEB.html  
 /CSS  
   /EstiloProyectoWEB1er.css  
 /JavaScrpit  
   /DetrasDeTodosWEB2do.js  
 /Imagenes  
   /ImagenesFIGMA  
   /ImagenesProyect  
   /LogoTransparente.png  
   /Logo.png  
 • EstiloProyectoWEB1er.css.- Es la hoja de estilos que maneja el programa, donde se encuentra la expansividad.  
 • Proyecto1erSemWEB.html.- Es la estructura que tiene la página web.
```

- /DetrasDeTodosWEB2do.js.- Es la hoja de lógica que usa la página web, que hace que todos los botones y funcionalidades dentro de la página funcionen.
- Imágenes.- Es la carpeta donde se guardan las imágenes que se usaron en la página web, como el logo de la página.

3. Logica detrás del sistema.

3.1 Variables Generales.

```
const contenedor = document.getElementById("contenedor-obra");
const modal = document.getElementById("modal");
const cerrar = document.getElementById("cerrar");
const modalInfo = document.getElementById("modal-info");
const btnCargarMas = document.getElementById("cargar-mas");

let allIDs = [];
let obrasMostradas = 0;
const obrasPorCarga = 20;

const calificaciones = {};
```

3.2 Inicialización:

Al iniciar la página se ejecuta la carga de 20 obras con la función cargar20prim() que está en la hoja de logica /DetrasDeTodosWEB2do.js.

```
async function cargar20prim() {
    allIDs = await obtenerObrasIDs();
    cargarBloque();
}
```

Esta función asincrona espera a que se cumpla la promesa, en este caso la promesa es la función obtenerObrasIDs(), para poder cargar los IDs de las obras en la lista allIDs.

Luego de ejecutar eso se carga la función de cargarBloque() para cargar un bloque de 20 obras.

3.3 Funciones obtenerObrasIDs(), cargarBloque() y obtenerDatosObra()

3.3.1 Función obtenerObrasIDs().

```
async function obtenerObrasIDs() {
    const url = await
fetch("https://collectionapi.metmuseum.org/public/collection/v1/objects?departmentIDs=11");
    const data = await url.json();
    return data.objectIDs;
}
```

La función obtenerObrasIDs es una función asíncrona, y se usa el await para poder hacer consultas con fetch que es una función de consulta HTTP, se hace la consulta a la API The Metropolitan Museum of Art (MET) con su respectivo link y se guarda en una variable llamada url, luego se convierte el contenido de la url en un objeto json(), y se guarda en data, luego se selecciona solo los IDs con data.objectIDs y se devuelven estos.

3.3.2 función cargarBloque()

```
async function cargarBloque() {
    let obrasCargadas = 0;
    let cont = obrasMostradas;

    while (obrasCargadas < obrasPorCarga && cont < allIDs.length) {
        const obra = await obtenerDatosObra(allIDs[cont]);
        cont++;
        if (!obra.imagen) continue;

        mostrarObra(obra, obrasMostradas);
        obrasMostradas++;
        obrasCargadas++;
    }

    if (obrasMostradas >= allIDs.length) {
        btnCargarMas.style.display = "none";
    }
}
```

Esta función maneja dos variables, obrasCargadas que será el contador de cuantas veces se tiene que repetir, y serán solo 20, ya que en la condicional del while se menciona que será menor que la variable global obrasPorCargar que su valor es 20. Luego maneja la variable cont, que será un contador que hará referencia a la variable global obrasMostradas que es un contador general de cuantas obras se contaron ya en total, y esta variable cont se usa en la condicional del while para verificar que no se hayan usado todas las IDs de obras ya.

Dentro del while se usa un await para esperar la ejecución de la función obtenerDatosObra() que usa allIDs con las IDs ya cargadas que se usa como argumento, y esta misma tiene como argumento el cont para ver que no se repita ninguna obra.

Se usa un if para ver que la imagen de la obra no esté vacío, en caso de que si, entonces se salta a la siguiente vuelta sin contar la que se estaba haciendo ya que obrasMostradas y obrasCargadas no aumentan.

Luego se usa un if para ocultar el botón de “cargar más” si es que se llega al límite de obras que se pueden cargar.

3.3.3 Función obtenerDatosObra()

```
async function obtenerDatosObra(objectID) {
    const url = await
fetch(`https://collectionapi.metmuseum.org/public/collection/v1/objects/${objectI
D}`);
    const data = await url.json();
    return {
        titulo: data.title || "Sin título",
        autor: data.artistDisplayName || "Desconocido",
        anio: data.objectDate || "Sin fecha",
        descripcion: data.creditLine || "",
        imagen: data.primaryImage || ""
    };
}
```

Esta función no diferencia mucho en funcionamiento de la función obtenerObrasIDs(), solo que esta devuelve un objeto con los datos que se mostrarán de la obra y se almacena en una variable obra en la función CargarBloque().

4. Funciones para que se muestren las obras.

4.1 Función mostrarObra(obra, i)

```
function mostrarObra(obra, i) {
    const idObra = "obra-" + i;

    const divImagen = document.createElement("div");
    divImagen.classList.add("imagen-contenedor");
    divImagen.innerHTML =
        `★ 0.0</span>
        
        `;
    contenedor.appendChild(divImagen);

    if (!calificaciones[idObra]) calificaciones[idObra] = [];

    divImagen.addEventListener("click", () => abrirModal(obra, idObra));
}
```

La función mostrarObra genera un contenedor visual para cada obra obtenida de la API o de una lista local. Crea dinámicamente un div con la clase imagen-contenedor que incluye la imagen de la obra y un span donde se mostrará el promedio de calificaciones. Asigna un

ID único idObra para identificar la obra y su información de calificaciones. Inicializa un arreglo vacío en calificaciones si no existe aún para esta obra, permitiendo guardar futuras calificaciones. Finalmente, añade un evento click que abrirá el modal correspondiente al hacer clic en la obra, mostrando más información y permitiendo calificarla.

4.2 Función abrirModal(obra, idObra).

```
function abrirModal(obra, idObra) {
    modal.style.display = "flex";

    // Cargar calificaciones actuales
    const lista = calificaciones[idObra];
    const suma = lista.reduce((s, c) => s + c.puntaje, 0);
    const total = lista.length;
    const promedio = total ? (suma / total).toFixed(1) : "0.0";

    document.getElementById(`promedio-img-${idObra}`).textContent = `⭐
${promedio}`;

    modalInfo.innerHTML = generarContenidoModal(obra, idObra);

    // Mostrar calificaciones previas
    cargarCalificacionesPrevias(idObra);

    // Botón de guardar
    const btnGuardar = document.getElementById(`guardar-${idObra}`);
    btnGuardar.addEventListener("click", () => guardarCalificacion(idObra));
}
```

La función abrirModal es responsable de mostrar la información completa de la obra seleccionada dentro de un modal. Hace visible el modal cambiando su propiedad display a flex. Calcula y actualiza el promedio de calificaciones existentes de la obra sobre la imagen correspondiente. Genera el contenido del modal, incluyendo título, autor, año, descripción, imagen y formulario de calificación mediante la función generarContenidoModal. Carga las calificaciones previas de la obra para que se muestren al abrir el modal mediante la función cargarCalificacionesPrevias. Finalmente, añade un evento al botón GUARDAR para registrar nuevas calificaciones usando la función guardarCalificacion.

4.3 Función generarContenidoModal(obra, idObra)

```
function generarContenidoModal(obra, idObra) {
    return `
```

```

<h2>${obra.titulo}</h2>
<p><strong>Autor:</strong> ${obra.autor}</p>
<p><strong>Año:</strong> ${obra.anio}</p>
<p><strong>Descripción:</strong> ${obra.descripcion}</p>


<div class="form-calificacion">
    <input type="text" id="nombre-${idObra}" placeholder="Tu nombre">
    <select id="puntaje-${idObra}">
        <option value="1">1 estrella</option>
        <option value="2">2 estrellas</option>
        <option value="3">3 estrellas</option>
        <option value="4">4 estrellas</option>
        <option value="5">5 estrellas</option>
    </select>
    <textarea id="comentario-${idObra}"
placeholder="Comentario..."></textarea>
    <button id="guardar-${idObra}">GUARDAR</button>
</div>

<div id="contenedor-${idObra}" class="contenedor-calificaciones"></div>
`;
}

}

```

La función generarContenidoModal devuelve un bloque de código HTML que representa toda la información de la obra que se mostrará en el modal. Incluye el título, el autor, el año, la descripción y la imagen de la obra. También incluye un formulario de calificación con campos para el nombre del usuario, el puntaje seleccionado mediante un select y un área de texto para comentarios, junto con un botón para guardar la calificación. Este contenido se inserta dinámicamente en el modal cuando se abre.

4.4 Función cargarCalificacionesPrevias(idObra)

```

function cargarCalificacionesPrevias(idObra) {
    const cont = document.getElementById(`contenedor-${idObra}`);

    calificaciones[idObra].forEach(c => {
        const estrellas = generarEstrellas(c.puntaje);
        cont.innerHTML += `
            <div class="calificacion">
                <h4>${c.nombre} – ${estrellas}</h4>
                <p>${c.comentario}</p>
            </div>`;
    });
}

```

```
});  
}
```

La función cargarCalificacionesPrevias recorre el arreglo de calificaciones de la obra correspondiente y genera bloques HTML con cada calificación ya guardada. Calcula el número de estrellas basado en el puntaje y los muestra junto con el nombre y el comentario del usuario dentro del contenedor de calificaciones del modal. Esto permite que al abrir nuevamente el modal se mantengan visibles todas las calificaciones anteriores.

4.5 Función guardarCalificacion(idObra)

```
function guardarCalificacion(idObra) {  
    const nombre = document.getElementById(`nombre-${idObra}`).value;  
    const puntaje = parseInt(document.getElementById(`puntaje-${idObra}`).value);  
    const comentario = document.getElementById(`comentario-${idObra}`).value;  
  
    if (!nombre || !comentario) {  
        alert("Completa todos los campos.");  
        return;  
    }  
  
    const nueva = { nombre, puntaje, comentario };  
    calificaciones[idObra].push(nueva);  
  
    actualizarPromedio(idObra);  
  
    const cont = document.getElementById(`contenedor-${idObra}`);  
    const estrellas = generarEstrellas(puntaje);  
    cont.innerHTML += `  
        <div class="calificacion">  
            <h4>${nombre} – ${estrellas}</h4>  
            <p>${comentario}</p>  
        </div>`;  
  
    // Limpiar campos  
    document.getElementById(`nombre-${idObra}`).value = "";  
    document.getElementById(`comentario-${idObra}`).value = "";  
}
```

La función guardarCalificacion se encarga de registrar la calificación ingresada por el usuario en el modal. Obtiene el nombre, el puntaje y el comentario de los campos del formulario, valida que no estén vacíos y luego guarda la información en el arreglo calificaciones correspondiente a la obra. Actualiza el promedio de calificaciones

mostrado en la imagen y agrega un bloque HTML con la nueva calificación dentro del contenedor de calificaciones. Finalmente, limpia los campos del formulario para permitir nuevas entradas.

4.6 Función generarEstrellas(p)

```
function generarEstrellas(p) {  
    return "★★★★★".slice(0, p) + "☆☆☆☆☆".slice(0, 5 - p);  
}
```

La función generarEstrellas recibe como parámetro un número entero p que representa el puntaje otorgado a una obra. Genera una cadena de texto que contiene un total de cinco símbolos, combinando estrellas llenas (★★★★★) según el puntaje p y estrellas vacías (☆☆☆☆☆) para completar hasta cinco. Esto permite mostrar visualmente la calificación de manera clara y uniforme en el frontend.

4.7 Función actualizarPromedio(idObra).

```
function actualizarPromedio(idObra) {  
    const lista = calificaciones[idObra];  
    const suma = lista.reduce((s, c) => s + c.puntaje, 0);  
    const total = lista.length;  
    const promedio = (suma / total).toFixed(1);  
  
    document.getElementById(`promedio-img-${idObra}`).textContent = `★  
${promedio}`;  
}
```

La función actualizarPromedio recibe como parámetro el idObra correspondiente a la obra a actualizar. Toma la lista de calificaciones de esa obra desde el objeto global calificaciones, suma todos los puntajes existentes y calcula el promedio dividiendo la suma entre la cantidad de calificaciones. Redondea el resultado a un decimal y actualiza el texto del elemento HTML con el ID promedio-img-{idObra} para reflejar el promedio actual de calificaciones sobre la imagen de la obra. Esta función centraliza la lógica de cálculo y actualización del promedio, asegurando que siempre esté sincronizado con los datos guardados.

5. Responsividad y orden por columnas.

Para que las obras se acomoden en columnas utilizamos estos estilos en el id de contenedor-obra:

```
#contenedor-obra {  
    display: grid;  
    grid-template-columns: repeat(4, 1fr); /* 4 columnas en escritorio */
```

```
    gap: 15px;
    margin: 20px;
}
```

Para que se ajuste a diferentes tamaños de dispositivos usamos las siguientes líneas de código:

```
@media (max-width: 1024px) {
    #contenedor-obra {
        grid-template-columns: repeat(3, 1fr); /* 3 columnas en pantallas medianas */
    }
}

@media (max-width: 768px) {
    #contenedor-obra {
        grid-template-columns: repeat(2, 1fr); /* 2 columnas en tablet */
    }
}

@media (max-width: 480px) {
    #contenedor-obra {
        grid-template-columns: 1fr; /* 1 columna en móvil */
    }
}
```

Los cambios se determinan por el ancho de pantalla y tenemos para pantallas medianas con 1024px, para tablets con 768px, para móvil con 480px y para pantalla de computadora grande que viene por defecto.