IIC3585-1
DCC

Tarea 2
Web Assembly

GRUPO 5 - SFEIR - MUÑOZ - VALDÉS

Web Assembly - Importaciones

wasm_bindgen

```rust
use wasm_bindgen::prelude::*;
```

- Exportar funciones desde Rust a JavaScript.
- Importar funciones de JavaScript para usarlas desde Rust.
- **Permite que Rust “hable” con el navegador.**

Web Assembly - Importaciones

image

```
use image::{DynamicImage, ImageFormat, ImageBuffer, Rgba};
```

- **DynamicImage**: Leer cualquier tipo de imagen y convertirla a un formato común para aplicar filtros.
- **ImageFormat:** Especificar cómo exportar la imagen procesada.
- **ImageBuffer:** Acceso rápido y directo a cada píxel para modificarlo.
- **Rgba:** Crear o modificar cada píxel con precisión.

Web Assembly - Importaciones

Cursor

```rust
use std::io::Cursor;
```

- Cargar imágenes directamente desde un arreglo de bytes (como los que vienen del navegador).
- Codificar una imagen procesada para devolverla al navegador.

Web Assembly - Utils

```rust
fn encode_image_to_vec(img: DynamicImage) -> Vec<u8> {
    let mut buffer = Vec::new();
    img.write_to(&mut Cursor::new(&mut buffer), ImageFormat::Png)
        .expect("Failed to encode image");
    buffer
}

fn rgba_to_image(width: u32, height: u32, data: &[u8]) -> DynamicImage {
    let img_buffer = ImageBuffer::<Rgba<u8>, _>::from_raw(width, height, data.to_vec())
        .expect("Failed to create image buffer");
    DynamicImage::ImageRgba8(img_buffer)
}
```

- **encode_image_to_vec**: Convierte una imagen procesada en Rust a un arreglo de bytes PNG, que luego puede ser usado por el navegador
- **rgba_to_image:** Toma una imagen que viene del navegador (como Uint8Array) y la convierte en una estructura que Rust puede procesar fácilmente.

Web Assembly - Filters

Estructura común

```rust
#[wasm_bindgen]
pub fn filter_function(image_data: &[u8], width: u32, height: u32) -> Vec<u8> {
    let img = rgba_to_image(width, height, image_data);
    let filtered = img.filter();
    encode_image_to_vec(filtered)
}
```

1. **Convierte los bytes en una imagen (rgba_to_image).**
2. **Aplica el filtro correspondiente, usando funciones del crate image.**
3. **Convierte la imagen final en Vec<u8> PNG (encode_image_to_vec).**

Funciones PWA

1. Service Worker y Funcionamiento Offline

La aplicación sigue funcionando incluso sin conexión a internet, mostrando los recursos previamente almacenados.

```
self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request).then((response) => {
      return response || fetch(event.request);
    })
  );
});
```

2. Guardado de Imágenes en el Caché (IndexedDB)

Las imágenes procesadas se guardan localmente, lo que permite al usuario acceder a ellas sin necesidad de volver a subirlas nuevamente.

```
export async function saveImage(imageData) {
  const db = await dbPromise;
  await db.add('images', { data: imageData, timestamp: Date.now() });
}

export async function getImages() {
  const db = await dbPromise;
  return await db.getAll('images');
}
```

3. Descarga de Imágenes

Los usuarios pueden guardar las imágenes procesadas en su dispositivo para usarlas fuera de la aplicación.

```
<button
  onClick={() => {
    const link = document.createElement('a');
    link.href = processedImage; // Base64 string
    link.download = 'processed-image.png';
    link.click();
  }}
>
  Download Image
</button>
```

4. Instalación de la App en el Dispositivo

La app se instala en el dispositivo del usuario, funcionando como una app nativa con su propio ícono y sin necesidad de abrir el navegador

```
window.addEventListener('beforeinstallprompt', (e) => {
  e.preventDefault();
  setDeferredPrompt(e); // Guardamos el evento
});

const handleInstallClick = () => {
  if (deferredPrompt) {
    deferredPrompt.prompt(); // Muestra el diálogo de instalación
    deferredPrompt.userChoice.then((choiceResult) => {
      if (choiceResult.outcome === 'accepted') {
        console.log('App instalada');
      }
    });
  }
};
```