CropCut

Tarea 2: Web Assembly y PWAs

GRUPO 7 SECCIÓN 1

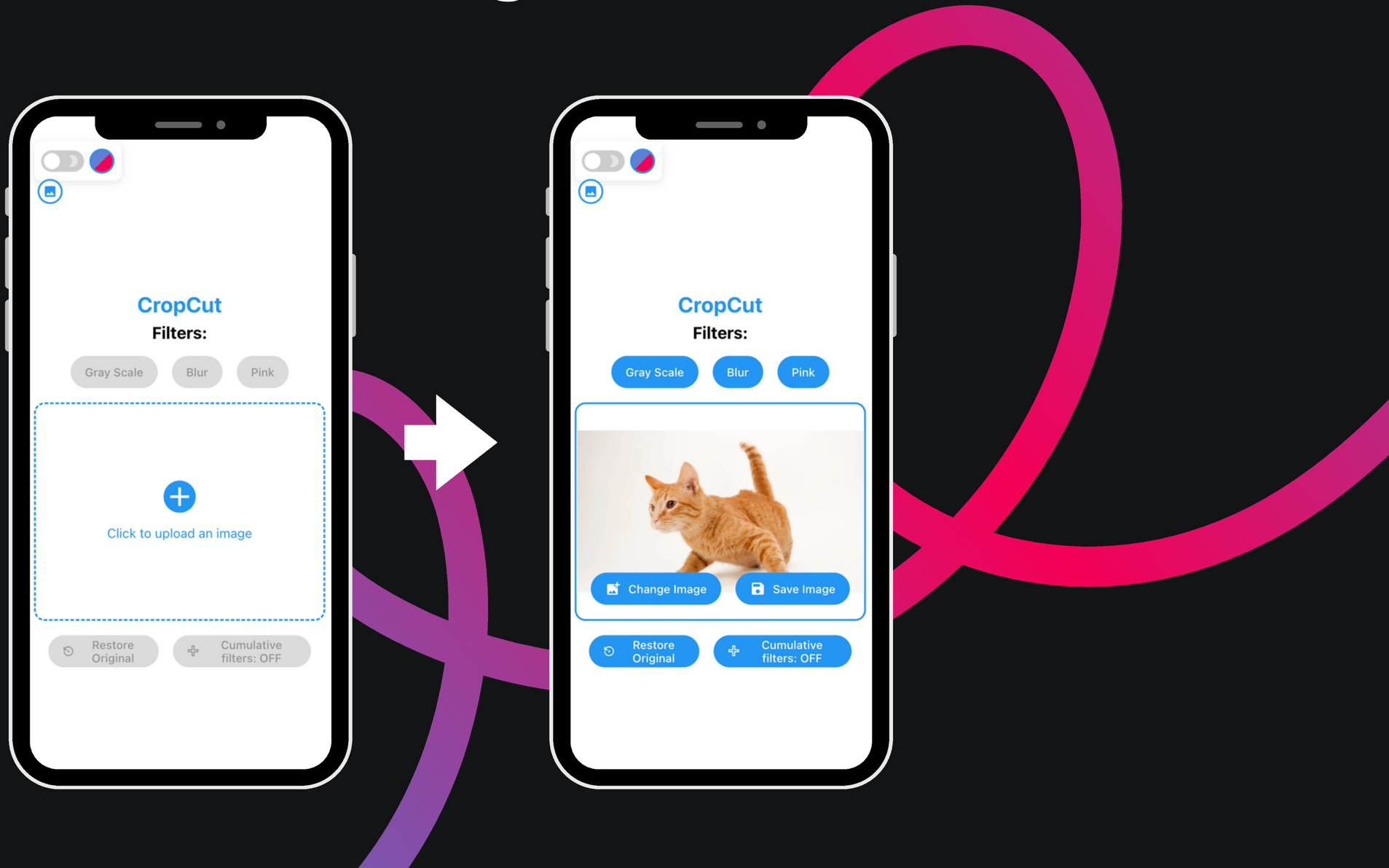
Camila Gervasoni
Juan Pablo Palma
Giacomo Pasquali

Procesamiento de Imágenes: Almacenamiento

```tsx
<ImageContainer>
  <input
    type="file"
    ref={fileInputRef}
    onChange={handleFileChange}
    accept="image/*"
    style={{ display: 'none' }}
  />
  {selectedImage ? (
    <ImagePreview>
      <PreviewImage src={selectedImage} alt="Selected" />
      <ButtonContainer>
        <ActionButton onClick={handleChangeImage}>
          Change Image
        </ActionButton>
        <ActionButton onClick={handleSave}>
          Save Image
        </ActionButton>
      </ButtonContainer>
    </ImagePreview>
  ) : (
    <UploadArea
      onClick={handleClick}
      onDrop={handleDrop}
      onDragOver={handleDragOver}
      onDragLeave={handleDragLeave}
      isDragActive={isDragActive}
    >
      Click to upload an image
    </UploadArea>
  )}
</ImageContainer>
```

```tsx
const [selectedImage, setSelectedImage] = useState<string | null>(null);
const [originalImage, setOriginalImage] = useState<string | null>(null);
```

Procesamiento de Imágenes: Almacenamiento



Procesamiento de Imágenes: Aplicar filtros

Desde la PWA:

1. Importar funciones de WASM:

```
1   import init, { grayscale, blur, pink_filter } from '../../wasm-lib/pkg/wasm_lib';
```

Procesamiento de Imágenes: Aplicar filtros

Desde la PWA:

2. Llamar funciones

```ts
const processImage = async (file: File, filter: string) => {
  const arrayBuffer = await file.arrayBuffer();
  const uint8Array = new Uint8Array(arrayBuffer);
  await init();
  let result;
  switch (filter) {
    case 'grayscale':
    result = grayscale(uint8Array);
    break;
    case 'blur':
    result = blur(uint8Array, 5);
    break;
    case 'pink':
    result = pink_filter(uint8Array);
    break;
    default:
    throw new Error(`Unsupported filter: ${filter}`);
  }
  const blob = new Blob([result], { type: "image/png" });
  const url = URL.createObjectURL(blob);
  return url;
};
```

Procesamiento de Imágenes: Aplicar filtros

Desde WASM:

fn grayscale(image_data)

```rust
#[wasm_bindgen]
pub fn grayscale(image_data: &[u8]) -> Vec<u8> {
    let img = image::load_from_memory(image_data).expect("Failed to load image");
    let gray_img = img.grayscale();
    let mut buf = Cursor::new(Vec::new());
    gray_img
        .write_to(&mut buf, image::ImageFormat::Png)
        .expect("Failed to write image");
    buf.into_inner()
}
```

fn blur(image_data, radius)

```rust
#[wasm_bindgen]
pub fn blur(image_data: &[u8], radius: f32) -> Vec<u8> {
    let img = image::load_from_memory(image_data).expect("Failed to load image");
    let blurred_img = img.blur(radius);
    let mut buf = Cursor::new(Vec::new());
    blurred_img
        .write_to(&mut buf, image::ImageFormat::Png)
        .expect("Failed to write image");
    buf.into_inner()
}
```

Se utilizan las funciones **grayscale** y **blur**, funciones previamente definidas de la libreria **image**.

Procesamiento de Imágenes: Aplicar filtros

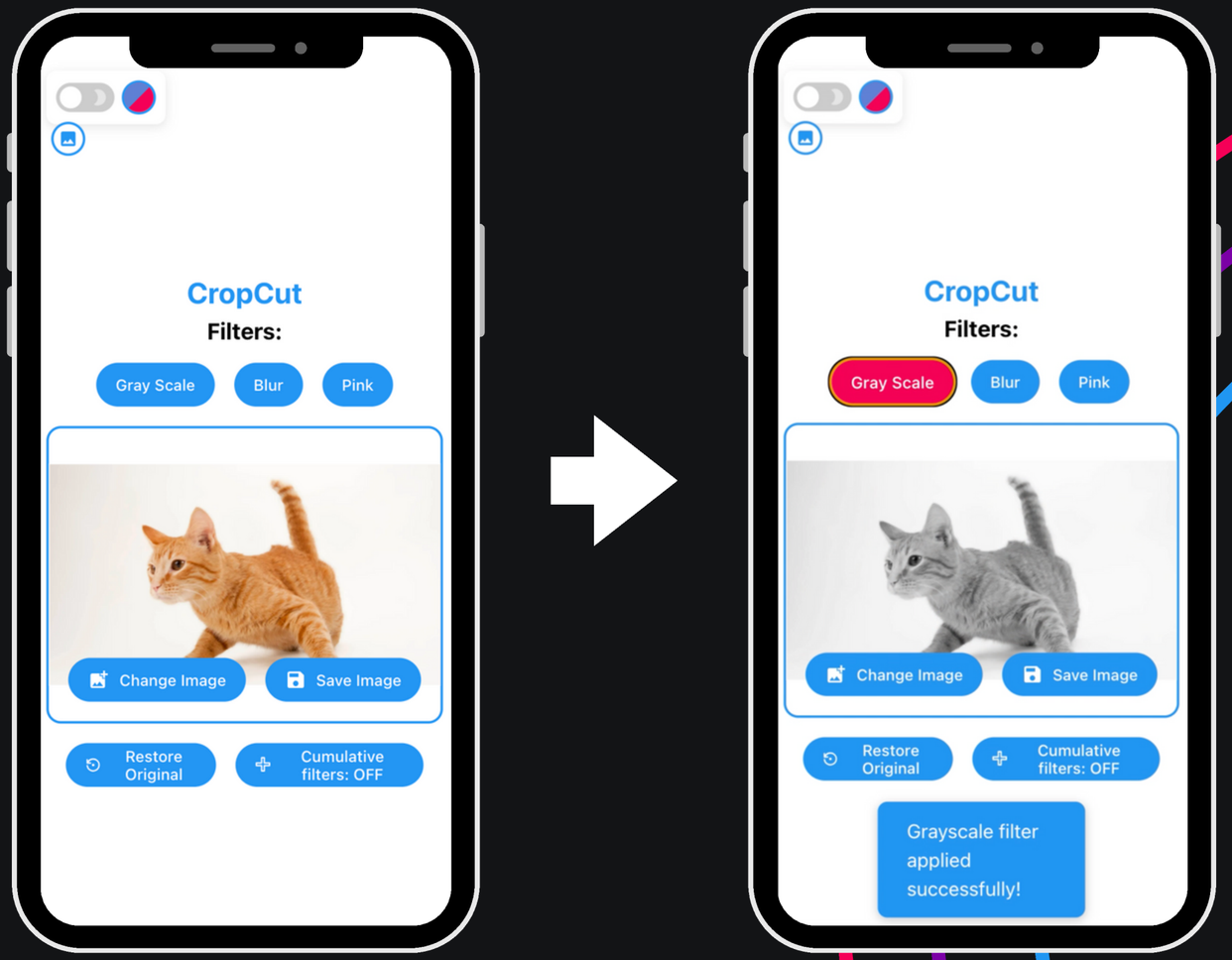
Desde WASM:

fn pink_filter(image_data)

```rust
#[wasm_bindgen]
pub fn pink_filter(image_data: &[u8]) -> Vec<u8> {
    let img = image::load_from_memory(image_data).expect("Failed to load image");
    let (width, height) = img.dimensions();
    let mut pink_img: ImageBuffer<Rgba<u8>, Vec<u8>> = ImageBuffer::new(width, height);
    for (x, y, pixel) in img.pixels() {
        let mut pink_pixel = pixel.to_rgba();
        pink_pixel[0] = (pink_pixel[0] as f32 * 1.2).min(255.0) as u8;
        pink_pixel[1] = (pink_pixel[1] as f32 * 0.8).min(255.0) as u8;
        pink_pixel[2] = (pink_pixel[2] as f32 * 1.1).min(255.0) as u8;
        pink_img.put_pixel(x, y, pink_pixel);
    }
    let mut buf = Cursor::new(Vec::new());
    pink_img
        .write_to(&mut buf, image::ImageFormat::Png)
        .expect("Failed to write image");
    buf.into_inner()
}
```

Se manipulan los colores RGB de cada pixel de la imagen con funciones de la librería **image**

Procesamiento de Imágenes: Aplicar filtros



Uso de IndexedDB: En código

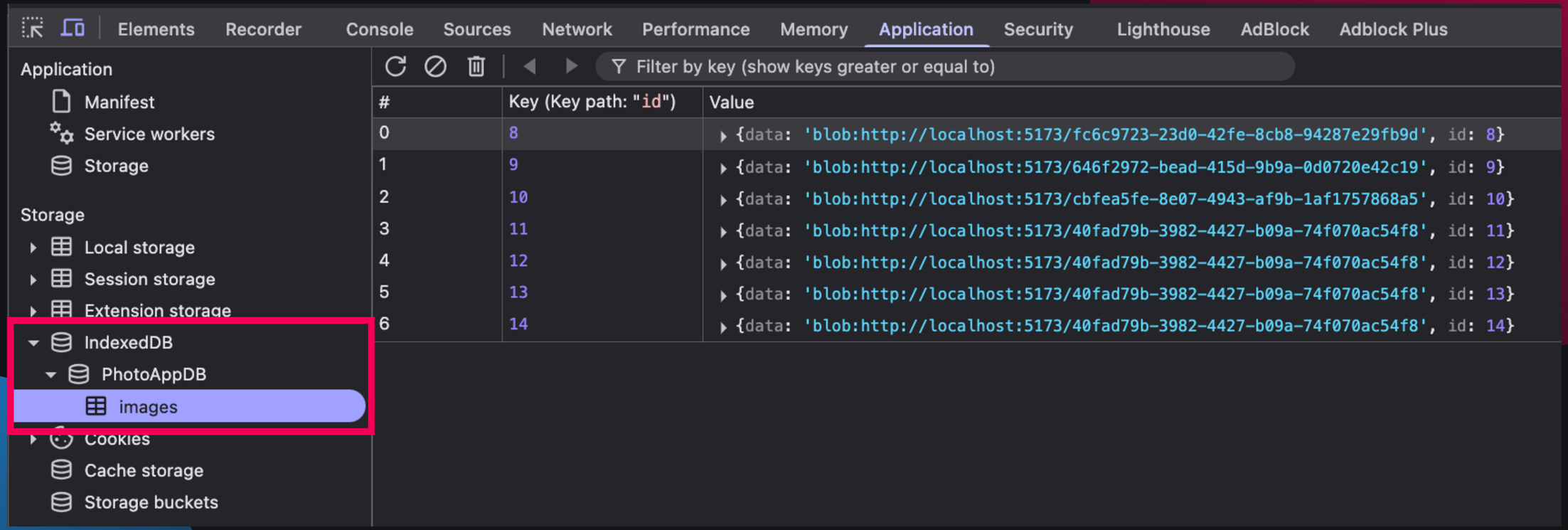
```ts
export const saveImageToDB = async (imageData: string): Promise<void> => {
  const db = await openDatabase();
  const transaction = db.transaction(['images'], 'readwrite');
  const store = transaction.objectStore('images');
  return new Promise((resolve, reject) => {
    const request = store.add({ data: imageData });
    request.onsuccess = () => resolve();
    request.onerror = () => reject(request.error);
  });
};
const openDatabase = (): Promise<IDBDatabase> => {
  return new Promise((resolve, reject) => {
    const request = indexedDB.open('PhotoAppDB', 1);
    request.onerror = () => reject(request.error);
    request.onsuccess = () => resolve(request.result);
    request.onupgradeneeded = (event) => {
      const db = (event.target as IDBOpenDBRequest).result;
      if (!db.objectStoreNames.contains('images')) {
        db.createObjectStore('images', { keyPath: 'id', autoIncrement: true });
      }
    };
  });
};
```

Guardar imagen en almacen **images** en DB

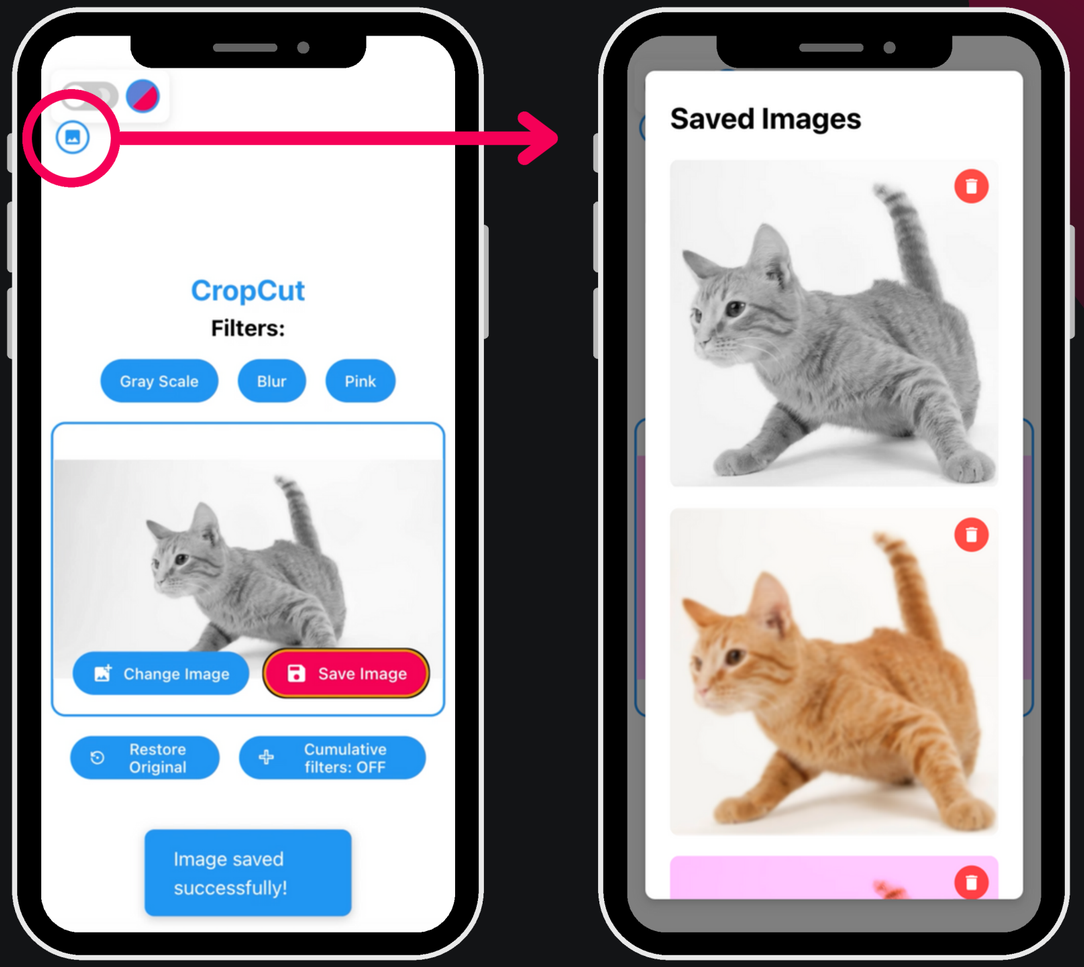
Abrir **PhotoAppDB**

Crear almacen **images** en DB

Uso de IndexedDB: En la práctica



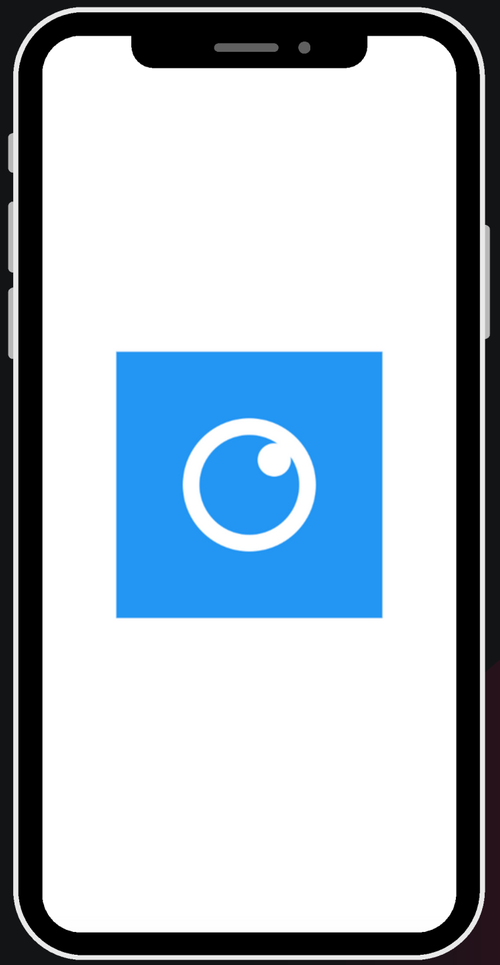
Uso de IndexedDB: En la PWA



Personalización App: Splash screen e Ícono

```tsx
import React, { useEffect, useState } from 'react';

export const SplashScreen: React.FC = () => {
  const [hidden, setHidden] = useState(false);
  useEffect(() => {
    const timer = setTimeout(() => {
      setHidden(true);
    }, 2000);
    return () => clearTimeout(timer);
  }, []);
  if (hidden) return null;
  return (
    <div className={`app-splash-screen ${hidden ? 'hidden' : ''}`}>
      <img
        src="/pwa-512x512.png"
        alt="App Logo"
        style={{
          width: '200px',
          height: '200px',
          animation: 'pulse 2s infinite'
        }}
      />
      <style>
        {`
          @keyframes pulse {
            0% { transform: scale(1); }
            50% { transform: scale(1.1); }
            100% { transform: scale(1); }
          }
        `}
      </style>
    </div>
  );
};
```

Se muestra por 2 segundos

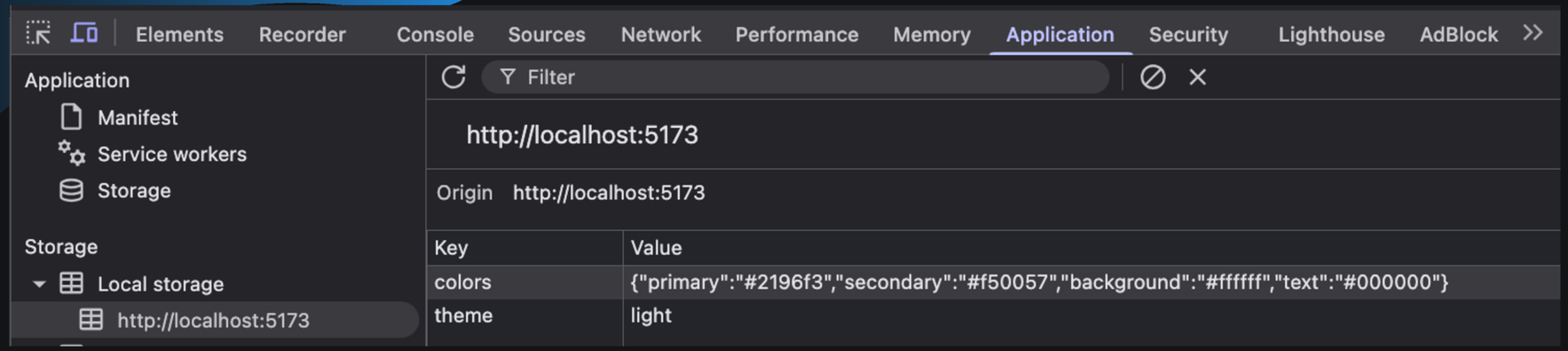


Personalización App: Tema y colores

```tsx
type Theme = 'light' | 'dark' | 'custom';
type ColorScheme = {
  primary: string;
  secondary: string;
  background: string;
  text: string;
};

interface ThemeContextType {
  theme: Theme;
  colors: ColorScheme;
  setTheme: (theme: Theme) => void;
  setColors: (colors: ColorScheme) => void;
}

const lightColors: ColorScheme = {
  primary: '#2196f3',
  secondary: '#f50057',
  background: '#ffffff',
  text: '#000000',
};

const darkColors: ColorScheme = {
  primary: '#64b5f6',
  secondary: '#ff4081',
  background: '#121212',
  text: '#ffffff',
};
```

```tsx
export const ThemeProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const [theme, setThemeState] = useState<Theme>('light');
  const [colors, setColors] = useState<ColorScheme>(lightColors);
  useEffect(() => {
    // Load saved theme and colors from localStorage
    const savedTheme = localStorage.getItem('theme') as Theme;
    const savedColors = localStorage.getItem('colors');
    if (savedTheme) setThemeState(savedTheme);
    if (savedColors) setColors(JSON.parse(savedColors));
  }, []);
  const setTheme = (newTheme: Theme) => {
    setThemeState(newTheme);
    // Update colors based on theme
    if (newTheme === 'light') {
      setColors(lightColors);
    } else if (newTheme === 'dark') {
      setColors(darkColors);
    }
    // For custom theme, keep existing colors
  };
  useEffect(() => {
    // Save theme and colors to localStorage
    localStorage.setItem('theme', theme);
    localStorage.setItem('colors', JSON.stringify(colors));
    // Apply theme to document
    document.documentElement.setAttribute('data-theme', theme);
    // Apply colors to CSS variables
    Object.entries(colors).forEach(([key, value]) => {
      document.documentElement.style.setProperty(`--${key}-color`, value);
    });
    // Apply background and text colors to body
    document.body.style.backgroundColor = colors.background;
    document.body.style.color = colors.text;
  }, [theme, colors]);
  return (
    <ThemeContext.Provider value={{ theme, colors, setTheme, setColors }}>
      {children}
    </ThemeContext.Provider>
  );
};
```

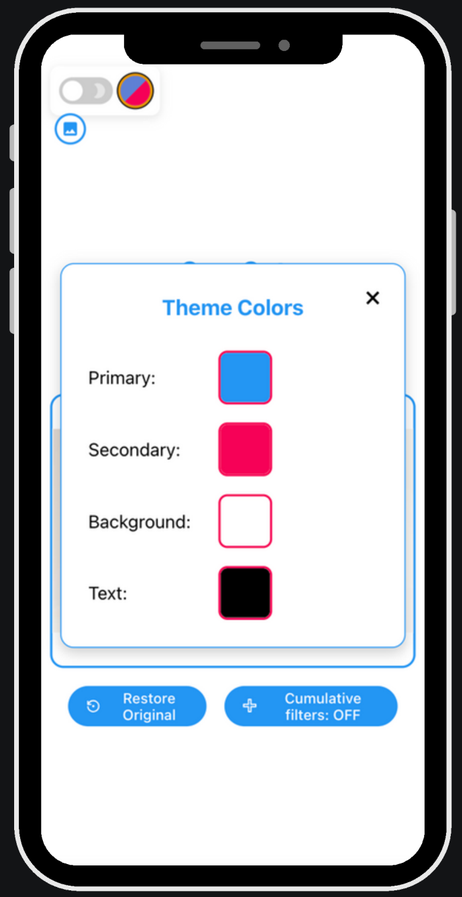
Personalización App: Tema y colores

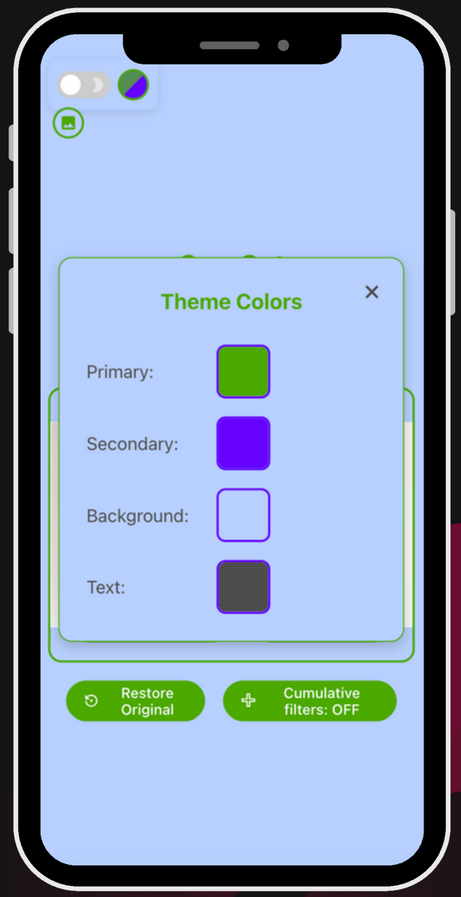


Personalización App: Tema y colores

Light

Dark



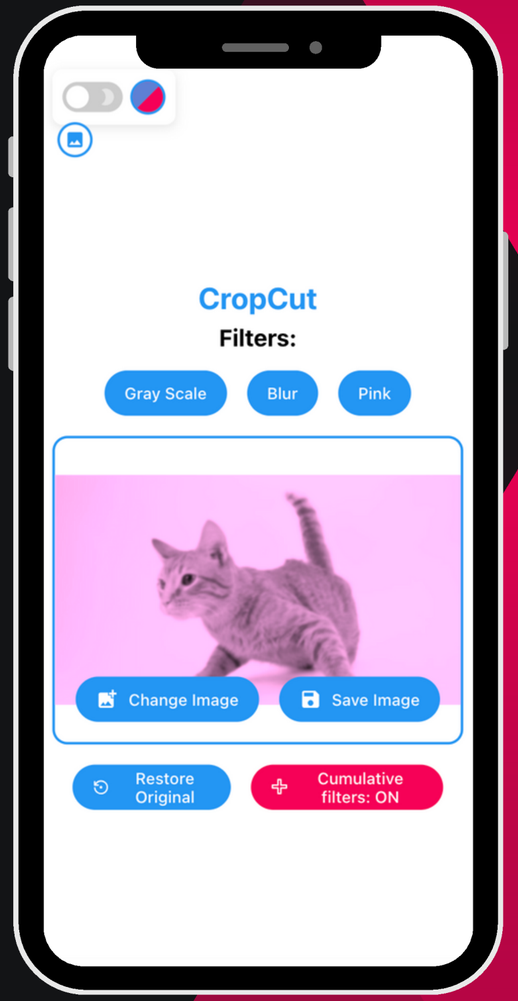


Custom

Extra: Filtros Acumulativos

```
1   const [cumulativeFilters, setCumulativeFilters] = useState<boolean>(false);
```

```
1   const sourceImage = cumulativeFilters ? selectedImage : originalImage;
```



CropCut

¡Gracias por su atención!