

CropCut

# Tarea 2: Web Assembly y PWAs

GRUPO 7 SECCIÓN 1

Camila Gervasoni  
Juan Pablo Palma  
Giacomo Pasquali

---



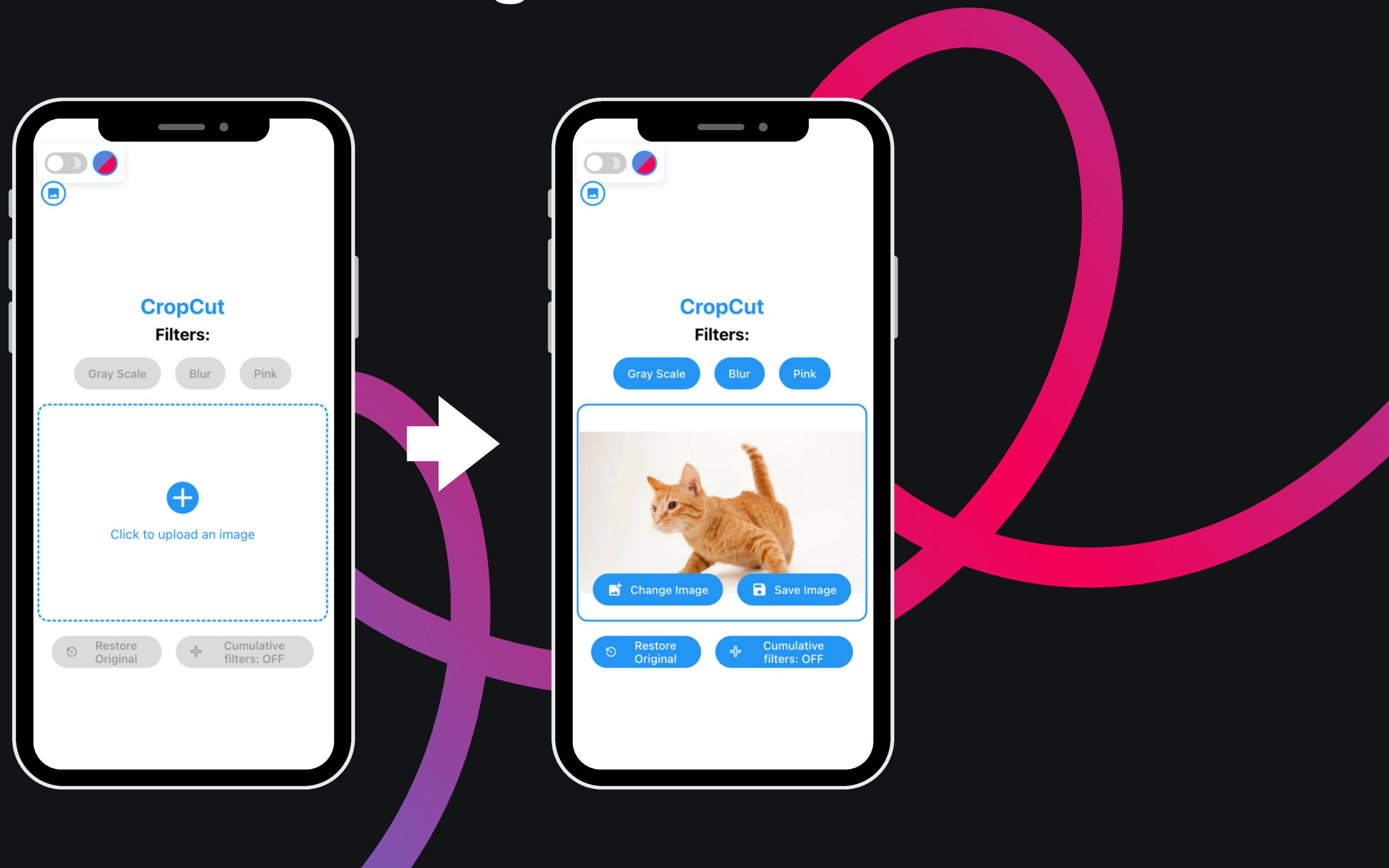
# Procesamiento de Imágenes: Almacenamiento

```
● ○ ●  
1 <ImageContainer>  
2   <input  
3     type="file"  
4     ref={fileInputRef}  
5     onChange={handleFileChange}  
6     accept="image/*"  
7     style={{ display: 'none' }}  
8   />  
9   {selectedImage ? (  
10     <ImagePreview>  
11       <PreviewImage src={selectedImage} alt="Selected" />  
12     <ButtonContainer>  
13       <ActionButton onClick={handleChangeImage}>  
14         Change Image  
15       </ActionButton>  
16       <ActionButton onClick={handleSave}>  
17         Save Image  
18       </ActionButton>  
19     </ButtonContainer>  
20   </ImagePreview>  
21 ) : (  
22   <UploadArea  
23     onClick={handleClick}  
24     onDrop={handleDrop}  
25     onDragOver={handleDragOver}  
26     onDragLeave={handleDragLeave}  
27     isDragActive={isDragActive}  
28   >  
29     Click to upload an image  
30   </UploadArea>  
31 )  
32 </ImageContainer>
```



```
1 const [selectedImage, setSelectedImage] = useState<string | null>(null);  
2 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

```
● ● ●  
1  const processImage = async (file: File, filter: string) => {  
2    const arrayBuffer = await file.arrayBuffer();  
3    const uint8Array = new Uint8Array(arrayBuffer);  
4    await init();  
5    let result;  
6    switch (filter) {  
7      case 'grayscale':  
8        result = grayscale(uint8Array);  
9        break;  
10     case 'blur':  
11       result = blur(uint8Array, 5);  
12       break;  
13     case 'pink':  
14       result = pink_filter(uint8Array);  
15       break;  
16     default:  
17       throw new Error(`Unsupported filter: ${filter}`);  
18   }  
19   const blob = new Blob([result], { type: "image/png" });  
20   const url = URL.createObjectURL(blob);  
21   return url;  
22 };
```

# Procesamiento de Imágenes: Aplicar filtros

Desde WASM:

fn grayscale(image\_data)

```
1 #[wasm_bindgen]
2 pub fn grayscale(image_data: &[u8]) -> Vec<u8> {
3     let img = image::load_from_memory(image_data).expect("Failed to load image");
4     let gray_img = img.grayscale();
5     let mut buf = Cursor::new(Vec::new());
6     gray_img
7         .write_to(&mut buf, image::ImageFormat::Png)
8         .expect("Failed to write image");
9     buf.into_inner()
10 }
```

fn blur(image\_data, radius)

```
1 #[wasm_bindgen]
2 pub fn blur(image_data: &[u8], radius: f32) -> Vec<u8> {
3     let img = image::load_from_memory(image_data).expect("Failed to load image");
4     let blurred_img = img.blur(radius);
5     let mut buf = Cursor::new(Vec::new());
6     blurred_img
7         .write_to(&mut buf, image::ImageFormat::Png)
8         .expect("Failed to write image");
9     buf.into_inner()
10 }
```

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



# Procesamiento de Imágenes: Aplicar filtros

Desde WASM:

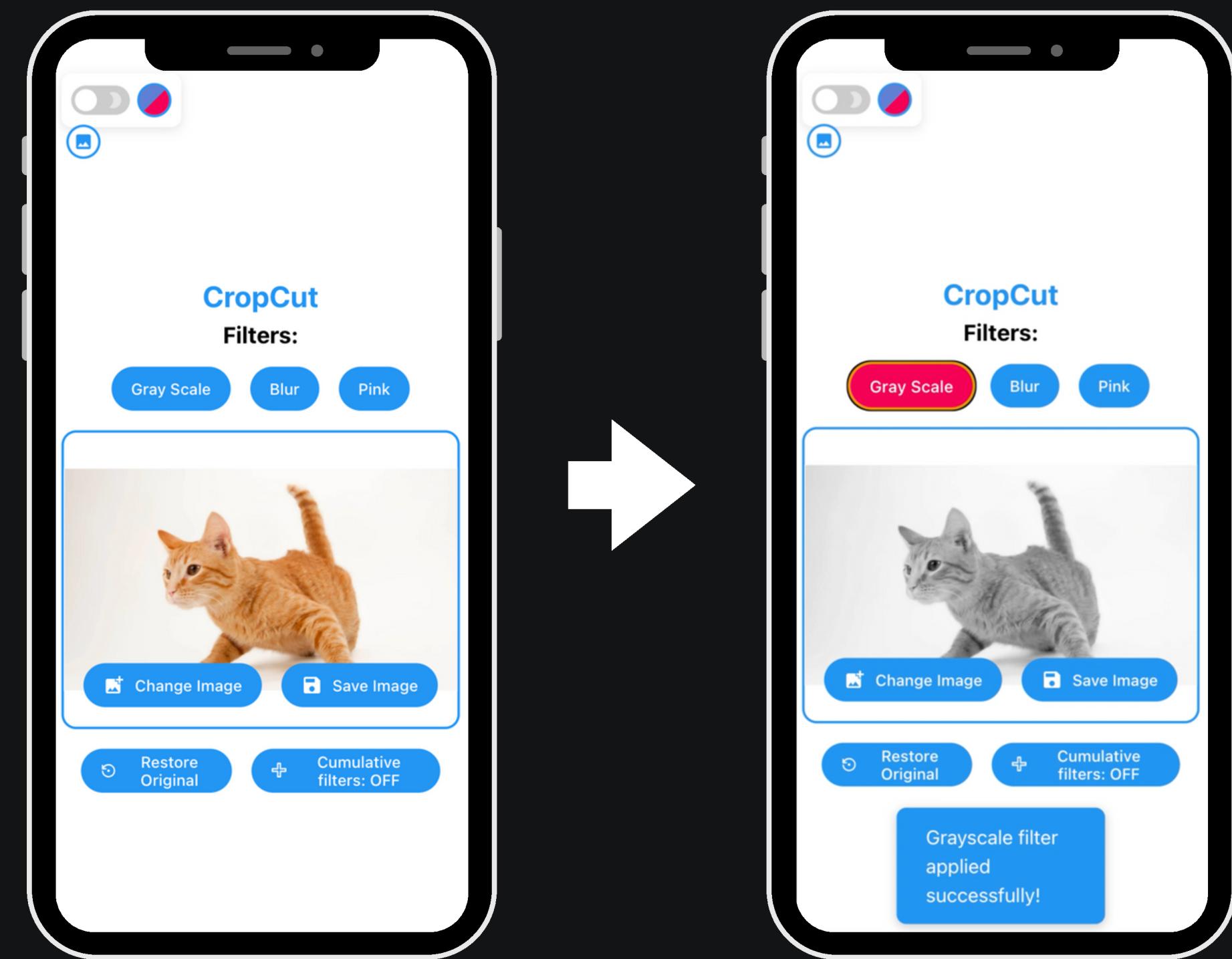
fn pink\_filter(image\_data)

```
● ● ●  
1 #[wasm_bindgen]  
2 pub fn pink_filter(image_data: &[u8]) -> Vec<u8> {  
3     let img = image::load_from_memory(image_data).expect("Failed to load image");  
4     let (width, height) = img.dimensions();  
5     let mut pink_img: ImageBuffer<Rgba<u8>, Vec<u8>> = ImageBuffer::new(width, height);  
6     for (x, y, pixel) in img.pixels() {  
7         let mut pink_pixel = pixel.to_rgba();  
8         pink_pixel[0] = (pink_pixel[0] as f32 * 1.2).min(255.0) as u8;  
9         pink_pixel[1] = (pink_pixel[1] as f32 * 0.8).min(255.0) as u8;  
10        pink_pixel[2] = (pink_pixel[2] as f32 * 1.1).min(255.0) as u8;  
11        pink_img.put_pixel(x, y, pink_pixel);  
12    }  
13    let mut buf = Cursor::new(Vec::new());  
14    pink_img  
15        .write_to(&mut buf, image::ImageFormat::Png)  
16        .expect("Failed to write image");  
17    buf.into_inner()  
18 }
```

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



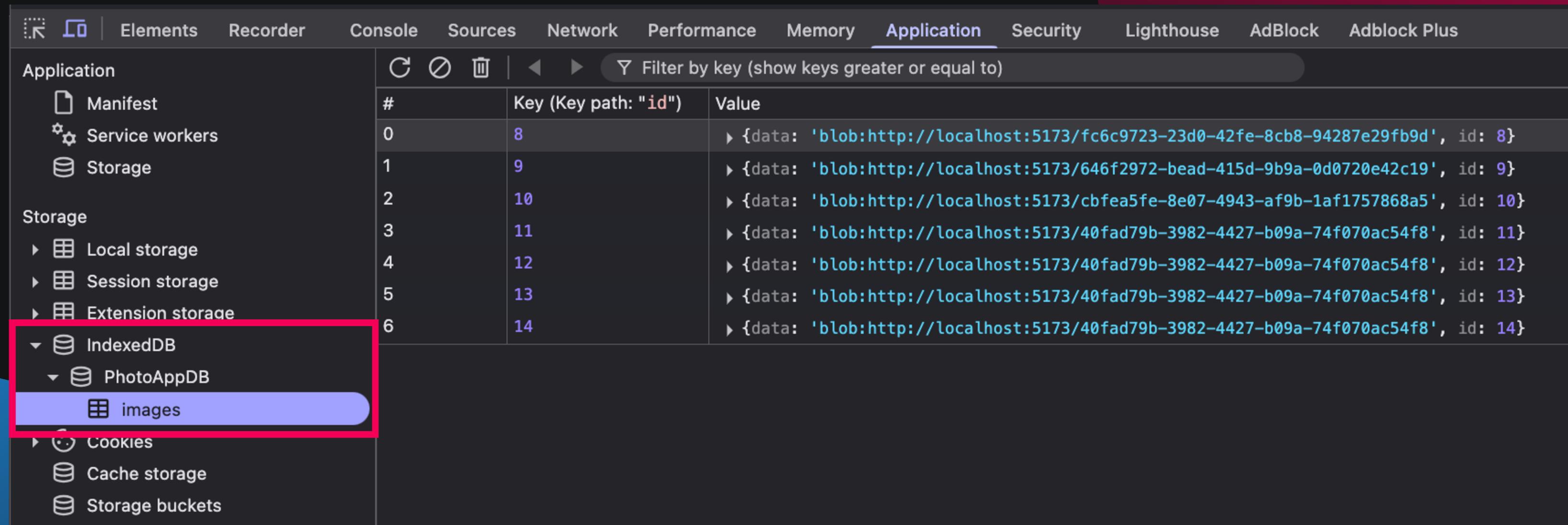
```
1  export const saveImageToDB = async (imageData: string): Promise<void> => {
2    const db = await openDatabase();
3    const transaction = db.transaction(['images'], 'readwrite');
4    const store = transaction.objectStore('images');
5    return new Promise((resolve, reject) => {
6      const request = store.add({ data: imageData });
7      request.onsuccess = () => resolve();
8      request.onerror = () => reject(request.error);
9    });
10 };
11 const openDatabase = (): Promise<IDBDatabase> => {
12   return new Promise((resolve, reject) => {
13     const request = indexedDB.open('PhotoAppDB', 1);
14     request.onerror = () => reject(request.error);
15     request.onsuccess = () => resolve(request.result);
16     request.onupgradeneeded = (event) => {
17       const db = (event.target as IDBOpenDBRequest).result;
18       if (!db.objectStoreNames.contains('images')) {
19         db.createObjectStore('images', { keyPath: 'id', autoIncrement: true });
20       }
21     };
22   });
23 }
```

Guardar imagen en  
almacen **images** en DB

Abrir **PhotoAppDB**

Crear almacen **images** en DB

# Uso de IndexedDB: En la práctica

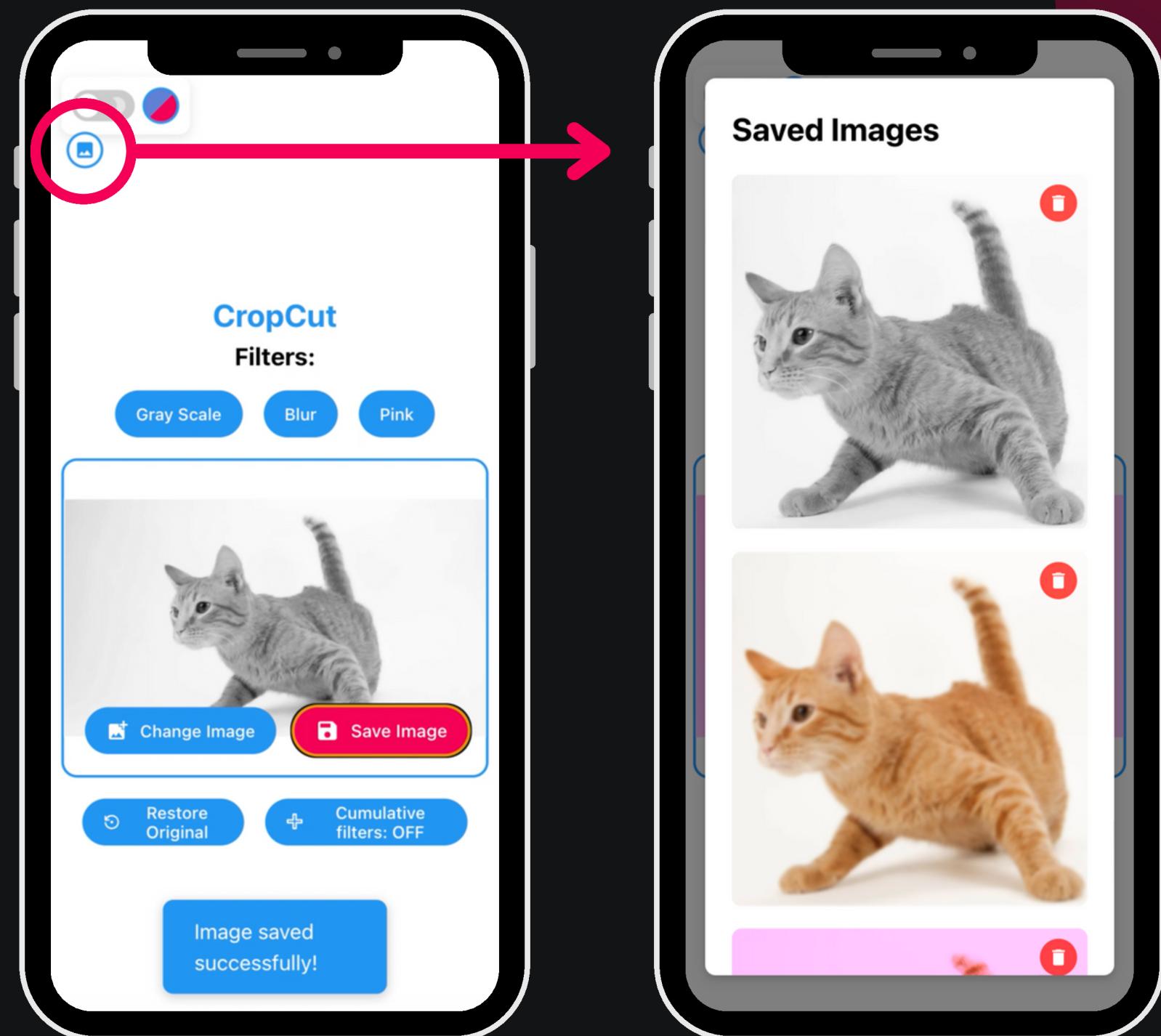


The screenshot shows the Chrome DevTools Application tab interface. On the left, there's a sidebar with various storage-related items: Manifest, Service workers, Storage, Local storage, Session storage, Extension storage, IndexedDB (which is expanded to show PhotoAppDB and images), Cookies, Cache storage, and Storage buckets. A red box highlights the IndexedDB section. The main area displays a table with columns for #, Key (Key path: "id"), and Value. The table contains 15 rows, indexed from 0 to 14. Each row shows a key and its corresponding value, which is a blob URL pointing to a local host. The table also includes controls for clearing data and filtering by key.

#	Key (Key path: "id")	Value
0	8	▶ {data: 'blob:http://localhost:5173/fc6c9723-23d0-42fe-8cb8-94287e29fb9d', id: 8}
1	9	▶ {data: 'blob:http://localhost:5173/646f2972-bead-415d-9b9a-0d0720e42c19', id: 9}
2	10	▶ {data: 'blob:http://localhost:5173/cbfea5fe-8e07-4943-af9b-1af1757868a5', id: 10}
3	11	▶ {data: 'blob:http://localhost:5173/40fad79b-3982-4427-b09a-74f070ac54f8', id: 11}
4	12	▶ {data: 'blob:http://localhost:5173/40fad79b-3982-4427-b09a-74f070ac54f8', id: 12}
5	13	▶ {data: 'blob:http://localhost:5173/40fad79b-3982-4427-b09a-74f070ac54f8', id: 13}
6	14	▶ {data: 'blob:http://localhost:5173/40fad79b-3982-4427-b09a-74f070ac54f8', id: 14}
7		
8		
9		
10		
11		
12		
13		
14		
15		



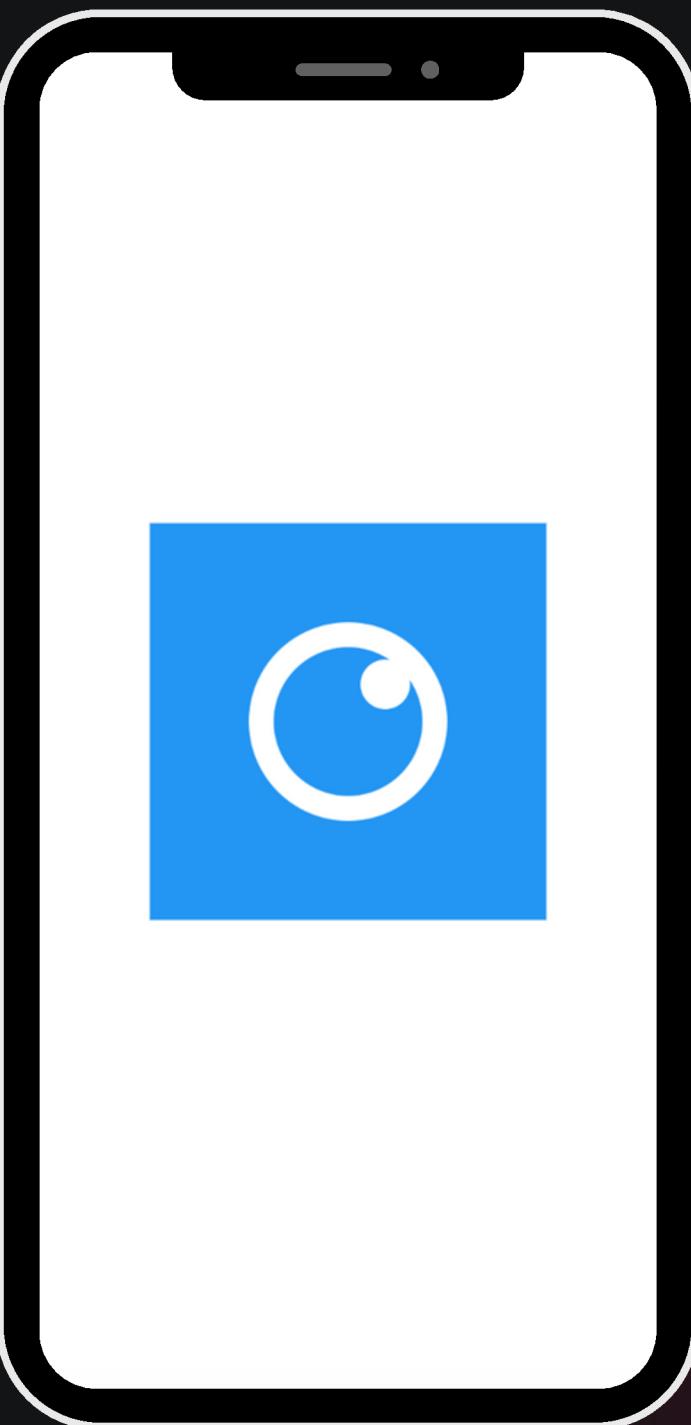
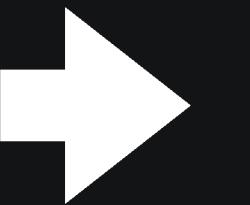
# Uso de IndexedDB: En la PWA





# Personalización App: Splash screen e Ícono

```
1 import React, { useEffect, useState } from 'react';
2
3 export const SplashScreen: React.FC = () => {
4   const [hidden, setHidden] = useState(false);
5   useEffect(() => {
6     const timer = setTimeout(() => {
7       setHidden(true);
8     }, 2000); → Se muestra por 2 segundos
9     return () => clearTimeout(timer);
10  }, []);
11  if (hidden) return null;
12  return (
13    <div className={`app-splash-screen ${hidden ? 'hidden' : ''}`}>
14      
23      <style>
24        `<br>
25        @keyframes pulse {
26          0% { transform: scale(1); }
27          50% { transform: scale(1.1); }
28          100% { transform: scale(1); }
29        }
30      `</style>
31    </div>
32  );
33}
34;
```





# Personalización App: Tema y colores



```
1 type Theme = 'light' | 'dark' | 'custom';
2 type ColorScheme = {
3   primary: string;
4   secondary: string;
5   background: string;
6   text: string;
7 };
8
9 interface ThemeContextType {
10   theme: Theme;
11   colors: ColorScheme;
12   setTheme: (theme: Theme) => void;
13   setColors: (colors: ColorScheme) => void;
14 }
15
16 const lightColors: ColorScheme = {
17   primary: '#2196f3',
18   secondary: '#f50057',
19   background: '#ffffff',
20   text: '#000000',
21 };
22
23 const darkColors: ColorScheme = {
24   primary: '#64b5f6',
25   secondary: '#ff4081',
26   background: '#121212',
27   text: '#ffffff',
28 };
```



```
1 export const ThemeProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
2   const [theme, setThemeState] = useState<Theme>('light');
3   const [colors, setColors] = useState<ColorScheme>(lightColors);
4   useEffect(() => {
5     // Load saved theme and colors from localStorage
6     const savedTheme = localStorage.getItem('theme') as Theme;
7     const savedColors = localStorage.getItem('colors');
8     if (savedTheme) setThemeState(savedTheme);
9     if (savedColors) setColors(JSON.parse(savedColors));
10 }, []);
11 const setTheme = (newTheme: Theme) => {
12   setThemeState(newTheme);
13   // Update colors based on theme
14   if (newTheme === 'light') {
15     setColors(lightColors);
16   } else if (newTheme === 'dark') {
17     setColors(darkColors);
18   }
19   // For custom theme, keep existing colors
20 };
21 useEffect(() => {
22   // Save theme and colors to localStorage
23   localStorage.setItem('theme', theme);
24   localStorage.setItem('colors', JSON.stringify(colors));
25   // Apply theme to document
26   document.documentElement.setAttribute('data-theme', theme);
27   // Apply colors to CSS variables
28   Object.entries(colors).forEach(([key, value]) => {
29     document.documentElement.style.setProperty(`--${key}-color`, value);
30   });
31   // Apply background and text colors to body
32   document.body.style.backgroundColor = colors.background;
33   document.body.style.color = colors.text;
34 }, [theme, colors]);
35 return (
36   <ThemeContext.Provider value={{ theme, colors, setTheme, setColors }}>
37     {children}
38   </ThemeContext.Provider>
39 );
40 };
```



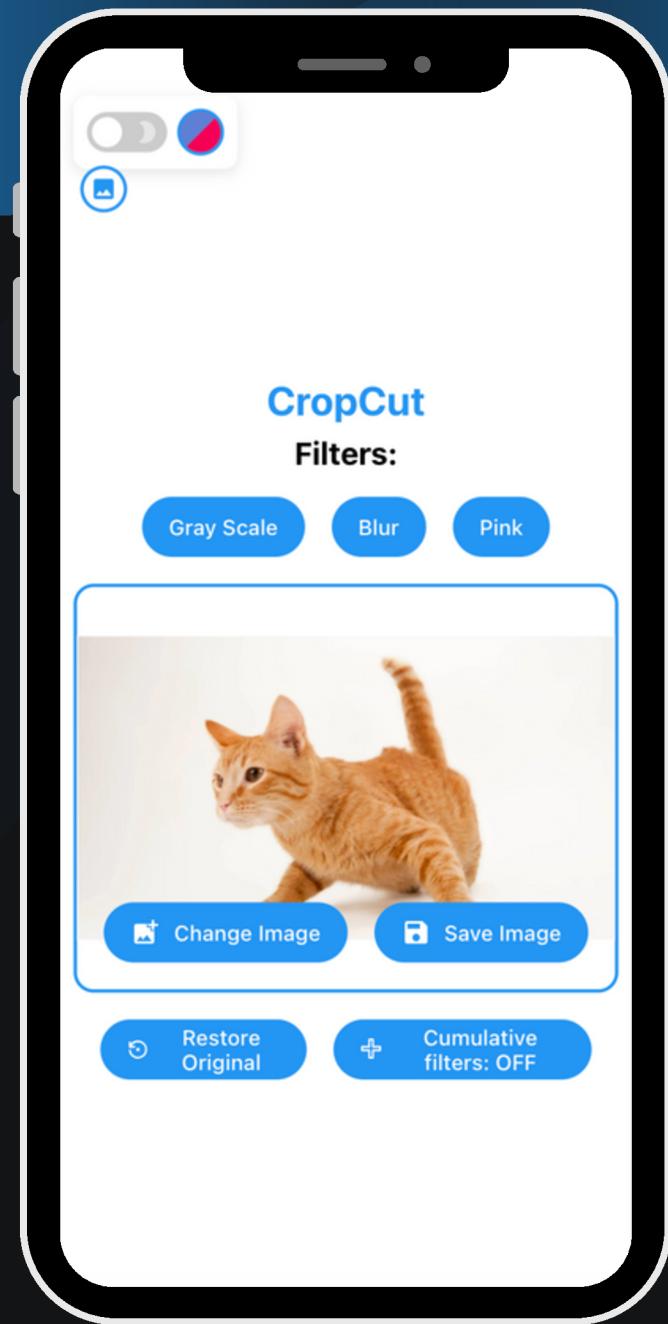
# Personalización App: Tema y colores

The screenshot shows the Chrome DevTools Application tab interface. On the left, there's a sidebar with sections for Application (Manifest, Service workers, Storage), Network (Local storage), and a table for the current origin (http://localhost:5173). The main area displays the contents of the Local storage.

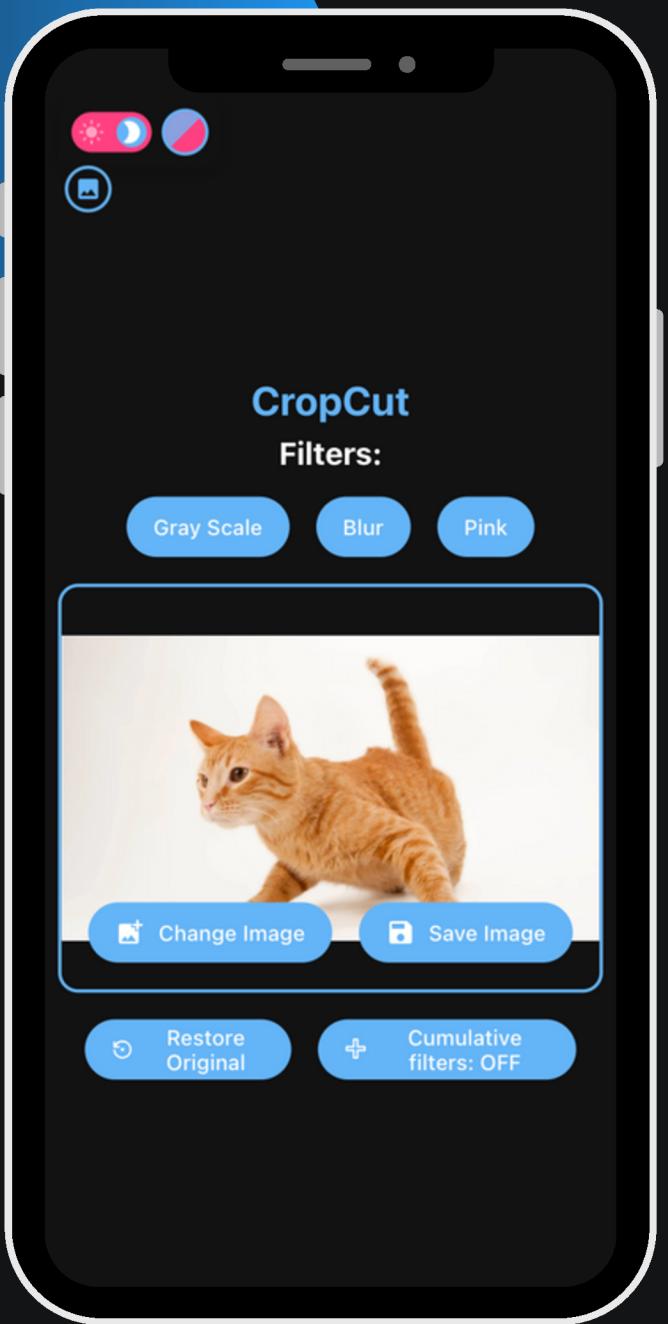
Key	Value
colors	{"primary": "#2196f3", "secondary": "#f50057", "background": "#ffffff", "text": "#000000"}
theme	light



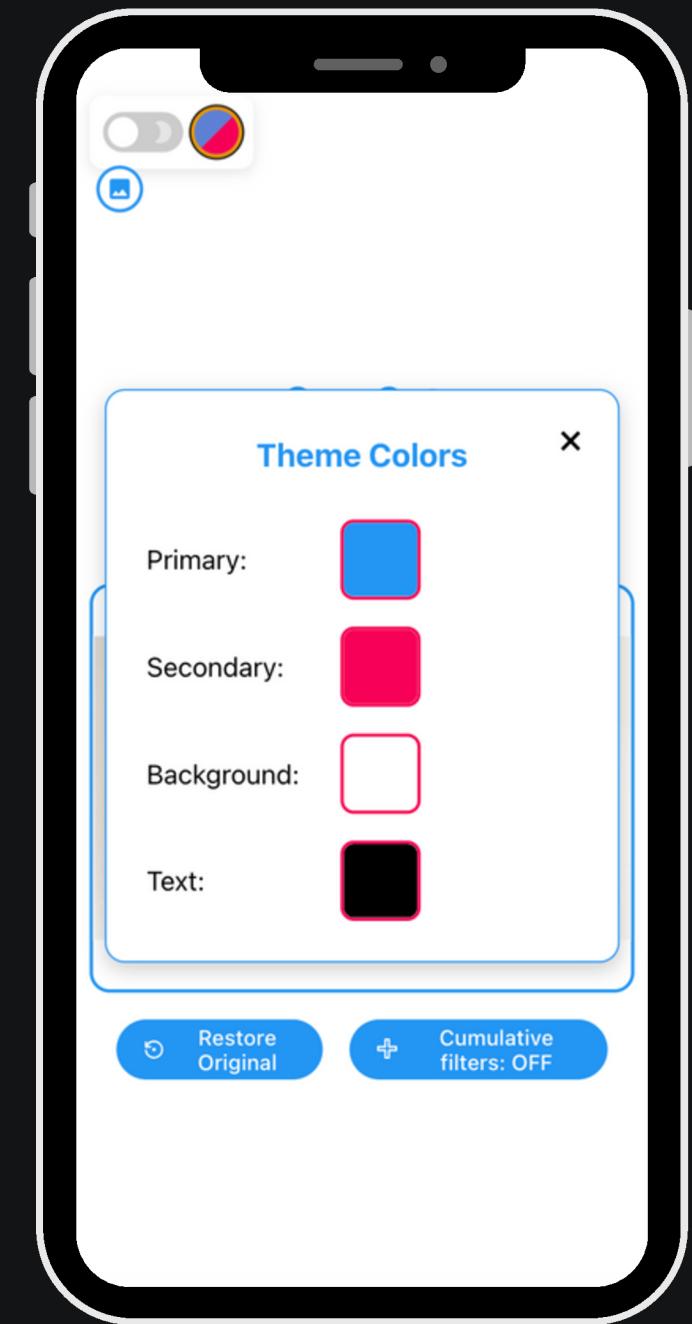
# Personalización App: Tema y colores



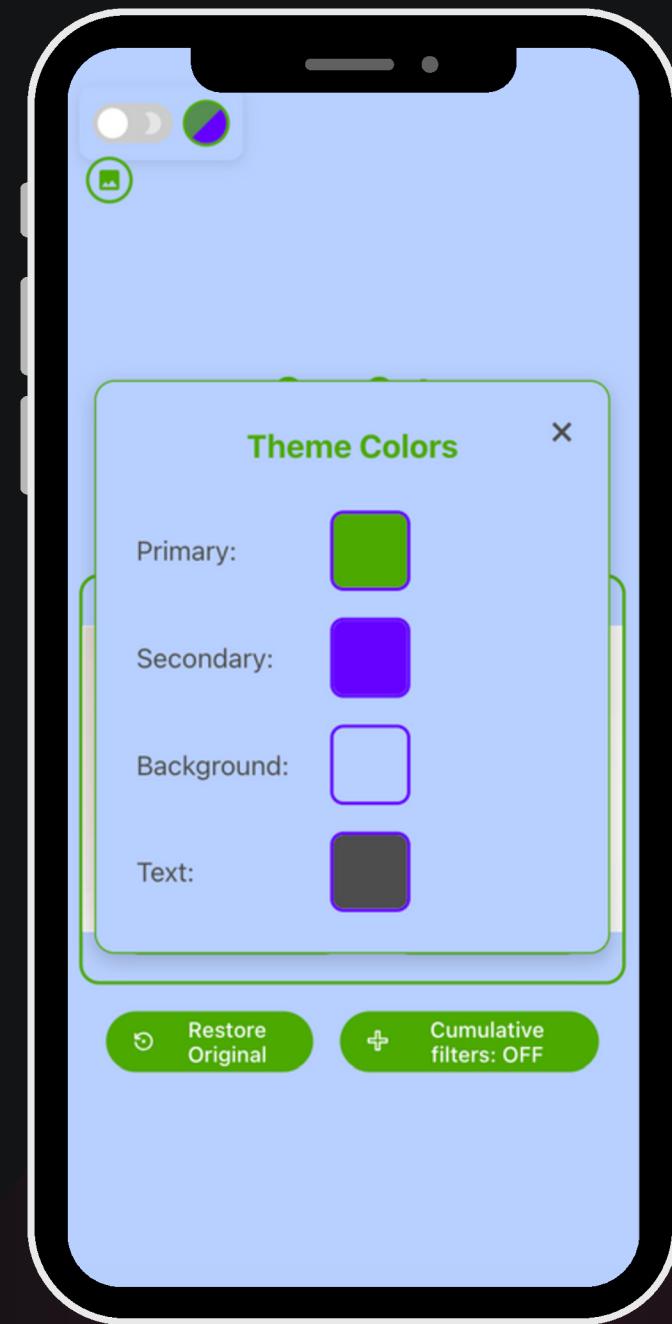
Light



Dark



Custom

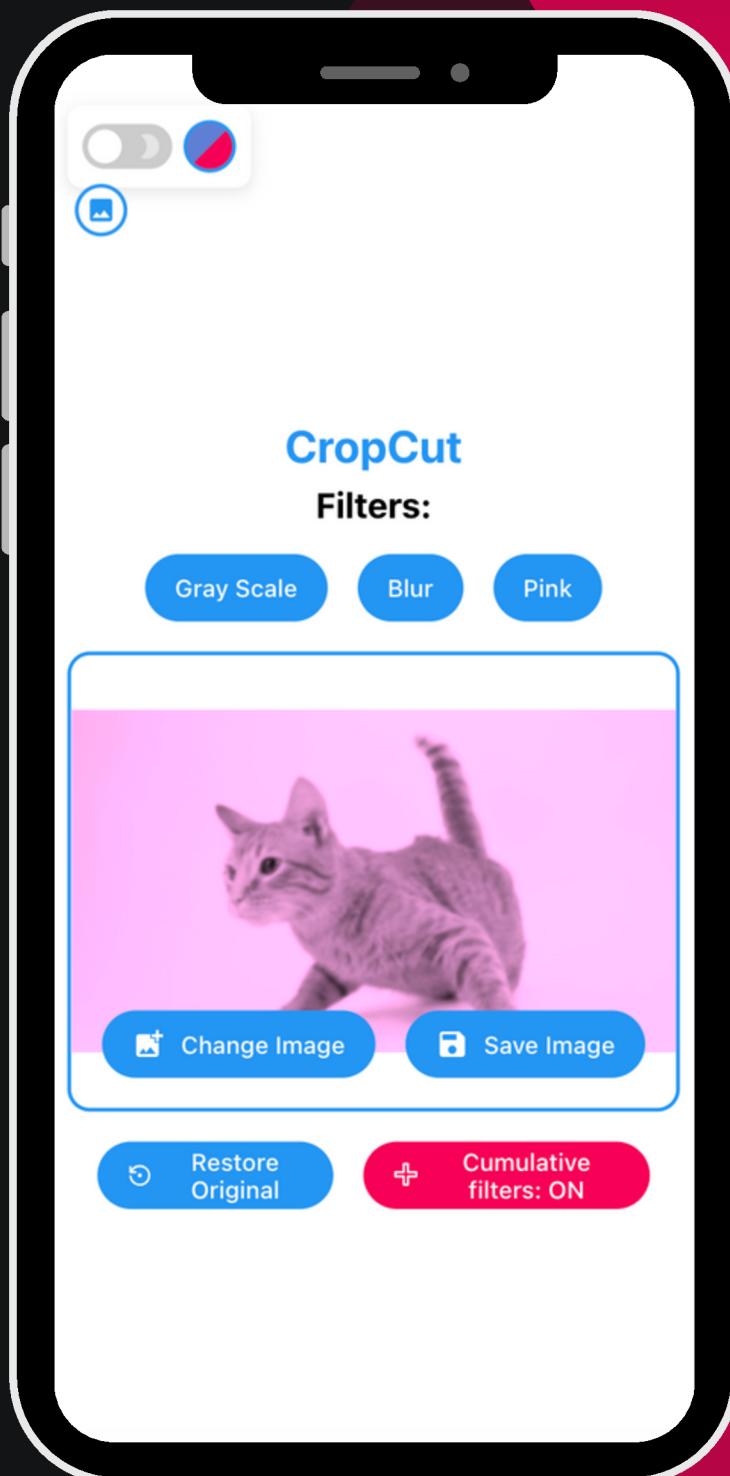
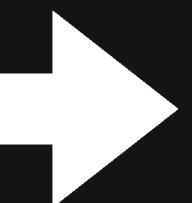


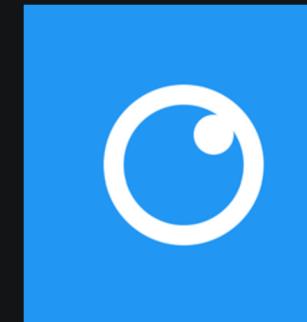


# Extra: Filtros Acumulativos

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

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





**CropCut**

**¡Gracias por su atención!**

---