INFORME TÉCNICO COMPLETO

Transformador GIF / SVG Online

Conversor de Video a GIF Animado y SVG Animado en el Navegador

Responsable del Proyecto

Eduardo Andrés Fierro Duque

Santiago de Chile — Agosto 2026

Índice

1. Resumen Ejecutivo

El Transformador GIF / SVG Online es una herramienta web que convierte videos cortos a formatos animados para web: GIF animado y SVG animado. La operación se realiza íntegramente en el navegador del usuario — el video nunca se sube a ningún servidor — garantizando privacidad total y eliminando costos de infraestructura. La herramienta permite recortar el video, ajustar los FPS de salida, y configurar el tamaño y la calidad del resultado.

A diferencia de las otras herramientas del ecosistema FAP, el Transformador GIF/SVG es la única que utiliza un framework frontend moderno (React + Vite) en lugar de JavaScript vanilla. Esta decisión se justifica por la complejidad de la UI: múltiples controles de configuración interdependientes, vistas previas en tiempo real, y gestión de estado compleja durante el procesamiento asíncrono de video.

2. Stack Tecnológico

Capa	Tecnología
Framework	React 19 (build de producción con Vite) — Bundle compilado: index-C2aVIujH.js
CSS	CSS Modules compilado en archivo único: index-TbJDzSK5.css
Procesamiento Video	HTMLVideoElement + Canvas 2D API para extracción de frames individuales
Encoder GIF	gif.js — Algoritmo LZW (Lempel-Ziv-Welch) ejecutado en Web Workers dedicados (gif.worker.js)
Encoder SVG	Generación procedural de SVG inline con elementos <image> por cada frame y animación SMIL (<animate>)
Entrada	<input type="file"> con aceptación de formatos de video (mp4, webm, mov, etc.)
Exportación	Blob API + URL.createObjectURL para descarga de GIF y SVG
Frontend	HTML5 + CSS3 + React (SPA con punto de entrada en div#root)
Autenticación	Sin verificación de sesión en esta versión — acceso público
Navegación	Navbar FAP estándar — Inicio, Herramientas, Salir

3. Arquitectura del Sistema

La herramienta sigue una arquitectura SPA (Single Page Application) donde React gestiona todo el ciclo de vida de la UI. El flujo principal es: (1) el usuario carga un archivo de video mediante un input file, (2) React renderiza una vista previa del video en un elemento <video> oculto, (3) el usuario configura los parámetros de salida (FPS, dimensiones, rango de recorte, calidad), (4) al iniciar la conversión, se extraen frames individuales del video usando Canvas 2D, (5) los frames se codifican en GIF mediante gif.js en Web Workers o se ensamblan en un SVG animado con SMIL, y (6) el resultado se ofrece como descarga.

4. Análisis de Componentes

4.1 Interfaz de Usuario (index.php)

El archivo index.php (42 líneas) es mínimo: carga el bundle de React compilado por Vite y renderiza un div#root donde React monta la aplicación. Incluye la barra de navegación FAP estándar con logo y enlaces. A diferencia de otras herramientas, no tiene protección de sesión PHP en esta versión, lo que la hace accesible sin registro.

4.2 Bundle React (index-C2aVIujH.js)

El archivo JavaScript compilado contiene la aplicación React completa: componentes de UI (selector de archivo, controles de FPS, sliders de recorte, previsualización), lógica de extracción de frames del video, integración con gif.js para codificación GIF, y generación de SVG animado. Al estar compilado con Vite, el código está minificado y tree-shaken, con un peso total aproximado de 150-250 KB.

4.3 GIF Worker (gif.worker.js)

Script que ejecuta el algoritmo de compresión LZW en un Web Worker dedicado. gif.js coordina múltiples workers para paralelizar la codificación de frames. Cada worker recibe datos de imagen (píxeles RGBA), aplica cuantización de color (reducción de la paleta a 256 colores mediante el algoritmo de corte de mediana), y codifica los datos de imagen en el formato GIF89a.

4.4 Estilos (index-TbJDzSK5.css)

Hoja de estilos compilada que contiene los estilos de todos los componentes React. Utiliza la paleta de colores FAP estándar (amarillo #ffdc00, tinta #070706, naranja #ff4200) y las fuentes Outfit + Plus Jakarta Sans para mantener la consistencia visual con el resto del ecosistema.

5. Proceso de Conversión

5.1 Video a GIF Animado

El pipeline de conversión video→GIF sigue estos pasos:

- Carga del video: el archivo se carga como Blob → URL.createObjectURL → asignado a <video>.src. El video se configura con muted, playsInline y loop.
- Configuración de recorte: el usuario define startTime y endTime (en segundos) para seleccionar el segmento del video a convertir.
- Extracción de frames: se posiciona el video en intervalos regulares (cada 1/FPS segundos). En cada posición, se dibuja el frame actual en un canvas offscreen y se extrae como ImageData.
- Redimensionamiento: cada frame se escala a las dimensiones configuradas usando drawImage con parámetros de destino.
- Codificación GIF: los frames redimensionados se añaden al encoder gif.js con un delay de 1000/FPS milisegundos entre frames. gif.js aplica compresión LZW y cuantización de color.
- Renderizado final: gif.js ensambla el archivo GIF89a completo y lo entrega como Blob. Se crea una URL de descarga con URL.createObjectURL.

5.2 Video a SVG Animado

La conversión a SVG animado sigue un enfoque diferente. En lugar de codificar los píxeles en un formato binario comprimido, se genera un documento SVG que contiene cada frame como un elemento <image> con datos en base64, y se utiliza SMIL (Synchronized Multimedia Integration Language) para animar la visibilidad de cada frame:

```
<svg xmlns='http://www.w3.org/2000/svg' width='W' height='H'>
<image id='frame0' href='...' visibility='visible'>
<animate attributeName='visibility' values='visible;hidden;hidden;...'
dur='TOTAL_DURATION' repeatCount='indefinite'/>
</image>
<image id='frame1' href='...' visibility='hidden'>
<animate attributeName='visibility' values='hidden;visible;hidden;...'
dur='TOTAL_DURATION' repeatCount='indefinite'/>
</image>
...
</svg>
```

La animación SMIL controla la visibilidad de cada frame en secuencia: para N frames, cada <animate> tiene N valores de visibility, con "visible" en la posición correspondiente al momento del frame y "hidden" en todas las demás. La duración total se divide en N intervalos iguales.

6. Fórmulas Matemáticas

6.1 Cálculo de FPS y Duración de Frames

Número de frames totales: N = floor((endTime - startTime) × fps)

Duración de cada frame en el GIF: delay = 1000 / fps (milisegundos)

Duración total del GIF: totalDuration = N × delay (milisegundos)

Ejemplo: video de 3 segundos a 12 fps:

N = 3 × 12 = 36 frames

delay = 1000 / 12 = 83.33... ms por frame

totalDuration = 36 × 83.33 = 3000 ms (3 segundos)

6.2 Algoritmo de Recorte (Trim)

El usuario configura dos marcadores de tiempo:

startTime ∈ [0, videoDuration]

endTime ∈ [startTime, videoDuration]

Para el frame i (0 ≤ i < N):

t_i = startTime + i × (endTime - startTime) / N

video.currentTime = t_i

Esperar evento "seeked" → dibujar frame en canvas

El posicionamiento del video (seek) es inherentemente asíncrono: el navegador debe decodificar el frame en la nueva posición antes de que esté disponible para dibujar. Por eso la extracción de frames se realiza secuencialmente (no en paralelo), esperando el evento "seeked" antes de capturar cada frame.

6.3 Redimensionamiento y Relación de Aspecto

Manteniendo la relación de aspecto original del video:

aspectRatio = videoWidth / videoHeight

Si se fija el ancho W: H = W / aspectRatio

Si se fija el alto H: W = H × aspectRatio

El canvas de salida se dimensiona a (W, H) y cada frame se dibuja:

ctx.drawImage(video, 0, 0, videoWidth, videoHeight, 0, 0, W, H)

La interpolación de píxeles durante el redimensionamiento la realiza el navegador usando el algoritmo configurado en imageSmoothingQuality (por defecto "low" en la mayoría de navegadores, pero configurable a "medium" o "high").

6.4 Compresión LZW (GIF)

El algoritmo LZW (Lempel-Ziv-Welch) es un método de compresión sin pérdida basado en diccionario, utilizado en el formato GIF. El fundamento matemático es:

1. Inicializar diccionario con 256 entradas (valores 0-255 para bytes individuales)

+ código CLEAR (256) + código EOF (257)

2. Leer secuencia de píxeles (índices de paleta) byte por byte:

Sea W el prefijo acumulado, K el siguiente byte

Si W+K está en el diccionario: W = W+K

Si no: emitir código de W, añadir W+K al diccionario, W = K

3. Al llenar el diccionario (4096 entradas máx): emitir CLEAR, reiniciar

La tasa de compresión típica para GIF oscila entre 3:1 y 10:1, dependiendo de la complejidad de la imagen. Imágenes con grandes áreas de color sólido se comprimen mejor que imágenes con texturas complejas o ruido.

6.5 Cuantización de Color para GIF

El formato GIF está limitado a 256 colores por frame. La cuantización reduce los millones de colores posibles (24-bit) a una paleta de 256 colores. El algoritmo más común es el de "corte de mediana" (median cut):

1. Representar todos los colores de la imagen como puntos en el espacio RGB 3D

2. Encontrar la dimensión (R, G o B) con mayor rango

3. Ordenar los colores por esa dimensión y dividir en la mediana

4. Repetir recursivamente hasta tener 256 "cajas"

5. El color representativo de cada caja es el promedio de los colores contenidos

7. Métodos Web Utilizados

- React 19 (Vite build): SPA con componentes funcionales y hooks. Gestión de estado con useState/useReducer. Efectos secundarios con useEffect. Bundle optimizado con tree-shaking y code splitting.
- HTMLVideoElement: Reproducción y seek de video. Propiedades: currentTime, duration, videoWidth, videoHeight, playbackRate. Eventos: seeked, loadedmetadata, canplay.
- Canvas 2D API: Extracción de frames: ctx.drawImage(video, ...). Redimensionamiento con parámetros de origen/destino. Conversión a ImageData para procesamiento.
- gif.js + Web Workers: Codificación LZW en hilos paralelos. gif.worker.js cargado como Worker script. Comunicación vía postMessage: addFrame, render. Callback on('finished') con Blob.
- Blob / URL.createObjectURL: Creación de URL para video de entrada. Creación de blobs descargables para GIF y SVG de salida. Revocación con URL.revokeObjectURL.
- File API: <input type="file"> con aceptación de formatos de video. Lectura del archivo como Blob para asignar a video.src.
- SVG SMIL Animation: Animación declarativa en SVG: elementos <animate> con atributos attributeName, values, dur, repeatCount. Control de visibilidad secuencial de frames.
- CSS Transform: Ajustes visuales en la UI de recorte y previsualización.

8. Modelo de RAM y Rendimiento

■■ *Memoria durante la conversión* ■■

Video original: 5-100 MB (dependiendo de duración y resolución)

Frames extraídos (en cola de gif.js): N × W×H×4 bytes

Ej: 36 frames × 480×360×4 = 36 × 691,200 = 24.9 MB

Paleta de colores: 256 × 3 bytes = 768 bytes (despreciable)

GIF comprimido final: 500 KB - 10 MB (típicamente)

Total RAM pico: ~30-130 MB

Tiempos típicos de conversión: para un clip de 3 segundos a 12 FPS con resolución 480p, el proceso completo toma 5-15 segundos. El cuello de botella es la extracción secuencial de frames (limitada por el tiempo de seek del video) y la codificación LZW (CPU-bound, paliada por Web Workers).

9. Flujo de Datos Completo

[USUARIO] → Seleccionar archivo de video (.mp4, .webm, .mov)
■

■■■ File → Blob → URL.createObjectURL → <video>.src
■ ■■■ video.load() → loadedmetadata → mostrar duración, dimensiones
■
■■■ Configurar parámetros:
■ ■■■ FPS (1-30) → delay = 1000/fps ms
■ ■■■ Dimensiones (W×H) → mantener aspect ratio
■ ■■■ Recorte (inicio/fin) → startTime, endTime en segundos
■ ■■■ Calidad (1-30) → parámetro de cuantización
■
■■■ Iniciar conversión → Para cada frame i en [0..N-1]:
■ ■■■ t = startTime + i × (endTime-startTime)/N
■ ■■■ video.currentTime = t
■ ■■■ Esperar evento 'seeked'
■ ■■■ Canvas offscreen: ctx.drawImage(video, 0,0,vW,vH, 0,0,W,H)
■ ■■■ ImageData = ctx.getImageData(0,0,W,H)
■ ■■■ gif.addFrame(imageData, {delay, copy:true})
■
■■■ gif.render() → LZW compression in Workers → Blob (image/gif)
■■■ URL.createObjectURL(blob) → <a download='video.gif'> → click()

[ALTERNATIVA SVG]
Para cada frame:
■■■ canvas.toDataURL('image/png') → base64 string
■■■ <image href='...' visibility='...'>
■■■ <animate attributeName='visibility' values='...'/>

10. Conclusiones

El Transformador GIF / SVG Online resuelve un problema concreto y frecuente para creadores de contenido: la necesidad de convertir fragmentos de video a formatos animados compatibles con la web sin recurrir a software de escritorio pesado ni servicios en línea que requieren subir el video a servidores externos.

La decisión de utilizar React para esta herramienta (a diferencia del resto del ecosistema que usa vanilla JS) responde a la complejidad de la UI: múltiples controles de configuración con dependencias entre sí (cambiar FPS afecta el número de frames, cambiar dimensiones mantiene la relación de aspecto, etc.) hacen que el modelo de componentes y estado reactivo de React sea más mantenible que un enfoque vanilla.

La generación de SVG animado mediante SMIL es un enfoque innovador y poco común. Mientras que la mayoría de herramientas solo ofrecen GIF, el SVG animado tiene ventajas significativas: es vectorial (escala sin pérdida), el código es editable con cualquier editor de texto, y puede ser indexado por motores de búsqueda. La desventaja es un tamaño de archivo mayor (cada frame se almacena como PNG en base64, lo que añade ~33% de overhead).

— Fin del Documento 07 — Transformador GIF / SVG Online —