INFORME TÉCNICO COMPLETO

FAP Mobile Web

Estudio de Animación 2D para Celulares y Tablets

Responsable del Proyecto

Eduardo Andrés Fierro Duque

Santiago de Chile — Agosto 2026

Índice

1. Resumen Ejecutivo

FAP Mobile Web es un estudio de animación 2D completo que opera íntegramente en el navegador de dispositivos móviles y tablets. A diferencia de aplicaciones nativas, no requiere instalación: el usuario accede mediante una URL, registra su email una única vez, y obtiene acceso inmediato a un entorno de animación cuadro por cuadro con 60 pinceles artísticos, paleta de 66 colores, sistema de capas con onion skin para ver frames anteriores y futuros, timeline interactivo, y exportación a GIF animado y video MP4.

La herramienta está implementada como una Single Page Application (SPA) monolítica en un único archivo PHP de aproximadamente 2800 líneas que contiene HTML, CSS embebido y JavaScript vanilla. La interfaz emula visualmente un teléfono móvil (390×844px) centrado en la pantalla, con bordes redondeados de 36px y sombra que simula el hardware de un dispositivo real. En viewports pequeños (<520px), el modo "teléfono" se desactiva y la interfaz ocupa el 100% de la pantalla real.

2. Stack Tecnológico

Capa	Tecnología
Frontend	HTML5 + CSS3 embebido + JavaScript vanilla ES5
Canvas	HTML5 Canvas 2D API (doble buffer: ctx + fbCtx)
Autenticación	PHP session_start() + verificación de $_SESSION
Animación	requestAnimationFrame con control de FPS variable (8-60)
Exportación GIF	gif.js (encoder LZW en Web Workers)
Exportación Video	MediaRecorder API + canvas.captureStream()
Almacenamiento	Canvas en memoria (frameCanvases, strokeCanvases, undoStacks)

3. Arquitectura del Sistema

3.1 Estructura de la Interfaz

La interfaz está organizada en 7 zonas visuales dispuestas verticalmente dentro del contenedor .phone de 390×844px:

- Toolbar (38px): Navegación de frames, play/stop, contador, selector FPS, exportación
- Color Strip (30px): 66 colores en círculos desplazables horizontalmente
- Brush Strip (30px): 60 pinceles con iconos en miniatura, desplazamiento horizontal
- Canvas Area (flexible): Lienzo de dibujo con relación de aspecto 9:16, zoom y paneo
- Tool Panel (50px): Herramientas (pincel, borrador, relleno, onion skin, mano) + sliders de tamaño y opacidad + zoom
- Timeline Wrap (72px): Scroll horizontal de frames con botones de desplazamiento

3.2 Arquitectura de Buffers

El motor de dibujo utiliza una arquitectura de buffers múltiples que separa las preocupaciones de renderizado:

drawCanvas (ctx) → Capa de visualización (lo que el usuario ve)
frameBuffer (fbCtx) → Capa permanente del frame actual
frameCanvases[key] → Diccionario de canvases persistentes para cada frame
strokeCanvases[key] → Trazos aislados (sin fondo blanco) para onion skin
storage.tmp → Canvas temporal para operaciones de tintado
undoStacks[key] → Array de hasta 30 estados anteriores por frame
redoStacks[key] → Array de estados rehechos por frame

Cada trazo de pincel se pinta simultáneamente en ctx y fbCtx (doble buffer). Al cambiar de frame, el contenido de fbCtx se persiste en frameCanvases[frame_X] y se carga el frame destino. Esta separación permite que el onion skin acceda a los trazos de frames adyacentes sin interferir con el dibujo actual.

3.3 Sistema de Sesión

<?php session_start();
if (!isset($_SESSION['email'])) {
header('Location:/login.php?redirect='.urlencode($_SERVER['REQUEST_URI']));
exit;
} ?>

La protección de acceso es la primera línea del archivo. Si no hay sesión activa, el usuario es redirigido inmediatamente a login.php con la URL actual como destino de retorno, garantizando que al autenticarse regrese exactamente a FAP Mobile.

4. Análisis Línea por Línea

4.1 Estructura HTML y CSS (líneas 1-468)

Las líneas 1-10 establecen la configuración de viewport para aplicaciones móviles web: user-scalable=no impide el zoom accidental (crítico para dibujo táctil), viewport-fit=cover aprovecha la pantalla completa en dispositivos con notch, y apple-mobile-web-app-capable permite funcionar como app instalable en iOS.

Las líneas 11-27 definen el sistema de variables CSS con una paleta oscura profesional (#15171c, #1a1d24, #252830) con acento naranja (#ff6d00). Este tema oscuro fue elegido para (a) reducir la fatiga visual durante sesiones prolongadas de dibujo, (b) maximizar el contraste con el canvas blanco, y (c) emular la estética de aplicaciones profesionales como Procreate o Clip Studio Paint.

Las líneas 47-58 definen el contenedor .phone con dimensiones fijas de 390×844px, bordes redondeados (36px), borde de 2px y triple capa de sombra que simula un dispositivo físico. La media query de la línea 152 (@media max-width:520px) elimina el modo teléfono y expande la interfaz a pantalla completa con altura calculada dinámicamente mediante la variable --vh (100% del viewport real en móviles).

Las líneas 60-75 definen el sistema de botones base con estados hover (8% blanco), active (14% blanco + escala 0.92), disabled (35% opacidad) y active (20% naranja). Los iconos PNG se colorean mediante filtros CSS (invert + sepia + saturate + hue-rotate) para cambiar de blanco a naranja en estado activo, evitando la necesidad de múltiples versiones de cada icono.

4.2 Motor de Dibujo y Sistema de Pinceles (líneas 469-578)

El sistema de pinceles es el corazón de FAP Mobile. Cada pincel está definido como un objeto JavaScript con hasta 10 propiedades que configuran su comportamiento. La función setBrushStyle() (líneas 580-593) aplica estas propiedades al contexto Canvas 2D antes de cada trazo:

```
function setBrushStyle(c, br, color, alpha, lw) {
var sz = br.fixed || lw;
c.fillStyle = color;
c.strokeStyle = color;
c.globalAlpha = alpha;
c.lineWidth = Math.max(1, sz);
c.lineCap = br.cap; // 'round', 'square', 'butt'
c.lineJoin = br.join; // 'round', 'miter'
c.shadowBlur = br.shadow || 0;
c.shadowColor = br.shadow ? color : 'transparent';
if (br.crisp) c.imageSmoothingEnabled = false;
}
```

Los 60 pinceles se pueden clasificar en las siguientes categorías:

- Pinceles básicos (4): round, square, pencil, pixel — utilizan stroke() directo con diferentes lineCap y lineJoin. El pencil fija el ancho a 1px independientemente del tamaño.
- Pinceles de textura (12): chalk, charcoal, crayon, sand, gravel, pastel, dryBrush, oil, acrylic, gouache, watercolor, sponge — generan partículas aleatorias alrededor del trazo usando distribuciones uniformes para posición, tamaño y opacidad.
- Pinceles de patrón (16): dots, hatch, fur, lace, chain, zigzag, wave, brick, diamond, stitch, grid, scales, pebble, tribal, marble, holographic — dibujan formas geométricas repetitivas en cada punto del trazo.
- Pinceles de efecto (16): glow, neon, splatter, drip, airbrush, smoke, sparkle, comet, confetti, rain, bubbles, embers, crackle, frost, drizzle, feather — combinan sombras, transparencia y aleatoriedad para crear efectos atmosféricos.
- Pinceles especiales (12): calligraphy, marker, ink, graffiti, spiderweb, vines, scribble, wire, hardEraser, smudge, rake, holographic — cada uno con lógica de renderizado única en el bloque de dibujo.

4.3 Sistema de Capas y Onion Skin (líneas 579-886)

El onion skin es la técnica de animación que permite ver los frames adyacentes como capas semitransparentes mientras se dibuja el frame actual. FAP Mobile muestra hasta 3 frames anteriores (en rojo) y 3 frames posteriores (en verde). La implementación se divide en dos fases:

- Extracción de capa de trazos (getStrokeLayer, líneas 702-720): Toma el canvas de un frame, analiza cada píxel, y vuelve transparentes todos los píxeles blancos (R>250, G>250, B>250). Esto aísla los trazos del artista del fondo blanco del canvas. La operación recorre el array de píxeles en saltos de 4 (RGBA).
- Tintado y composición (tintAndDraw, líneas 722-736): Dibuja el canvas de trazos en un canvas temporal, aplica composite operation "source-atop" para colorear solo los píxeles no transparentes con el color de tintado (rojo #FF4444 o verde #44FF44), y luego dibuja el resultado en el canvas de visualización con la opacidad calculada.

4.4 Timeline y Control de Frames

El timeline renderiza dinámicamente cajas de 50×50px para cada frame (línea 139). Las funciones addFrame(), removeFrame(), goToFrame() y duplicateFrame() manipulan el estado y reconstruyen el timeline llamando a renderTimeline(). La navegación entre frames llama a saveFrame() (persiste ctx → frameCanvases) seguido de loadFrame() (restaura el frame destino).

4.5 Exportación GIF y Video

Exportación GIF: Utiliza la librería gif.js (encoder LZW en Web Workers). Se renderiza cada frame desde frameCanvases a un canvas temporal, se añade al encoder con un delay calculado como 1000/fps milisegundos, y se renderiza el GIF final. La función nextExportName() genera nombres

secuenciales: FREEANIMATIONPOWER_01.gif, FREEANIMATIONPOWER_02.gif, etc.

Exportación Video: Utiliza MediaRecorder con canvas.captureStream(30) para capturar la reproducción en tiempo real. El video resultante se ofrece como descarga en formato WebM (codec VP9).

5. Fórmulas Matemáticas de FAP Mobile

5.1 Cálculo de Posición del Trazo

La función getPos() (líneas 773-783) convierte coordenadas de pantalla a coordenadas de canvas, compensando la diferencia de tamaño entre el elemento DOM y el buffer interno:

x_canvas = (clientX - rect.left) × (canvas.width / rect.width)

y_canvas = (clientY - rect.top) × (canvas.height / rect.height)

presión = touches ? touches[0].force || 1 : e.pressure || 1

La división canvas.width / rect.width representa la relación entre el tamaño del buffer de píxeles y el tamaño de visualización CSS. Esta relación cambia con el zoom y la densidad de píxeles del dispositivo (devicePixelRatio).

5.2 Interpolación de Trazos y Paso de Dibujo

Dado que los eventos pointermove pueden dispararse a intervalos irregulares, el motor interpola puntos intermedios para garantizar un trazo continuo:

dx = p.x - lastX (diferencia en X desde el último punto)

dy = p.y - lastY (diferencia en Y desde el último punto)

dist = √(dx² + dy²) (distancia euclidiana entre puntos consecutivos)

sz = (br.fixed || state.brushSize × br.scale) × dpr × 0.5

step = max(1, sz × 0.25)

Si dist > step, se considera que los puntos están suficientemente separados para requerir un nuevo segmento de trazo. El valor step = sz × 0.25 (un cuarto del radio del pincel) es un compromiso empírico entre suavidad (step pequeño → más puntos interpolados) y rendimiento (step grande → menos operaciones de dibujo).

5.3 Decaimiento del Onion Skin

La opacidad de cada capa de onion skin sigue una progresión de decaimiento lineal respecto a la distancia en frames:

$\alpha(i) = \alpha\_base \times (1 - (i - 1) \times factor\_decaimiento)$

$donde\ \alpha\_base = 0.35,\ factor\_decaimiento = 0.25,\ i \in \{1, 2, 3\}$

$i=1\ (frame\ adyacente):\ \alpha = 0.35 \times (1 - 0 \times 0.25) = 0.350\ (35\%)$

$i=2\ (a\ 2\ frames):\ \alpha = 0.35 \times (1 - 1 \times 0.25) = 0.263\ (26\%)$

$i=3\ (a\ 3\ frames):\ \alpha = 0.35 \times (1 - 2 \times 0.25) = 0.175\ (18\%)$

La suma total de opacidades de las 6 capas (3 rojas + 3 verdes) no excede el 100% (2 × (0.35 + 0.263 + 0.175) = 1.576), pero como cada capa opera sobre un fondo blanco y los colores se mezclan con composite "source-over", la saturación visual es menor. Los frames más lejanos (i=4+) simplemente no se muestran (el bucle tiene límite 3).

5.4 Algoritmos de los 60 Pinceles — Detalle Matemático

■■ *Splatter* ■■

$N\_partículas = 3 + \blacksquare random(0, 4) \blacksquare$

Para k = 0..N:

$\theta\_k = random(0, 2\pi)$

$r\_k = random(0, sz \times 3)$

$s\_k = random(0.5, sz \times 0.5)$

círculo en $(x + \cos(\theta\_k) \times r\_k,\ y + \sin(\theta\_k) \times r\_k)$ *con radio* s_k

■■ *Caligrafía (Calligraphy)* ■■

Para cada punto del trazo:

Guardar contexto $\rightarrow$ *translate(x, y)* $\rightarrow$ *rotate(ángulo_slant* $\times \pi/180$*)*

$\rightarrow$ *scale(ratio, 1)* $\rightarrow$ *círculo(0, 0, sz)* $\rightarrow$ *restaurar contexto*

Slant fijo = 45°, Ratio fijo = 0.3 $\rightarrow$ *elipse 3.3× más ancha que alta*

■■ *Acuarela (Watercolor)* ■■

$3\ pasadas\ superpuestas\ L \in \{1.0, 0.7, 0.4\}:$

$\alpha\_L = \alpha\_base \times L \times 0.5$

ancho_L = lineWidth × (1.5 - L × 0.3)

jitter_x = random(-ancho_L, +ancho_L)

línea desde (lastX+jitter_x, lastY+jitter_y) hasta (x+jitter_x, y+jitter_y)

■■ *Carboncillo (Charcoal)* ■■

N_pasos = ■dist / (sz × 0.35)■

Para cada paso j:

t = j / N_pasos

cx = lastX + dx×t + random(-sz×6, +sz×6)

cy = lastY + dy×t + random(-sz×6, +sz×6)

α_p = α × random(0.15, 1.0)

rectángulo en (cx, cy) de tamaño random(sz × 0.5)

■■ *Puntillismo (Dots)* ■■

spacing = lineWidth × 2.5

N_pasos = ■dist / spacing■

Para cada paso j: círculo en el punto interpolado con radio sz × 0.6

5.5 Algoritmo Flood Fill (Relleno por Inundación)

La herramienta de relleno (bucket) implementa el algoritmo clásico de Flood Fill por barrido de líneas (scanline flood fill), una optimización del algoritmo recursivo básico que evita el desbordamiento de pila en áreas grandes:

1. Obtener color_objetivo en el píxel (x, y) de partida

2. Si color_objetivo == color_relleno: terminar (ya está relleno)

3. Inicializar pila con (x, y)

4. Mientras la pila no esté vacía:

a. Extraer (x, y) de la pila

b. Avanzar hacia la izquierda hasta encontrar un píxel != color_objetivo

c. Avanzar hacia la derecha pintando hasta encontrar un píxel != color_objetivo

d. Para la fila superior (y-1) e inferior (y+1):

Buscar segmentos contiguos del color_objetivo y apilar sus extremos izquierdos

(solo si no fueron ya procesados en este span)

La complejidad temporal es O(W×H) en el peor caso (toda la imagen del mismo color). La complejidad espacial de la pila es O(W) para cada fila procesada, lo que la hace segura incluso en imágenes grandes (1920×1080 = 2 millones de píxeles).

6. Métodos Web Utilizados

- Canvas 2D API: Motor de renderizado principal: fillRect, arc, beginPath, stroke, fill, getImageData, putImageData, drawImage, globalCompositeOperation, createRadialGradient
- Pointer Events: Captura unificada de mouse y touch: pointerdown, pointermove, pointerup, pointerleave. Propiedad pressure para detectar fuerza del trazo. getCoalescedEvents() para trazos de alta precisión.
- requestAnimationFrame: Bucle de animación para playback y renderizado continuo sincronizado con VSync (60Hz). Control de FPS mediante acumulador de tiempo.
- MediaRecorder API: Exportación de video: captura el stream del canvas (captureStream) y lo codifica como video WebM con codec VP9.
- Web Workers: Encoder GIF (gif.js) ejecutado en hilo separado para no bloquear la UI durante la exportación.
- FileReader API: Lectura de archivos .fap (proyectos guardados) desde el sistema de archivos local.
- URL.createObjectURL / revokeObjectURL: Creación de blobs descargables para GIF, video y archivos .fap. Gestión de memoria con revokeObjectURL.
- localStorage: No se utiliza en esta herramienta (todo el estado es volátil en memoria).
- Blob API: Creación de archivos descargables: GIF (image/gif), Video (video/webm), Proyecto (application/json .fap).
- Drag & Drop: No implementado en esta versión (el canvas acepta eventos de puntero directos).

7. Modelo de RAM y Rendimiento

■■ *Cálculo de Memoria por Frame* ■■

Dimensiones del canvas: 390 × 692 píxeles (ratio 9:16 en contenedor)

Bytes por frame (RGBA): 390 × 692 × 4 = 1,079,520 bytes ≈ 1.03 MB

Cada frame requiere 2 canvases: frameCanvas + strokeCanvas = 2.06 MB

24 frames × 2.06 MB = 49.44 MB (máximo teórico)

Undo stacks (30 por frame): 30 × 1.03 MB = 30.9 MB por frame activo

Total máximo estimado: ~80 MB (24 frames + 1 frame con undo stack completo)

En la práctica, el consumo es menor porque los frameCanvases no se crean hasta que el frame es dibujado (creación lazy), y los undoStacks solo existen para frames que han sido editados. La memoria se libera al llamar a clearAllFrameData() durante el redimensionamiento del canvas.

8. Flujo de Datos Completo

[ENTRADA] Pointer Event (touch/mouse)
■
■■■ startDraw(e) → getPos(e) → coordenadas canvas
■ ■■■ activeTool == 'hand' → activa paneo (isPanning = true)
■ ■■■ activeTool == 'fill' → floodFill(x, y)
■ ■■■ activeTool == 'brush'/'eraser' → isDrawing = true, drawDot(x, y)
■
■■■ moveDraw(e) → getPos(e)
■ ■■■ paneo: viewport.panX/Y += delta/zoom → applyViewport()
■ ■■■ dibujo: interpola puntos (step = sz × 0.25)
■ ■■■ [ctx, fbCtx].forEach → setBrushStyle → algoritmo del pincel
■
■■■ endDraw(e) → commitFrame()
■■■ frameCanvases['frame_'+currentFrame] = cloneCanvas(fbCtx)
■■■ loadFrame(currentFrame) → onion skin → ctx.drawImage(strokeLayer)

[CAMBIO DE FRAME]
saveFrame() → loadFrame(nuevoFrame)
■■■ fbCtx.fillRect(blanco) → fbCtx.drawImage(frameCanvas)
■■■ ctx.fillRect(blanco) → onion skin (3 rojos + 3 verdes)
■■■ ctx.drawImage(strokeLayer actual) → drawGrid → drawSafeZone

[PLAYBACK]
setInterval con periodo 1000/fps ms
■■■ frame = (frame + 1) % totalFrames → loadFrame(frame)

[EXPORTACIÓN]
GIF: frameCanvases → gif.addFrame(canvas, {delay: 1000/fps}) → gif.render()
Video: canvas.captureStream(30) → MediaRecorder → blob WebM

9. Conclusiones

FAP Mobile Web demuestra que es posible construir un estudio de animación 2D completo operando exclusivamente en el navegador móvil, sin depender de WebGL, WebAssembly ni frameworks de UI. La combinación de Canvas 2D API para el renderizado, Pointer Events para la captura unificada de entrada, y requestAnimationFrame para el playback constituye un stack tecnológico mínimo pero sorprendentemente capaz.

El sistema de 60 pinceles representa el componente más sofisticado del motor: cada pincel es efectivamente un pequeño programa generativo que combina aleatoriedad controlada (Math.random con distribuciones uniformes), operaciones geométricas (rotaciones, escalados, traslaciones) y propiedades del contexto Canvas (opacidad, sombras, composite operations) para producir texturas artísticas convincentes.

La arquitectura de buffers múltiples (drawCanvas, frameBuffer, frameCanvases, strokeCanvases) es el patrón arquitectónico clave que habilita funcionalidades avanzadas como el onion skin sin comprometer el rendimiento del dibujo en tiempo real. Esta separación de responsabilidades entre buffers de visualización, persistencia y análisis es directamente análoga a los sistemas de capas de software profesional como Adobe Animate o TVPaint.

— Fin del Documento 02 — FAP Mobile Web —