Desarrollo de Front-END

Tema 7

APIs de HTML5 Parte 1

César Andrés Sánchez

Universidad Camilo José Cela
Escuela Politécnica Superior de Tecnología y Ciencia

Curso 2025 - 2026

APIs de HTML5

“python -m http.server”

1. El problema: abrir archivos con `file://`

- Cuando abrimos un archivo directamente con doble clic, el navegador usa la URL:

  `file:///C:/mis_proyectos/index.html`

- En este contexto **no hay servidor web**, por lo tanto:
  - No se permite usar **Service Workers**.
  - No se cargan **Worklets** ni módulos ES6.
  - Muchas APIs modernas requieren un *contexto seguro*.
- Resultado: errores como *"Failed to register a ServiceWorker"* o *"CSS.paintWorklet no soportado"*.

2. Contexto seguro: qué significa

- Un **contexto seguro** es una página servida por:
  - un protocolo `https://`, o
  - la dirección local `http://localhost`.
- En ese entorno el navegador habilita APIs sensibles como:
  - `Service Worker`
  - `CSS Paint Worklet`
  - `getUserMedia`, `Notification`, etc.

3. Solución práctica: servidor local

Python incluye un mini servidor HTTP integrado:

Comando

```
python -m http.server 5174
```

- Publica los archivos del directorio actual en `http://localhost:5174`.
- Permite cargar módulos, Service Workers y Worklets correctamente.
- No requiere instalación adicional ni configuración.

¡Ahora el navegador confía en la página como si fuera HTTPS!

4. Alternativas posibles

- **Extensión VS Code:** Live Server.
- **Node.js:** `npx http-server -p 5174`
- **Entornos HTTPS locales:** con certificados autocertificados.
- **Despliegue remoto:** GitHub Pages o Netlify (HTTPS automático).

5. Resumen

Modo de acceso	URL	¿Funciona SW/Worklet?
Abrir archivo local	`file://...`	No
Servidor local Python	`http://localhost:5174`	Sí
Sitio HTTPS	`https://...`	Sí

Conclusión

“python -m http.server” nos da un entorno seguro, moderno y compatible para probar todas las APIs de HTML5 sin salir de nuestro equipo.

Ejercicio 1: Crea una mini-aplicación “Notas Offline”

Objetivo del prototipo

- Mostrar un ejemplo unificado con tres APIs de almacenamiento:
  - **Web Storage (localStorage)**: preferencias simples (tema) con TTL.
  - **IndexedDB**: base de datos local de notas.
  - **Cache Storage + Service Worker**: *app shell* offline y *navigation fallback*.
- Arquitectura pensada para PWA y pruebas sin conexión.

Mapa de componentes

- `index.html`, `styles.css`: interfaz y estilos.
- `app.js`: lógica de UI, tema (localStorage con TTL), registro del SW.
- `db.js`: acceso a **IndexedDB** (CRUD de notas).
- `sw.js`: **Service Worker** con precache y **Cache Storage**.
- `offline.html`: página de *fallback* cuando no hay red.
- `manifest.json`: metadatos PWA.

Paso 1 — Preferencias con Web Storage (TTL)

Meta: guardar el tema claro/oscuro con expiración y sincronizar entre pestañas.

- Namespacing: `prefs:theme`
- TTL: 30 días
- Evento `storage`: sincroniza cambios entre pestañas

Código clave: Web Storage + TTL

```
const NS = 'prefs';
const THEME_KEY = `${NS}:theme`;
const THEME_TTL_MS = 30 * 24 * 60 * 60 * 1000; // 30 días

function setWithTTL(key, value, ttlMs){
  const payload = { v: 1, expiry: Date.now() + ttlMs, data: value
  localStorage.setItem(key, JSON.stringify(payload));
}
function getWithTTL(key, fallback){
  const raw = localStorage.getItem(key);
  if (!raw) return fallback;
  try {
    const obj = JSON.parse(raw);
    if (obj.expiry && Date.now() > obj.expiry) {
      localStorage.removeItem(key); return fallback;
    }
    return obj.data;
```

Paso 2 — Notas en IndexedDB

Meta: persistir notas {`id, title, body, updatedAt`} y listarlas por fecha.

- BD: `notes-db` Store: `notes` Índice: `by_updatedAt`
- Operaciones: `put`, `delete`, `listDesc`
- Ventajas: asíncrono, transaccional, datos estructurados

Código clave: IndexedDB (db.js)

```
const DB_NAME = 'notes-db'; const DB_VERSION = 1; const STORE = '

function openDB() {
  return new Promise((resolve, reject) => {
    const req = indexedDB.open(DB_NAME, DB_VERSION);
    req.onupgradeneeded = () => {
      const db = req.result;
      if (!db.objectStoreNames.contains(STORE)) {
        const os = db.createObjectStore(STORE, { keyPath: 'id' })
        os.createIndex('by_updatedAt', 'updatedAt', { unique: fal
      }
    };
    req.onsuccess = () => resolve(req.result);
    req.onerror = () => reject(req.error);
  });
}
export async function putNote(note){ /* tx.readwrite + store.put(
```

Paso 3 — Offline con Cache Storage + SW

Meta: precache del *app shell* y *navigation fallback* a `offline.html`.

- Estrategia: **cache-first** para estáticos.
- `fetch` modo `navigate`: intentar red, si falla mostrar `offline.html`.
- Actualización de versión de caché: `app-shell-v2`.

Código clave: Service Worker (sw.js)

```
const CACHE_NAME = 'app-shell-v2';
const ASSETS = ['./','./index.html','./styles.css','./app.js','./

self.addEventListener('install', (e) => {
  e.waitUntil(caches.open(CACHE_NAME).then(c => c.addAll(ASSETS))
  self.skipWaiting();
});
self.addEventListener('activate', (e) => {
  e.waitUntil(caches.keys().then(keys => Promise.all(keys.filter(
  self.clients.claim();
});
// Cache-first para estáticos (GET)
self.addEventListener('fetch', (event) => {
  const { request } = event;
  if (request.mode === 'navigate') {
    event.respondWith((async () => {
      try { const preload = await event.preloadResponse; if (prel
```

Flujo completo

1. Usuario abre la app → SW instala y precachea el *app shell*.
2. Elige tema → se guarda en `localStorage` con TTL y se sincroniza entre pestañas.
3. Crea notas → se guardan en **IndexedDB**.
4. Cierra y vuelve → datos persisten localmente.
5. Sin conexión → navegación cae a `offline.html`; estáticos desde caché.

Cómo probar (pasos prácticos)

1. Descargar el ZIP y descomprimir.
2. Iniciar un servidor estático: `python -m http.server 5173`
3. Abrir: `http://localhost:5173`
4. Cambiar tema, crear notas, abrir en otra pestaña (ver sincronización).
5. Activar modo avión / desconectar y recargar (ver `offline.html`).

Buenas prácticas y extensiones

- **Seguridad:** no guardar secretos en Web Storage; usa cookies HttpOnly para auth.
- **Privacidad:** pide permisos sólo cuando sea necesario.
- **Rendimiento:** evita escribir en bucles intensivos; usa workers para CPU pesada.
- **Extensiones:** Background Sync, pantalla de *cola de cambios*, compresión de datos.

Ejercicio 2: Concurrencia

Ejercicio 2

Concurrencia

- Dedicated Worker: cálculo de primos con progreso sin bloquear la UI.
- Shared Worker: contador compartido entre pestañas (todas ven el mismo valor).
- Service Worker: app-shell offline con precache y fallback de navegación.
- CSS Paint Worklet (Houdini): fondo dinámico generado fuera del main thread.

Ejercicio 2

Qué probar

- Dedicated Worker
  Ajusta “Máximo” y pulsa Iniciar → verás el progreso y el tiempo total.
  Pulsa Detener para terminar el worker.
- Shared Worker
  Pulsa Incrementar y Reset.
  Abre otra pestaña en la misma URL: el valor se sincroniza automáticamente.
- Service Worker
  La primera visita precachea archivos.
  Desconéctate y recarga → la app sigue funcionando (modo offline básico).
- Paint Worklet
  Cambia Hue y Espaciado y pulsa Aplicar → el fondo se redibuja desde el worklet.

Objetivo de la demo

- Entender **cómo paralelizar tareas** y **no bloquear la UI** en la Web.
- Ver en funcionamiento:
  - **Dedicated Worker**: CPU pesada con progreso.
  - **Shared Worker**: estado compartido entre pestañas.
  - **Service Worker**: app-shell offline.
  - **CSS Paint Worklet**: render dinámico del fondo.
- Código base: `index.html`, `styles.css`, `main.js`, `sw.js`, `workers/*`, `worklets/*`.

Preparación del entorno

1. Servir por `http://localhost`: `python -m http.server 5174`
2. Abrir `http://localhost:5174`
3. Ver en DevTools (F12) la consola y el Application tab (Service Workers).

Motivo

APIs modernas (SW, Worklets) requieren **contexto seguro**: HTTPS o localhost.

Paso 1 — Dedicated Worker (cálculo de primos)

- **Problema:** Cálculo intensivo bloquea el hilo principal.
- **Idea:** Mover la lógica a un *Dedicated Worker* y reportar progreso.
- UI: input del máximo, botones *Iniciar/Detener*, barra de progreso.

Código (main.js) — lanzar y comunicar

```
// Arranque del worker
let primeWorker = new Worker('./workers/dedicated.js', { type: 'module' });
primeWorker.onmessage = (e) => {
  const msg = e.data;
  if (msg.type === 'progress') {
    primeProgress.max = msg.max;
    primeProgress.value = msg.value;
  } else if (msg.type === 'done') {
    primeOutput.textContent += `Total: ${msg.count}, Tiempo: ${msg.ms} ms`;
    primeWorker.terminate(); primeWorker = null;
  }
};
primeWorker.postMessage({ task: 'primes', max: Number(primeMax.value) || 400000 });
```

Código (workers/dedicated.js) — trabajo pesado

```
self.onmessage = (e) => {
  const { task, max } = e.data || {};
  if (task !== 'primes') return;
  const t0 = performance.now(), primes = [];
  for (let n=2; n<=max; n++){
    if (isPrime(n, primes)) primes.push(n);
    if ((n & 0x7FFF) === 0) self.postMessage({ type:'progress', value:n, max });
  }
  const ms = Math.round(performance.now() - t0);
  self.postMessage({ type:'done', count: primes.length, ms });
};
```

Paso 2 — Shared Worker (estado entre pestañas)

- **Caso de uso:** un contador común a todas las pestañas del mismo origen.
- **Conexión:** cada pestaña abre un puerto, el worker *broadcast* a todos.

Código (main.js) — conectar y usar

```
const sw = new SharedWorker('./workers/shared.js', { type: 'module' });
const port = sw.port; port.start();
port.onmessage = (e) => { if (e.data.type === 'count') sharedVal.textContent = e.data.value;
    };
port.postMessage({ type: 'get' });
incBtn.addEventListener('click', () => port.postMessage({ type: 'inc' }));
resetBtn.addEventListener('click', () => port.postMessage({ type: 'reset' }));
```

Código (workers/shared.js) — broadcast

```
let count = 0; const ports = new Set();
function broadcast(){ for (const p of ports) p.postMessage({ type:'count', value: count }); }
onconnect = (e) => {
  const port = e.ports[0]; ports.add(port); port.start();
  port.onmessage = (ev) => {
    const { type } = ev.data || {};
    if (type === 'inc') { count++; broadcast(); }
    else if (type === 'reset') { count = 0; broadcast(); }
    else if (type === 'get') { port.postMessage({ type:'count', value: count }); }
  };
};
```

Paso 3 — Service Worker (offline básico)

- **Objetivo:** precache del *app shell* y disponibilidad offline.
- **Ciclo:** *install* (cachear) $\rightarrow$ *activate* (limpieza) $\rightarrow$ *fetch* (servir).

Código (sw.js) — precache y fetch

```
const CACHE = 'concurrency-shell-v1';
const ASSETS =
     ['./','./index.html','./styles.css','./main.js','./workers/dedicated.js','./workers/shared.js','

self.addEventListener('install', (event) => {
  event.waitUntil(caches.open(CACHE).then(c => c.addAll(ASSETS)));
  self.skipWaiting();
});
self.addEventListener('activate', (event) => {
  event.waitUntil(caches.keys().then(keys =>
     Promise.all(keys.filter(k=>k!==CACHE).map(k=>caches.delete(k)))));
  self.clients.claim();
});
self.addEventListener('fetch', (event) => {
  const { request } = event;
  if (request.mode === 'navigate') {
    event.respondWith(fetch(request).catch(() => caches.match('./')));
    return;
  }
  if (request.method !== 'GET') return;
  event.respondWith(caches.match(request).then(cached => cached || fetch(request)));
});
```

Paso 4 — CSS Paint Worklet (Houdini)

- **Rol:** pintar un fondo cuadriculado calculado fuera del main thread.
- **Props custom:** `--hue` y `--gap` controlan color/espaciado.
- **Fallback:** CSS `@supports` si el navegador no soporta Paint Worklet.

Código (worklets/gridbg.js) — render del fondo

```
class GridBG {
  static get inputProperties() { return ['--hue','--gap']; }
  paint(ctx, geom, props){
    const hue = parseFloat(props.get('--hue')) || 210;
    const gap = parseFloat(props.get('--gap')) || 24;
    ctx.fillStyle = `hsl(${hue} 40% 12%)`;
    ctx.fillRect(0,0,geom.width, geom.height);
    ctx.strokeStyle = `hsl(${hue} 70% 54%)`;
    for (let x=0; x<geom.width; x+=gap) { ctx.beginPath(); ctx.moveTo(x,0);
     ctx.lineTo(x,geom.height); ctx.stroke(); }
    for (let y=0; y<geom.height; y+=gap) { ctx.beginPath(); ctx.moveTo(0,y);
     ctx.lineTo(geom.width,y); ctx.stroke(); }
  }
}
registerPaint('gridbg', GridBG);
```

Inicialización (main.js) — detección y registro

```
async function setupPaintWorklet(){
  const demo = document.querySelector('.worklet-demo');
  if (!('paintWorklet' in CSS)) return; // Fallback CSS
  if (CSS.registerProperty) {
    try { CSS.registerProperty({ name: '--hue', syntax: '<number>', inherits: false,
     initialValue: 210 }); } catch {}
    try { CSS.registerProperty({ name: '--gap', syntax: '<length>', inherits: false,
     initialValue: '24px' }); } catch {}
  }
  await CSS.paintWorklet.addModule('./worklets/gridbg.js');
  if (!demo.style.getPropertyValue('--hue')) demo.style.setProperty('--hue', '210');
  if (!demo.style.getPropertyValue('--gap')) demo.style.setProperty('--gap', '24px');
}
setupPaintWorklet();
```

Guía de pruebas rápidas

1. **Dedicated Worker:** iniciar, ver progreso, detener, medir ms.
2. **Shared Worker:** abrir dos pestañas, incrementar y resetear (sincronía).
3. **Service Worker:** recargar offline, ver que sigue operativo.
4. **Paint Worklet:** cambiar hue/gap y observar el fondo; probar fallback.

Errores comunes y soluciones

- ✗ *"Failed to register ServiceWorker"* $\Rightarrow$ no se sirve desde localhost/HTTPS.
- ✗ *"Failed to load module for paintWorklet"* $\Rightarrow$ ruta o servidor ausente.
- ✗ SharedWorker no soportado $\Rightarrow$ navegador antiguo/limitado.
- ✓ Abrir DevTools $\rightarrow$ Console & Application $\rightarrow$ verificar SW/Worklet.

Ejercicio 3: Renderizado

Objetivos del demo

- Explorar **cinco rutas de render** en la Web moderna.
- Comparar **modelo de ejecución**, **capas de abstracción** y **requisitos**.
- Probar **fallbacks** cuando una API no está disponible.

Preparación del entorno

1. Servir por `http://localhost`: `python -m http.server 5175`
2. Abrir `http://localhost:5175`
3. Contexto seguro requerido para OffscreenCanvas y WebGPU.

Estructura

`index.html`, `styles.css`, `main.js`, `webgl.js`, `webgpu.js`, `workers/offscreen-worker.js`

Paso 1 — Canvas 2D

Idea: animar pelotas rebotando con degradado radial.

- API inmediata (estado mutable): `fillRect`, `arc`, `createRadialGradient`.
- Bucle con `requestAnimationFrame`.

Código clave (Canvas 2D)

```
const c2d = document.getElementById('c2d');
const ctx = c2d.getContext('2d'), balls = [];
function step(){
  ctx.fillStyle = 'rgba(10,15,26,0.35)';
  ctx.fillRect(0,0,c2d.width,c2d.height);
  for (const b of balls){
    b.x+=b.vx; b.y+=b.vy;
    if (b.x<b.r||b.x>c2d.width-b.r) b.vx*=-1;
    if (b.y<b.r||b.y>c2d.height-b.r) b.vy*=-1;
    ctx.beginPath();
    const g=ctx.createRadialGradient(b.x,b.y,1,b.x,b.y,b.r);
    g.addColorStop(0,`hsla(${b.hue} 90% 65% / .95)`);
    g.addColorStop(1,`hsla(${b.hue} 70% 40% / .15)`);
    ctx.fillStyle=g; ctx.arc(b.x,b.y,b.r,0,Math.PI*2); ctx.fill();
  }
  requestAnimationFrame(step);
}
step();
```

Paso 2 — OffscreenCanvas en Worker

Meta: mover dibujo al **Worker** para no bloquear el hilo principal.

- `canvas.transferControlToOffscreen()`.
- En el Worker: `canvas.getContext('2d')` y `requestAnimationFrame`.
- Fallback: si no hay soporte, dibujar en main thread.

Main thread → Worker (OffscreenCanvas)

```
const c = document.getElementById('osc');
if ('OffscreenCanvas' in window) {
  const off = c.transferControlToOffscreen();
  const w = new Worker('./workers/offscreen-worker.js', { type:'module' });
  w.postMessage({ canvas: off, width: c.width, height: c.height }, [off]);
} else {
  // Fallback: dibujar en main thread
}
```

Worker (offscreen-worker.js)

```
let ctx, W=640, H=360, t=0;
onmessage = (e) => {
  const { canvas, width, height } = e.data || {};
  if (canvas) { ctx = canvas.getContext('2d'); W=width; H=height; loop(); }
};
function loop(){
  t += 0.016;
  ctx.fillStyle = 'rgba(10,15,26,0.25)'; ctx.fillRect(0,0,W,H);
  for (let i=0;i<60;i++){
    const x=(Math.sin(i*.17 + t*.8)*.45+.5)*W;
    const y=(Math.cos(i*.21 + t*.6)*.45+.5)*H;
    const r=3+2*Math.sin(t*1.5+i);
    ctx.beginPath(); ctx.fillStyle=`hsla(${(i*12)%360} 80% 60% / .8)`;
    ctx.arc(x,y,r,0,Math.PI*2); ctx.fill();
  }
  requestAnimationFrame(loop);
}
```

Paso 3 — SVG interactivo

Meta: gráfico con línea + área y **tooltip** en puntos.

- Escalado manual a `viewBox (640x360)`.
- Eventos de ratón sobre `circle`.

Código clave (SVG)

```
const data = Array.from({length:24}, (_,i)=>({x:i, y: 40 + Math.sin(i/2)*20}));
const scaleX = x => 20 + x*(600/24);
const scaleY = y => 320 - (y-0)*(280/80);
let d=''; data.forEach((p,i)=>{ d += (i?'L':'M')+scaleX(p.x)+' '+scaleY(p.y)+' '; });
const path = document.createElementNS('http://www.w3.org/2000/svg','path');
path.setAttribute('d', d); path.setAttribute('fill','none');
     path.setAttribute('stroke','#60a5fa');
plot.appendChild(path);
```

Paso 4 — WebGL 1/2

Meta: triángulo con colores por vértice.

- Intentar **WebGL2**; si falla, caer a **WebGL1**.
- Compilar shaders y `drawArrays(GL_TRIANGLES)`.

Código clave (WebGL)

```
let gl = canvas.getContext('webgl2'); let isGL2 = true;
if (!gl) { gl = canvas.getContext('webgl'); isGL2 = false; }
const vs2 = `#version 300 es
in vec2 pos; in vec3 col; out vec3 vcol;
void main(){ vcol=col; gl_Position=vec4(pos,0.,1.); }`;
const fs2 = `#version 300 es
precision mediump float; in vec3 vcol; out vec4 outColor;
void main(){ outColor=vec4(vcol,1.); }`;
// (si GL1: usar atributos/varyings legacy)
```

Paso 5 — WebGPU (si disponible)

Meta: triángulo WGSL.

- Requiere navegador compatible y contexto seguro.
- Pipeline de render: `device` → `pipeline` → `draw(3)`.

Código clave (WebGPU)

```
if (!('gpu' in navigator)) { /* fallback: ocultar canvas y log */ }
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();
const context = canvas.getContext('webgpu');
const format = navigator.gpu.getPreferredCanvasFormat();
context.configure({ device, format });
const shader = device.createShaderModule({ code: `
@vertex fn vs_main(@location(0) pos: vec2f, @location(1) col: vec3f)
  -> @builtin(position) vec4f { return vec4f(pos,0.,1.); }
@fragment fn fs_main(@location(1) col: vec3f) -> @location(0) vec4f
  { return vec4f(col,1.); }` });
// ... crear pipeline, buffer de vértices, encode y submit
```

Guía de pruebas rápidas

1. **Canvas 2D:** añadir/limpiar bolas; observar *trail*.
2. **OffscreenCanvas:** iniciar/detener; probar CPU ocupada y ver que la UI sigue fluida.
3. **SVG:** pasar el ratón por puntos y ver tooltip.
4. **WebGL:** verificar en el log si es GL2 o GL1.
5. **WebGPU:** si existe, ver mensaje de éxito; si no, fallback.

Errores comunes y soluciones

- ✗ WebGPU no disponible $\Rightarrow$ navegador sin soporte o no es HTTPS/localhost.
- ✗ OffscreenCanvas falla $\Rightarrow$ usar fallback a Canvas 2D en main thread.
- ✗ WebGL error de shader $\Rightarrow$ revisar versión (#version 300 es) vs GL1.