Desarrollo de Front-END

Tema 10

APIs de ejemplo

César Andrés Sánchez

Universidad Camilo José Cela
Escuela Politécnica Superior de Tecnología y Ciencia

Curso 2025 - 2026

APIs de ejemplo

Comparativa global de APIs vistas

- **Servicios multimedia**:
  - **Vídeo**: YouTube, Vimeo.
  - **Audio**: SoundCloud.
- **Servicios cartográficos**:
  - **Mapas libres**: OpenStreetMap + Leaflet.
  - **Mapas propietarios**: Google Maps JavaScript API.
- **Servicios de redes sociales**:
  - Twitter (X), Instagram, widgets sociales en general.
- En todos los casos:
  - Se ofrece una API JavaScript y/o widgets embebibles.
  - La página web actúa como contenedor e integra funciones avanzadas sin necesidad de reimplementarlas.

Comparativa: modelo de datos y control

- **Control del contenido**:
  - YouTube, Vimeo, SoundCloud, Twitter, Instagram: el contenido reside en servidores externos.
  - OSM + Leaflet, Google Maps: datos cartográficos servidos por terceros, pero la lógica de representación puede personalizarse más.
- **Nivel de control programático**:
  - **Alto**: YouTube IFrame API, Vimeo Player, Leaflet, Google Maps (múltiples eventos y métodos).
  - **Medio/Bajo**: widgets de Twitter, Instagram, SoundCloud, con APIs más acotadas.
- **Modelo de interacción**:
  - Basado en eventos (*callbacks*, *promises*).
  - Comunicación `postMessage` en el caso de iframes.

Comparativa: licencias y costes

- **Datos abiertos**:
  - OpenStreetMap: datos libres reutilizables bajo licencia abierta.
  - La representación (tiles) puede ser libre o de pago, según el servidor utilizado.
- **Servicios propietarios**:
  - Google Maps, YouTube, Vimeo, SoundCloud, Twitter, Instagram: políticas de uso y licencias propias.
  - En algunos casos existen cuotas, modelos de suscripción o monetización.
- **Implicaciones**:
  - Es necesario revisar términos de uso y políticas de privacidad.
  - En proyectos académicos o de baja escala suelen existir opciones gratuitas o con límites generosos.
  - Para proyectos comerciales, la sostenibilidad de costes debe planificarse desde el diseño.

Presente: tendencias actuales en integración de APIs

- **Estandarización en torno a JavaScript**:
  - APIs que exponen objetos, métodos y eventos coherentes con el ecosistema web moderno.
  - Uso intensivo de *promises* y programación asíncrona.
- **Importancia del rendimiento**:
  - Carga diferida (lazy load) de widgets y mapas.
  - Minimización del impacto de scripts de terceros.
- **Privacidad y consentimiento**:
  - Cumplimiento de normativas (por ejemplo, RGPD en Europa).
  - Aparición de patrones de diseño que piden consentimiento antes de cargar contenido embebido.
- **Multiplataforma**:
  - APIs diseñadas para funcionar bien en dispositivos móviles y aplicaciones híbridas.

Futuro: evolución probable de estas APIs

- **Mayor abstracción y componentes web**:
  - Encapsulamiento en componentes reutilizables (Web Components, frameworks modernos).
  - Configuración mediante atributos y propiedades declarativas.
- **Más énfasis en la privacidad**:
  - APIs que minimizan el rastreo del usuario.
  - Mecanismos más finos de control de permisos.
- **Interoperabilidad con datos propios**:
  - Superposición de datos de sensores, IoT, analítica, etc.
  - Integración con estándares geoespaciales (GeoJSON, WMS, etc.).
- **Mayor uso de tiempo real**:
  - Vídeo en directo, mapas en tiempo real, flujos sociales actualizados mediante WebSockets o APIs *streaming*.

Otras fuentes de datos y APIs geoespaciales

- **Servicios OGC (Open Geospatial Consortium)**:
  - WMS (Web Map Service), WFS (Web Feature Service), WCS (Web Coverage Service).
  - Permiten publicar datos geoespaciales con estándares abiertos.
- **Plataformas de datos abiertos**:
  - Portales de datos abiertos de administraciones públicas.
  - APIs REST para acceso a catálogos de datos (transporte, medio ambiente, demografía, etc.).
- **Servicios de enrutamiento sobre OSM**:
  - OSRM, GraphHopper, etc., con APIs HTTP/JSON para rutas, tiempos de viaje, isócronas.
- Estos servicios pueden integrarse en la misma interfaz que Leaflet o Google Maps, enriqueciendo el mapa con funcionalidad avanzada.

Otras fuentes de contenido multimedia y social

- **Plataformas de vídeo y streaming**:
  - Twitch, DailyMotion, plataformas institucionales de vídeo.
  - APIs similares basadas en iframes o reproductores JavaScript.
- **Plataformas de presentación y documentos**:
  - SlideShare, servicios de ofimática en la nube (Google Slides, Office 365) con APIs para inserción de documentos.
- **Otras redes sociales**:
  - Facebook, LinkedIn y plataformas emergentes con widgets incrustables.
- Todas ellas siguen patrones similares:
  - Código de incrustación + script oficial.
  - Configuración mediante atributos y datos estructurados.

Criterios para elegir una API u otra

- **Requisitos funcionales**:
  - ¿Qué tipo de contenido necesito (vídeo, audio, mapa, social)?
  - ¿Necesito control fino del reproductor o sólo incrustación básica?
- **Licencia y modelo de coste**:
  - ¿Es compatible con el modelo de negocio o con los requisitos del proyecto?
  - ¿Existen alternativas libres (por ejemplo, OSM frente a Google Maps)?
- **Privacidad, seguridad y datos**:
  - ¿Qué datos se envían al proveedor?
  - ¿Se respetan las normativas aplicables?
- **Facilidad de integración y mantenimiento**:
  - Calidad de la documentación.
  - Soporte a largo plazo y estabilidad de la API.

Resumen comparativo y líneas de proyecto

- Las APIs estudiadas ilustran diferentes familias de servicios:
  - Multimedia, cartografía y redes sociales.
- **Presente**:
  - Integración ubicua de contenido externo en aplicaciones web.
  - Creciente preocupación por rendimiento y privacidad.
- **Futuro**:
  - Mayor modularidad (componentes web, microfrontends).
  - APIs más declarativas y centradas en la experiencia de usuario.
  - Combinación de fuentes abiertas y servicios comerciales.
- **Aplicación práctica**:
  - Diseñar un prototipo que combine varias de estas APIs (por ejemplo, mapa OSM + vídeo explicativo + feed social), analizando sus ventajas e inconvenientes.

Integración de servicios externos en la Web

- En muchas aplicaciones web modernas es habitual integrar servicios de terceros:
  - Reproductores de vídeo (YouTube, Vimeo, etc.).
  - Mapas interactivos (OpenStreetMap, Google Maps).
  - Sistemas de autenticación, pasarelas de pago, etc.
- La integración suele hacerse mediante APIs JavaScript suministradas por el proveedor.
- Estas APIs abstraen la complejidad del servicio remoto y exponen una interfaz orientada a objetos.
- En esta sección estudiamos la API de IFrame de YouTube como ejemplo representativo.

API IFrame de YouTube: conceptos básicos

- YouTube proporciona una API basada en *iframes* para incrustar reproductores de vídeo.
- El reproductor se renderiza en un elemento HTML `<iframe>` gestionado por la librería de YouTube.
- La API expone una serie de métodos y eventos que permiten:
  - Reproducir, pausar, detener un vídeo.
  - Cambiar el vídeo cargado en el reproductor.
  - Escuchar cambios de estado (inicio, pausa, fin, etc.).
- La comunicación entre la página y el `iframe` se realiza mediante *postMessage*.

Flujo general de integración de YouTube

Insertar un reproductor de vídeo de YouTube en JavaScript es conceptualmente sencillo:

- En el cuerpo del HTML incluimos un `<div>` con un identificador, donde irá el reproductor.
- Cargamos el script `https://www.youtube.com/iframe_api` de forma síncrona o asíncrona.
- En la función `onYouTubeIframeAPIReady()`, creamos una instancia de `YT.Player` y fijamos:
  - Tamaño del reproductor (ancho y alto).
  - Identificador del vídeo (`videoId`).
  - Eventos relevantes (por ejemplo, `onReady`).
- En la función `onPlayerReady()`, podemos invocar: `event.target.playVideo()`

Estructura HTML mínima para YouTube

- La estructura básica del documento HTML podría ser:

```html
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <title>Ejemplo YouTube IFrame API</title>
  </head>
  <body>
    <!-- Contenedor donde se insertará el reproductor -->
    <div id="mi_reproductor"></div>

    <!-- Aquí se incluirá el código JavaScript de integración -->
    <script src="js/youtube.js"></script>
  </body>
</html>
```

- El elemento `div` actúa como ancla del reproductor, que será gestionado por `YT.Player`.

Carga asíncrona del script de YouTube

- Es recomendable cargar la API de forma asíncrona para no bloquear el renderizado de la página.
- El patrón habitual consiste en insertar dinámicamente un nuevo elemento `<script>`:

```
// Carga asíncrona del API del IFrame Player de YouTube
let tag = document.createElement('script');
tag.src = "https://www.youtube.com/iframe_api";
let firstScriptTag = document.getElementsByTagName('script')[0];
firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);
```

- Una vez cargado el script remoto, YouTube invocará a `onYouTubeIframeAPIReady()`.

Creación de la instancia `YT.Player`

- La función `onYouTubeIframeAPIReady()` es un *callback* global que debe estar definido.
- Dentro de ella creamos el objeto `YT.Player`:

```
let player;

function onYouTubeIframeAPIReady() {
  player = new YT.Player('mi_reproductor', {
    height: '768',
    width: '1024',
    videoId: 'lKCCZTUx0sI',
    events: {
      'onReady': onPlayerReady
    }
  });
}
```

- El primer argumento es el *id* del elemento HTML contenedor.
- El segundo argumento es un objeto de configuración con opciones y eventos.

Gestión del evento onReady

- La API llama a `onPlayerReady(event)` cuando el reproductor está listo.
- El parámetro `event` contiene, entre otros, la referencia al reproductor en `event.target`.
- Un comportamiento sencillo: iniciar automáticamente la reproducción.

```
// El API llama a esta función cuando el reproductor esté listo
function onPlayerReady(event) {
  event.target.playVideo();
}
```

- En aplicaciones reales, suele ser más adecuado respetar las preferencias del usuario y no hacer *autoplay* en todos los casos.

Ejemplo completo de integración básica

```
// Carga asíncrona del API del IFrame Player de YouTube
let tag = document.createElement('script');
tag.src = "https://www.youtube.com/iframe_api";
let firstScriptTag = document.getElementsByTagName('script')[0];
firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);

// Creación de un iframe con reproductor de YouTube
let player;
function onYouTubeIframeAPIReady() {
  player = new YT.Player('mi_reproductor', {
    height: '768',
    width: '1024',
    videoId: 'lKCCZTUx0sI',
    events: {
      'onReady': onPlayerReady
    }
  });
}

function onPlayerReady(event) {
  event.target.playVideo();
}
```

Parámetros avanzados: `playerVars`

- El objeto de configuración admite la clave `playerVars` para parámetros adicionales:
  - `autoplay`: inicio automático de reproducción.
  - `controls`: mostrar u ocultar controles.
  - `start`, `end`: segundos de inicio y fin.
  - `mute`: iniciar el vídeo sin sonido.

```
player = new YT.Player('mi_reproductor', {
  height: '360',
  width: '640',
  videoId: 'lKCCZTUx0sI',
  playerVars: {
    autoplay: 0,
    controls: 1,
    start: 10
  },
  events: { 'onReady': onPlayerReady }
});
```

Control programático del reproductor

- Una vez creado, `player` expone múltiples métodos:
  - `playVideo()`, `pauseVideo()`, `stopVideo()`.
  - `seekTo(segundos, allowSeekAhead)` para cambiar la posición.
  - `mute()`, `unMute()`, `setVolume(valor)`.
  - `loadVideoById(id)` para cargar otro vídeo.

```
document.getElementById('btn_pausa').addEventListener('click', () => {
  player.pauseVideo();
});

document.getElementById('btn_reinicia').addEventListener('click', () => {
  player.seekTo(0, true);
  player.playVideo();
});
```

Eventos del reproductor de YouTube

- Además de onReady, podemos registrar onStateChange:

```
function onYouTubeIframeAPIReady() {
  player = new YT.Player('mi_reproductor', {
    videoId: 'lKCCZTUx0sI',
    events: {
      'onReady': onPlayerReady,
      'onStateChange': onPlayerStateChange
    }
  });
}

function onPlayerStateChange(event) {
  if (event.data === YT.PlayerState.ENDED) {
    console.log("El vídeo ha terminado");
  }
}
```

- El manejo de eventos permite sincronizar la reproducción con otros componentes de la página.

Privacidad, accesibilidad y buenas prácticas

- YouTube ofrece el dominio `youtube-nocookie.com` para mejorar la privacidad.
- Debe incluirse texto alternativo y controles accesibles para personas con discapacidad.
- Conviene:
  - Evitar el *autoplay* cuando pueda resultar intrusivo.
  - Ajustar el tamaño del reproductor de forma adaptable (*responsive*).
  - Gestionar adecuadamente errores de red o vídeos no disponibles.
- La integración con YouTube debe considerar también las condiciones de uso del servicio.

OpenStreetMap: visión general

OpenStreetMap (OSM) es un proyecto colaborativo cuyo objetivo es crear un mapa del mundo editable y libre.

- Creado por Steve Coast en el Reino Unido en 2004.
- Filosofía similar a Wikipedia:
  - Contenido generado por la comunidad.
  - Licencias abiertas que permiten reutilización.
- Funcionalidad y calidad comparables a Google Maps en muchos contextos.
- Permite descarga y reutilización de los datos geográficos de forma libre.

Modelo de datos de OpenStreetMap

- OSM se basa en tres tipos de elementos fundamentales:
  - **Nodos**: puntos con coordenadas (latitud, longitud).
  - **Vías**: secuencias ordenadas de nodos (calles, ríos, límites).
  - **Relaciones**: estructuras lógicas que agrupan nodos y vías (rutas de transporte, áreas administrativas, etc.).
- Todos los elementos se anotan mediante pares *clave-valor* (*tags*).
- Este modelo es muy flexible y permite representar múltiples tipos de entidades geográficas.

Ecosistema de herramientas OSM

- Existen numerosas herramientas y APIs relacionadas:
  - Editores de datos (JOSM, iD).
  - Servidores de mapas y de *tiles*.
  - Librerías de visualización (Leaflet, OpenLayers).
  - Servicios de enrutamiento y geocodificación.
- Desde el punto de vista de desarrollo front-end, una de las opciones más populares es la librería Leaflet.

Mapa por *tiles* (mosaicos)

La representación de mapas se basa en *tiles* (azulejos, baldosas).

- Un *tile* es un mapa de bits con un fragmento de mapa.
- Tamaño típico: 256x256 píxeles (64x64 para móviles, 512x512 para alta resolución).
- El mapa completo se construye combinando miles de *tiles* en diferentes niveles de zoom.
- Cada *tile* se identifica por las coordenadas (`z`, `x`, `y`).

Servidores de tiles y licenciamiento

- Los datos de OpenStreetMap son libres (y gratuitos).
- El servidor de *tiles* no tiene por qué ser gratuito:
  - Puede contratarse un proveedor comercial.
  - Es posible desplegar un servidor propio.
- Para usos experimentales y con poca carga de datos, se pueden usar servidores proporcionados por la comunidad, como:
  - `https://tile.osm.org`
- Es importante respetar las políticas de uso y las limitaciones de tráfico (*rate limiting*).

Leaflet: librería ligera para mapas

- Leaflet (`http://leafletjs.com`) es una librería JavaScript ligera para mapas interactivos.
- Características:
  - Fácil de usar y bien documentada.
  - Orientada a dispositivos móviles.
  - Arquitectura basada en capas y marcadores.
- Permite integrar mapas OSM (y otros proveedores) en una página HTML con poco código.

Inclusión de Leaflet desde un CDN

Para representar un mapa OpenStreetMap en una página HTML usando Leaflet:

- Incluimos la librería desde el CDN:

```
<script src="https://unpkg.com/leaflet@1.2.0/dist/leaflet.js"></script>
<link rel="stylesheet"
 href="https://unpkg.com/leaflet@1.2.0/dist/leaflet.css"/>
```

- Esto crea un objeto global llamado `L`.
- Es imprescindible incluir tanto el JavaScript como la hoja de estilos.

Contenedor para el mapa en HTML

- En nuestro HTML definimos un `<div>` donde irá el mapa.
- Es necesario que tenga definido un atributo CSS `height` con la altura:

```
<style>
  #id_mapa {
    height: 400px;
  }
</style>

<body>
  <div id="id_mapa"></div>
</body>
```

- Sin una altura explícita, el mapa no será visible, ya que el contenedor tendrá altura cero.

Inicialización del mapa Leaflet

- Creamos un objeto de tipo `L.map()`, pasando las coordenadas del centro del mapa deseado y el zoom inicial.
- El zoom 0 corresponde a un mapa de toda la Tierra; valores mayores muestran más detalle.
- A continuación, invocamos el método `L.tileLayer()`, con la URL del mapa en el servidor de *tiles* y el mensaje de atribución.

Código básico de inicialización con Leaflet

```
$(document).ready(function() {
  let latitud  = 40.417;   // coordenada y
  let longitud = -3.703;   // coordenada x
  let zoom     = 16;

  let mi_mapa = L.map('id_mapa').setView([latitud, longitud], zoom);

  L.tileLayer('https://{s}.tile.osm.org/{z}/{x}/{y}.png', {
    attribution:
      '&copy; <a href="https://osm.org/copyright">OpenStreetMap</a>'
  }).addTo(mi_mapa);
});
```

- Este código crea un mapa centrado en unas coordenadas y con un nivel de zoom dado.

Uso de múltiples capas de *tiles*

- Es posible definir varias capas base (por ejemplo, mapa estándar y mapa en blanco y negro).

```
let capa_osm = L.tileLayer('https://{s}.tile.osm.org/{z}/{x}/{y}.png', {
  attribution: '&copy; OpenStreetMap contributors'
});

let capa_toner = L.tileLayer(
  'https://{s}.tile.stamen.com/toner/{z}/{x}/{y}.png', {
  attribution: '&copy; Stamen Design, OpenStreetMap'
});

let mi_mapa = L.map('id_mapa', {
  center: [40.417, -3.703],
  zoom: 14,
  layers: [capa_osm]
});

L.control.layers({
  "OSM estándar": capa_osm,
  "Toner": capa_toner
}).addTo(mi_mapa);
```

Marcadores y mensajes *popup*

- Podemos añadir marcadores invocando el método: `L.marker(<COORDENADAS>).addTo(<OBJETO MAPA>)`
- Podemos añadir mensajes emergentes de tipo *popup* con el método `.bindPopup`.

```
let coord_labIII = [40.417, -3.703];

let mi_marcador = L.marker(coord_labIII).addTo(mi_mapa);
mi_marcador.bindPopup("Laboratorios III").openPopup();
```

- El *popup* se abre automáticamente con `openPopup()`; también puede abrirse en respuesta a eventos.

Marcadores personalizados

- Leaflet permite definir iconos personalizados para los marcadores:

```
let icono_personalizado = L.icon({
  iconUrl: 'img/icono.png',
  iconSize:     [32, 32],
  iconAnchor:   [16, 32],
  popupAnchor:  [0, -32]
});

L.marker([40.417, -3.703], {icon: icono_personalizado})
 .addTo(mi_mapa)
 .bindPopup("Marcador con icono personalizado");
```

- Esto facilita señalar puntos de interés específicos (paradas, edificios, etc.).

Eventos de interacción en mapas Leaflet

- Leaflet ofrece numerosos eventos:
  - `click`, `dblclick`, `mousemove`.
  - `zoomend`, `moveend`, `dragend`.

```
mi_mapa.on('click', function(e) {
  let latlng = e.latlng;
  L.marker(latlng).addTo(mi_mapa)
   .bindPopup("Nuevo marcador en: " + latlng.toString())
   .openPopup();
});
```

- Estos eventos permiten construir interfaces ricas, como selección de ubicaciones por parte del usuario.

Integración de datos GeoJSON

- Leaflet soporta directamente el formato GeoJSON, estándar de facto para datos geográficos en la Web.

```
let datos_geojson = {
  "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [[[...]]]
  },
  "properties": {
    "nombre": "Zona de estudio"
  }
};

L.geoJSON(datos_geojson, {
  onEachFeature: function(feature, layer) {
    layer.bindPopup(feature.properties.nombre);
  }
}).addTo(mi_mapa);
```

- Esto facilita la visualización de capas vectoriales complejas.

Geolocalización del usuario

- Leaflet puede integrarse con la API de geolocalización del navegador.

```
mi_mapa.locate({setView: true, maxZoom: 16});

mi_mapa.on('locationfound', function(e) {
  L.marker(e.latlng).addTo(mi_mapa)
   .bindPopup("Estás aquí").openPopup();
});

mi_mapa.on('locationerror', function(e) {
  alert("No ha sido posible obtener la ubicación: " + e.message);
});
```

- Permite centrar el mapa en la posición actual del usuario, mejorando la usabilidad.

Rendimiento y buenas prácticas en mapas web

- Limitar el número de marcadores simultáneos (por ejemplo, técnicas de *clustering*).
- Cargar datos bajo demanda según el nivel de zoom y el área visible.
- Utilizar servidores de *tiles* apropiados y respetar sus limitaciones.
- Reducir el peso de los recursos estáticos (iconos, hojas de estilo, scripts).
- Probar el comportamiento en dispositivos móviles y en diferentes navegadores.

Resumen y líneas de trabajo

- Hemos estudiado dos ejemplos de integración de servicios externos:
  - Reproductores de vídeo con la API IFrame de YouTube.
  - Mapas interactivos usando datos de OpenStreetMap y la librería Leaflet.
- Ambos casos ilustran:
  - El uso de APIs JavaScript para enriquecer aplicaciones web.
  - La importancia de entender los eventos, métodos y parámetros de configuración.
- Posibles extensiones:
  - Sincronizar vídeo y mapa en una misma página.
  - Integrar datos propios (p.ej., rutas, sensores) sobre mapas OSM.
  - Analizar cuestiones de rendimiento, accesibilidad y privacidad en mayor profundidad.

Vimeo

- Vimeo es una plataforma de alojamiento de vídeo orientada a contenidos de alta calidad y uso profesional.
- Al igual que YouTube, ofrece:
  - Reproductores embebibles mediante `<iframe>`.
  - Una API JavaScript basada en *postMessage*.
- Ventajas habituales:
  - Mayor control sobre la marca (*branding*) del reproductor.
  - Opciones avanzadas de privacidad y distribución.
- La integración básica es conceptualmente similar al caso de YouTube.

Incrustar un vídeo de Vimeo (HTML básico)

- El código mínimo de incrustación usa un `<iframe>` proporcionado por Vimeo.

```
<div style="padding:56.25% 0 0 0;position:relative;">
  <iframe
    src="https://player.vimeo.com/video/123456789"
    style="position:absolute;top:0;left:0;width:100%;height:100%;"
    frameborder="0"
    allow="autoplay; fullscreen; picture-in-picture"
    allowfullscreen>
  </iframe>
</div>
<script src="https://player.vimeo.com/api/player.js"></script>
```

- La inclusión del script `player.js` permite controlar el reproductor desde JavaScript.

API JavaScript de Vimeo: objeto `Player`

- La librería `player.js` define el constructor `Vimeo.Player`.
- Permite:
  - Reproducir, pausar, saltar a una posición.
  - Escuchar eventos de reproducción (play, pause, ended, etc.).

```
let iframe = document.querySelector('iframe');
let player = new Vimeo.Player(iframe);

player.on('play', function() {
  console.log('El vídeo ha comenzado a reproducirse');
});

player.getDuration().then(function(segundos) {
  console.log('Duración del vídeo:', segundos);
});
```

Control programático de Vimeo

- Ejemplo de integración con botones de la interfaz:

```
let botonPlay = document.getElementById('btn_play');
let botonPause = document.getElementById('btn_pause');

botonPlay.addEventListener('click', () => {
  player.play();
});

botonPause.addEventListener('click', () => {
  player.pause();
});

player.on('ended', () => {
  alert('El vídeo ha finalizado');
});
```

- La API es promisificada: muchos métodos devuelven `Promise`.

SoundCloud

- SoundCloud es una plataforma de distribución de audio, centrada en música y podcasts.
- Proporciona widgets embebibles para reproducir:
  - Pistas individuales.
  - Listas de reproducción.
  - Perfiles de usuario.
- Dispone de una *Widget API* JavaScript para controlar el reproductor.
- Resulta útil en aplicaciones web donde se quiera complementar contenido con audio.

Incrustación básica de SoundCloud

```
<iframe id="sc_player"
  width="100%" height="166"
  scrolling="no" frameborder="no"
  src="https://w.soundcloud.com/player/\
?url=https%3A//api.soundcloud.com/tracks/123456789&color=%23ff5500">
</iframe>

<script src="https://w.soundcloud.com/player/api.js"></script>
```

- La URL incluye parámetros de configuración (color, auto_play, etc.).
- El script `api.js` expone el objeto `SC.Widget`.

API del widget de SoundCloud

- La API permite crear una instancia de widget asociada a un `iframe`:

```
let iframe = document.getElementById('sc_player');
let widget = SC.Widget(iframe);

widget.bind(SC.Widget.Events.READY, function() {
  console.log('Reproductor listo');
});

widget.bind(SC.Widget.Events.PLAY, function() {
  console.log('Comienza la reproducción');
});
```

- Se pueden registrar *callbacks* para eventos como PLAY, PAUSE, FINISH, etc.

Control de reproducción en SoundCloud

```
document.getElementById('btn_sc_play').addEventListener('click', () => {
  widget.play();
});

document.getElementById('btn_sc_pause').addEventListener('click', () => {
  widget.pause();
});

document.getElementById('btn_sc_pos').addEventListener('click', () => {
  widget.getPosition(function(ms) {
    console.log('Posición actual (ms):', ms);
  });
});
```

- El API mezcla estilo de *callbacks* con el patrón de eventos.
- Permite crear interfaces personalizadas alrededor del reproductor estándar.

Twitter: widgets embebibles

- Twitter proporciona widgets para:
  - Incrustar *tweets* individuales.
  - Mostrar líneas de tiempo de usuario o de listas.
  - Botones de compartir, seguir, etc.
- La integración se realiza cargando el script oficial y usando atributos `data-*`.
- Es un ejemplo de API centrada en la representación de contenido social.

Incrustar un tweet individual

```
<blockquote class="twitter-tweet">
  <p lang="es" dir="ltr">
    Texto del tweet...
  </p>
  &mdash; Usuario (@usuario)
  <a href="https://twitter.com/usuario/status/1234567890">
    Fecha
  </a>
</blockquote>

<script async src="https://platform.twitter.com/widgets.js"
        charset="utf-8"></script>
```

- El script busca elementos con la clase `twitter-tweet` y los transforma en un widget.
- La URL del *tweet* determina el contenido mostrado.

Línea de tiempo de usuario

```
<a class="twitter-timeline"
   data-height="400"
   href="https://twitter.com/usuario">
  Tweets by usuario
</a>

<script async src="https://platform.twitter.com/widgets.js"
        charset="utf-8"></script>
```

- El widget genera una caja con desplazamiento vertical para los tweets recientes.
- Se pueden ajustar parámetros como altura, tema (claro/oscuro), idioma, etc.

Privacidad y rendimiento con widgets sociales

- La integración de widgets de redes sociales tiene implicaciones:
  - Carga de scripts de terceros en todas las páginas.
  - Posible seguimiento de usuarios (*tracking*).
- Buenas prácticas:
  - Carga diferida (lazy load) de los widgets.
  - Mostrar un *placeholder* y pedir consentimiento antes de cargar contenido embebido.
  - Analizar el impacto en tiempos de carga y en la experiencia de usuario.

Instagram: contenido visual embebible

- Instagram permite incrustar:
  - Publicaciones individuales (imagen o vídeo).
  - Reels, según la política vigente.
- La incrustación se basa en un bloque `<blockquote>` y un script de transformación.
- Es adecuada para enriquecer páginas con contenido visual procedente de cuentas oficiales.

Incrustar una publicación de Instagram

```
<blockquote class="instagram-media"
  data-instgrm-permalink="https://www.instagram.com/p/XXXXXXXXX/"
  data-instgrm-version="14">
</blockquote>

<script async src="//www.instagram.com/embed.js"></script>
```

- El *script* procesa los elementos con clase `instagram-media`.
- La URL en `data-instgrm-permalink` apunta a la publicación a mostrar.

Aspectos de maquetación y responsive

- Los widgets de Instagram se adaptan al ancho del contenedor.
- Conviene:
  - Colocarlos en contenedores con ancho definido (por ejemplo, columnas).
  - Asegurar que no rompen la maquetación en pantallas pequeñas.
- La carga asíncrona del script ayuda a mantener tiempos de carga razonables.

Limitaciones y API de Instagram

- La API de Instagram está más orientada a acceso vía servidor (Graph API).
- Para el cliente web:
  - La personalización del widget es relativamente limitada.
  - Es necesario respetar términos de uso y políticas de privacidad.
- Aun así, la incrustación de publicaciones es suficiente en muchas aplicaciones de presentación de contenido.

Google Maps JavaScript API

- Google Maps ofrece una API JavaScript para incrustar y personalizar mapas interactivos.
- Comparte objetivos con Leaflet + OSM, pero:
  - Utiliza datos propietarios de Google.
  - Requiere una clave de API y está sujeta a cuotas de uso.
- Permite:
  - Dibujar marcadores, polígonos, rutas.
  - Integrar servicios de geocodificación y direcciones.

Carga del script de Google Maps

```
<script async
  src="https://maps.googleapis.com/maps/api/js?key=TU_API_KEY&callback=initMap">
</script>
```

- `TU_API_KEY` debe sustituirse por una clave válida.
- El parámetro `callback` indica la función a invocar cuando la API esté lista.
- La carga asíncrona evita bloquear el renderizado inicial de la página.

Inicialización básica de un mapa de Google

```
<div id="mapa_google" style="height:400px;"></div>
```

```
function initMap() {
  let centro = {lat: 40.417, lng: -3.703};
  let mapa = new google.maps.Map(
    document.getElementById('mapa_google'),
    {
      zoom: 15,
      center: centro
    }
  );

  new google.maps.Marker({
    position: centro,
    map: mapa,
    title: "Marcador inicial"
  });
}
```

- La API se basa en la creación de objetos `google.maps.Map` y asociados.

Costes, cuotas y buenas prácticas

- Google Maps JavaScript API tiene un modelo de precios basado en uso.
- Es necesario:
  - Configurar restricciones sobre la clave de API (dominios permitidos).
  - Monitorizar el consumo para evitar costes inesperados.
- En el diseño de la aplicación:
  - Cargar la API sólo en las páginas que realmente la necesitan.
  - Considerar alternativas libres (OSM + Leaflet) cuando sea posible.