

Tema 4:

Apps web en

dispositivos móviles

Parte I: Introducción y APIs

Javascript

Tema 4: Intro a las apps web en dispositivos móviles

4.1.

Tipos de aplicaciones en móviles

Aplicaciones nativas

Desarrolladas con el SDK nativo de la plataforma y en los lenguajes soportados por ellas (*Kotlin, Java en Android, Swift, Obj-C en iOS*)

- Ventajas:
 - “Exprimen” al máximo el hardware
 - “Look & Feel” de la plataforma
- Inconvenientes
 - Nula portabilidad
 - Desarrollo costoso, especializado



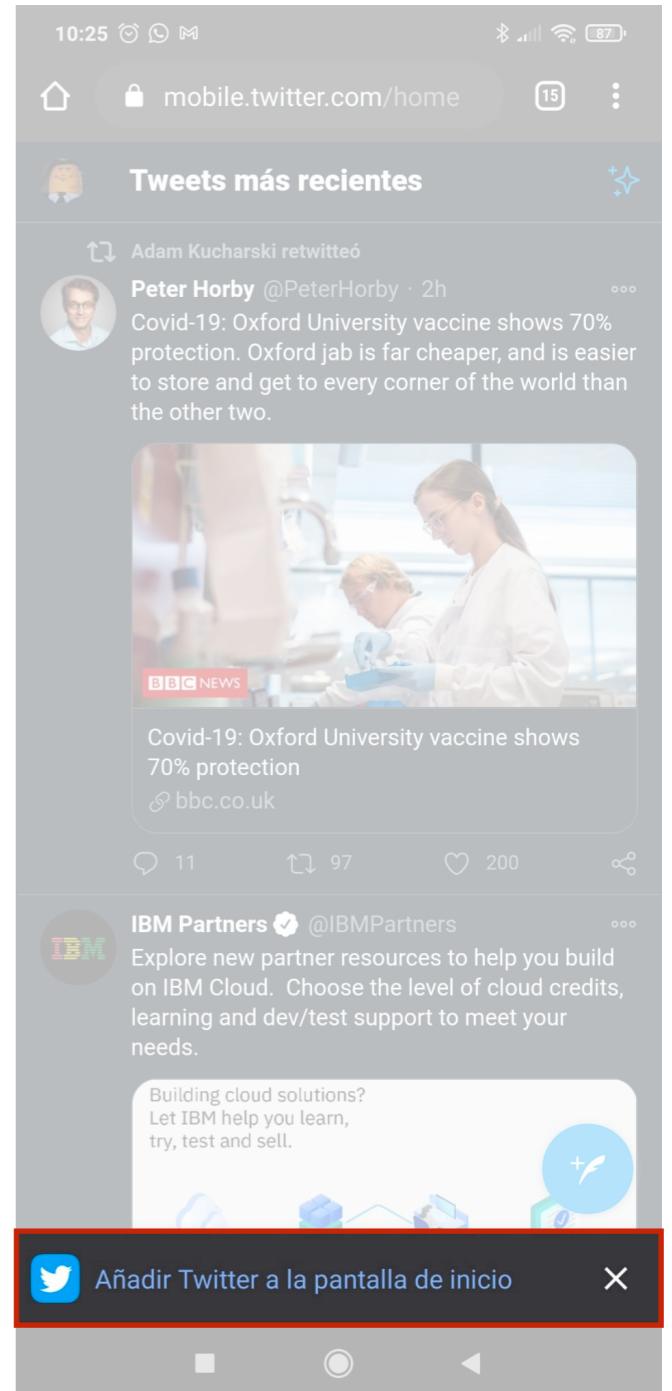
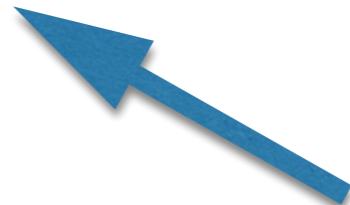
Aplicaciones web

Código en **Javascript**, interfaz en **HTML/CSS**

- Ventajas:
 - Portabilidad
 - Se puede acceder a muchos APIs hace años reservados a apps nativas (cámara, micro, agenda, geolocalización, ...)
 - Baja “barrera de entrada” para el desarrollador: “solo” conocer HTML/CSS/JS
 - Se puede escapar al control de las tiendas de apps oficiales
- Inconvenientes:
 - Rendimiento no aceptable para algunos tipos de aplicaciones (p.ej. juegos 3D)
 - No tienen el “look & feel” de la plataforma móvil
 - No se pueden vender en las tiendas de apps oficiales, ya que son sitios web
 - Algunos APIs no accesibles, aunque cada vez menos. Fugu API tracker, proyecto siguiendo los gaps entre APIs nativos/APIs Web <https://web.dev/fugu-status/>

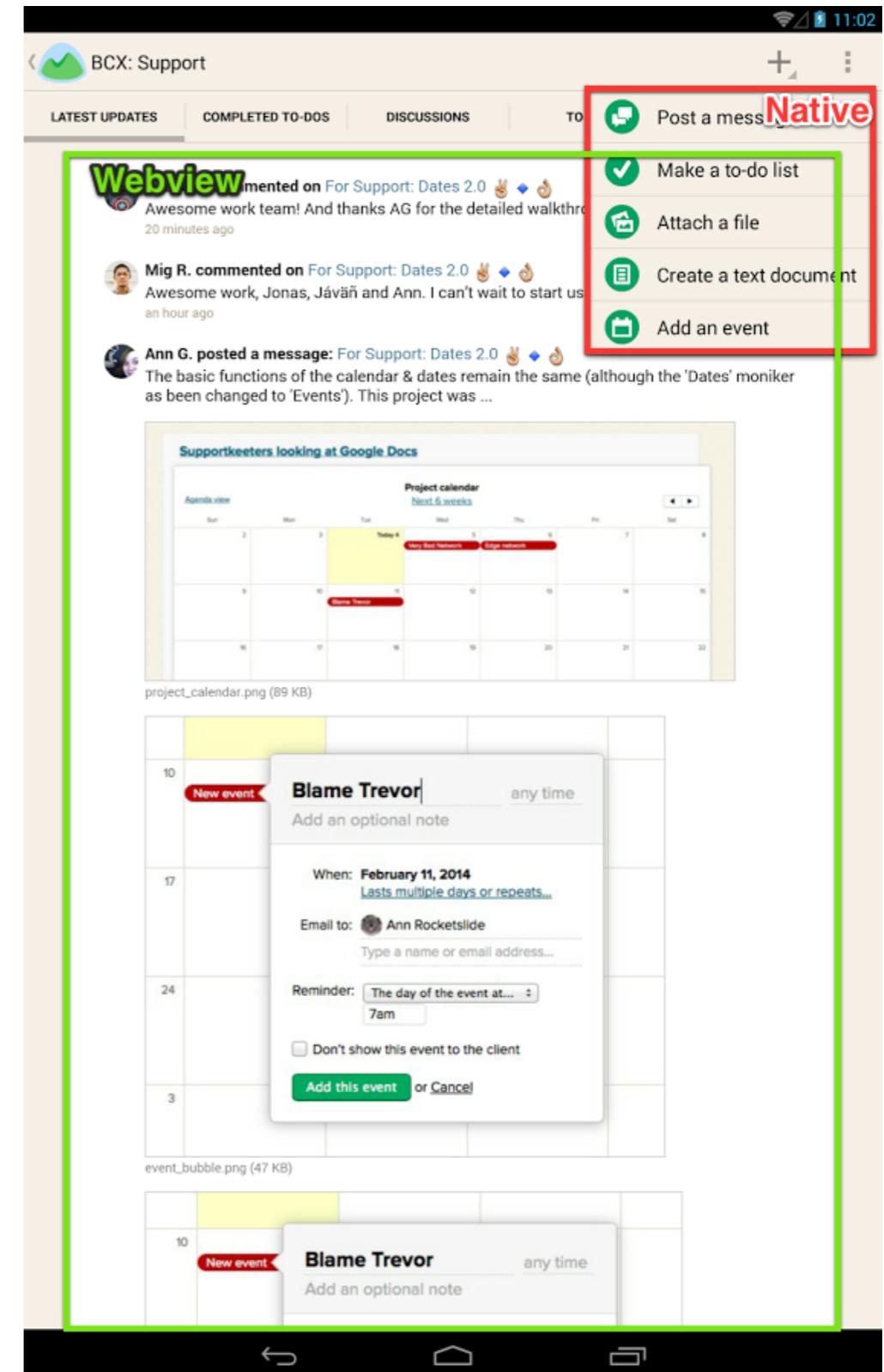
Ejemplo de apps web: PWAs

- **Progressive Web Apps:** son aplicaciones web pero dan la impresión de ser nativas
- No se instalan desde la tienda de apps sino desde la web, cuando la visitas por primera vez
- Las veremos con más detalle la semana que viene



Aplicaciones híbridas

- La aplicación web está contenida en un “envoltorio” que es una app nativa
- Ventajas:
 - Portabilidad entre plataformas
 - Al ser “por fuera” una app clásica, se puede distribuir sin pegas en las stores
 - El “envoltorio” suele dar acceso a APIs nativos desde Javascript
- Inconvenientes:
 - “Look & Feel” no nativo
 - Menor rendimiento (relevante o no según el tipo de app)



Hybrid sweet spot: Native navigation, web content, del blog de @dhh

Apps híbridas con Capacitor

- <https://capacitorjs.com/>. La **plataforma híbrida** más conocida en la actualidad
 - La sucesora de Apache Cordova
- Dispone de un conjunto de **plugins que permiten acceder a código nativo** (Swift/ObjC/Kotlin/Java) y por tanto a los APIs nativos del dispositivo
- Hay bastantes *plugins* y también podemos crear otros propios

▼ Official Plugins
Introduction
Accessibility
App
Background Task
Browser
Camera
Clipboard
Console
Device
Filesystem
Geolocation
Haptics
Keyboard
Local Notifications
Modals
Motion
Network
Permissions
Push Notifications
Share
Splash Screen
Status Bar
Storage
Toast





Apps “casi” nativas

- El código original se escribe en un lenguaje distinto al nativo pero luego **se compila todo a código/componentes nativos**
- Más que la portabilidad entre plataformas (que la hay), su principal ventaja es la “portabilidad” de habilidades de desarrollo. Hace **accesible el desarrollo “nativo” a desarrolladores que conocen otros lenguajes**
- Ejemplos
 - Desarrollo en JS: React native (React), NativeScript (Angular, Vue, Typescript, vanilla JS)
 - Desarrollo en C#: Xamarin



Tema 4: Intro a las apps web en dispositivos móviles

4.2.

Principios generales de diseño de apps web móviles

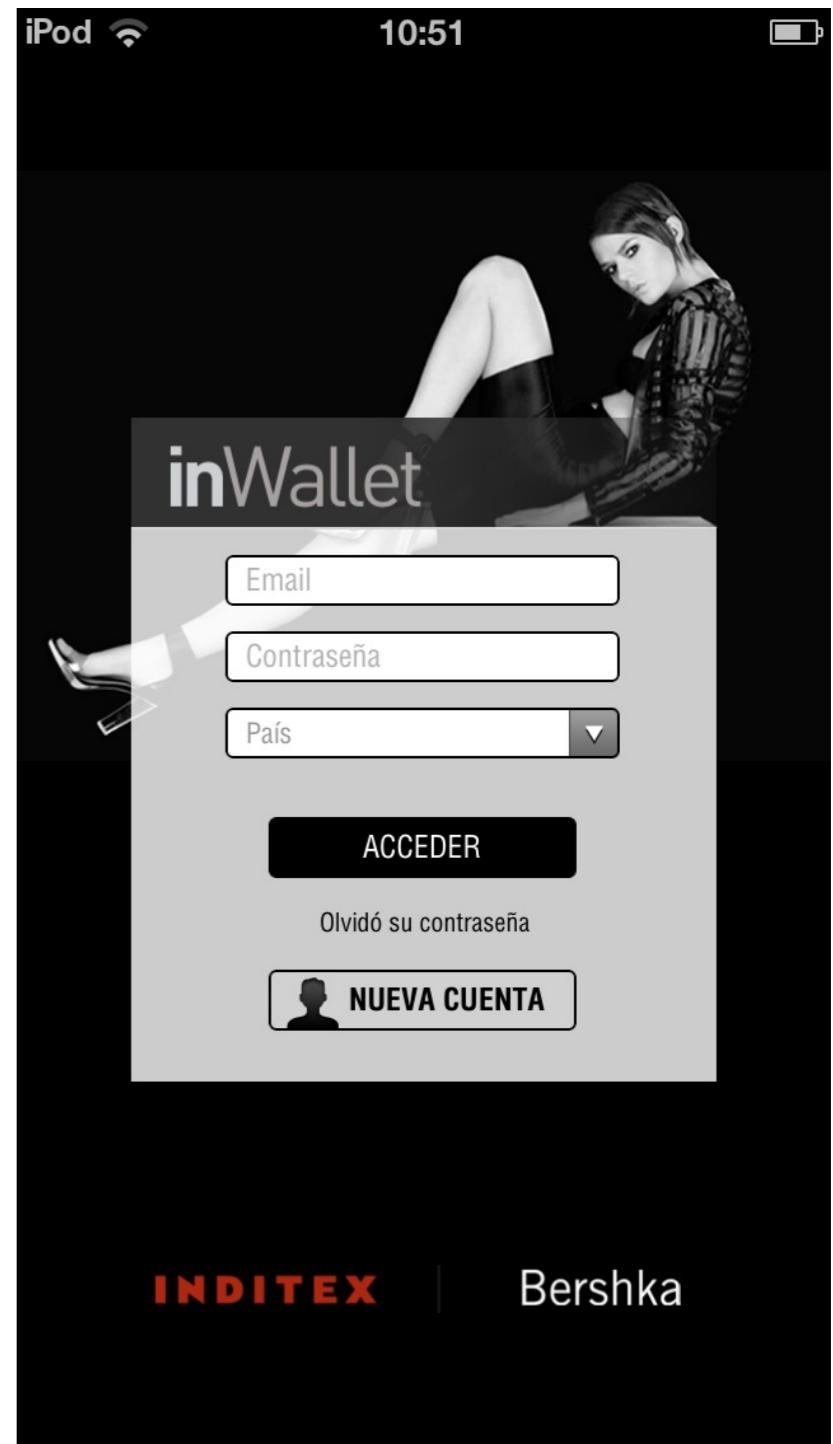
La pantalla

Aunque la resolución está a la par del escritorio, **no debemos colocar demasiada información ya que la pantalla es muy pequeña**



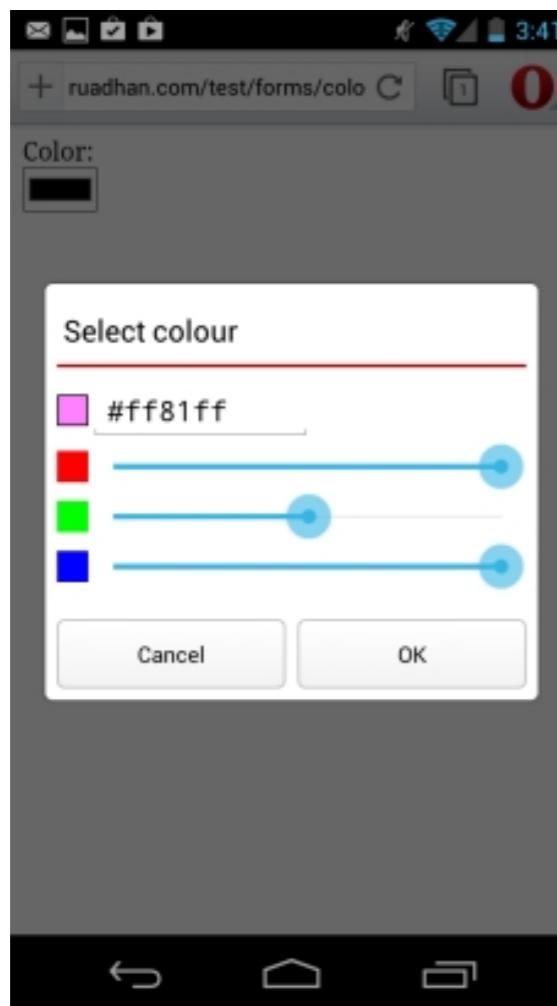
La entrada de datos es tediosa

- Introducir datos en un móvil es incómodo y tedioso.
 - La aplicación debe reducir esta necesidad al mínimo imprescindible
 - Ejemplo: recordar login y password, recordar preferencias,...
- Los controles táctiles deben ser grandes, para poder ser pulsados cómodamente con el dedo.

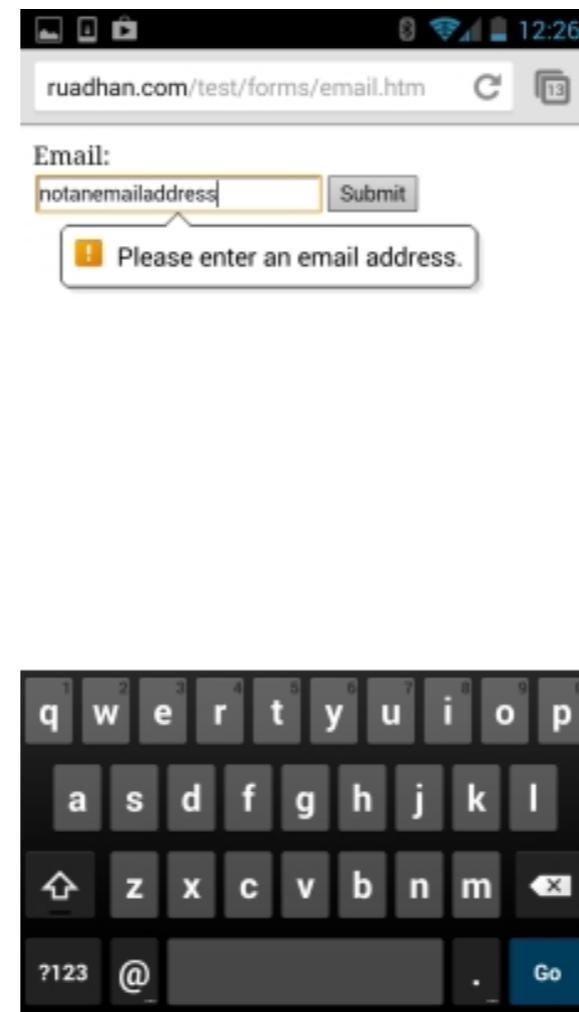


Facilitar la entrada de datos

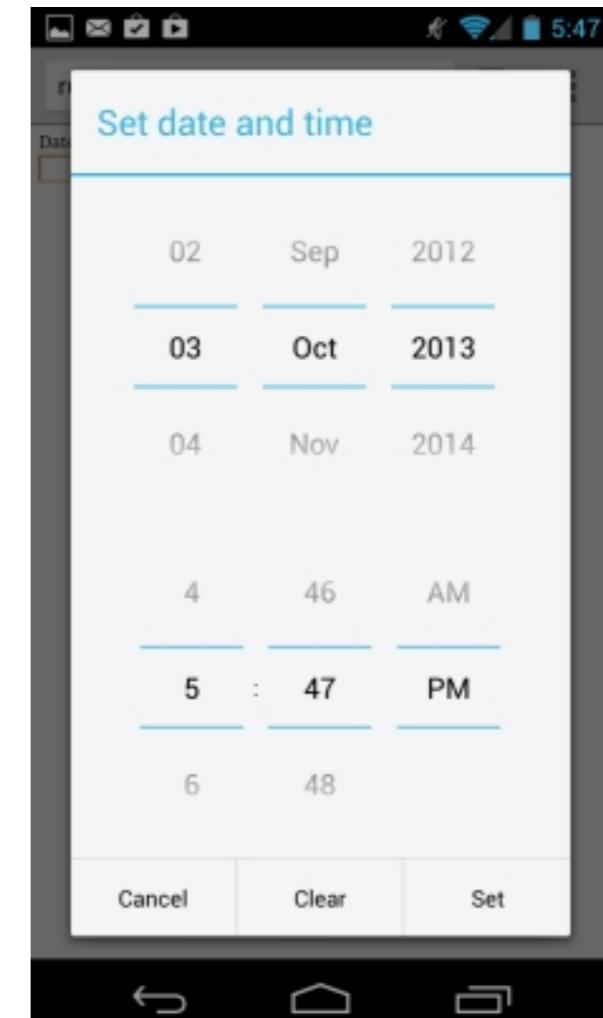
- Usar <input> del type más apropiado (number, date, email, color,...)



<input type="color"/>



<input type="email"/>

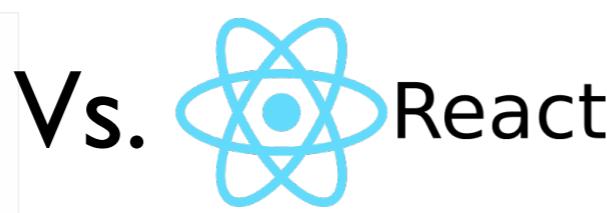


<input type="datetime-local"/>

<https://mobiforge.com/design-development/html5-mobile-web-forms-and-input-types>

Usar los menores recursos posibles

- Los móviles tienen restricciones de
 - Batería
 - Latencia de red
 - Memoria y capacidad de procesamiento
- Consejos habituales:
 - Reducir al máximo el número y tamaño de las dependencias



React vs. Preact. Preact has the same API as **React**, but is a faster **3k alternative to React's ~45k**. Preact does this by stripping away a lot of the “extra” stuff in **React** mainly by pulling out **React's** events implementation and using the browser's native event listener instead.

[React vs. Preact - Front End Development - Confluence - Atlassian](#)

<https://openedx.atlassian.net/wiki/spaces/FEDX/pages/162294202/React+vs.+Preact>

- No malgastar recursos hardware (p.ej. precisión en la localización, como luego veremos)

Más consejos

- En <https://developers.google.com/web/fundamentals/design-and-ux/principles?hl=es>

The screenshot shows a web browser displaying the URL <https://developers.google.com/web/fundamentals/design-and-ux/principles?hl=es>. The page is titled "¿Qué hace que un sitio para dispositivo móvil sea bueno?" (What makes a good mobile site?). The left sidebar has a navigation menu with sections like "Overview", "Architectural Patterns", "Design & User Experience" (which is expanded to show "Designing Great User Experiences", "Basics of User Experiences", and "What Makes a Good Mobile Site?", the latter of which is highlighted), "Integration & Engagement", "Media & VR", "Performance", and "Security". The main content area shows a list of tips under the heading "Contenido": "Navegación por el sitio y página principal", "Mantén las llamadas en acción en el centro y frente", "Haz menús cortos y dulces", and "Haz que volver a la página principal sea sencillo". Below this, it says "...". A profile picture of Jenny Gove is shown with the text "By [Jenny Gove](#)". A bio follows: "Jenny Gove is a UX Research Lead at Google, where she conducts research on smartphone experiences. She obtained her PhD from the University of Southampton, UK." Further down, it says "Google y AnswerLab realizaron un [estudio de investigación](#) para responder esta pregunta." A box contains the text: "Los usuarios de dispositivos móviles están orientados al objetivo. Esperan poder obtener lo que necesitan, inmediatamente, y bajo sus propias condiciones." At the bottom, it says "El estudio se llevó a cabo durante sesiones de usabilidad en persona de 119 horas, con participantes en los EE. UU."

Tema 4: Intro a las apps web en dispositivos móviles

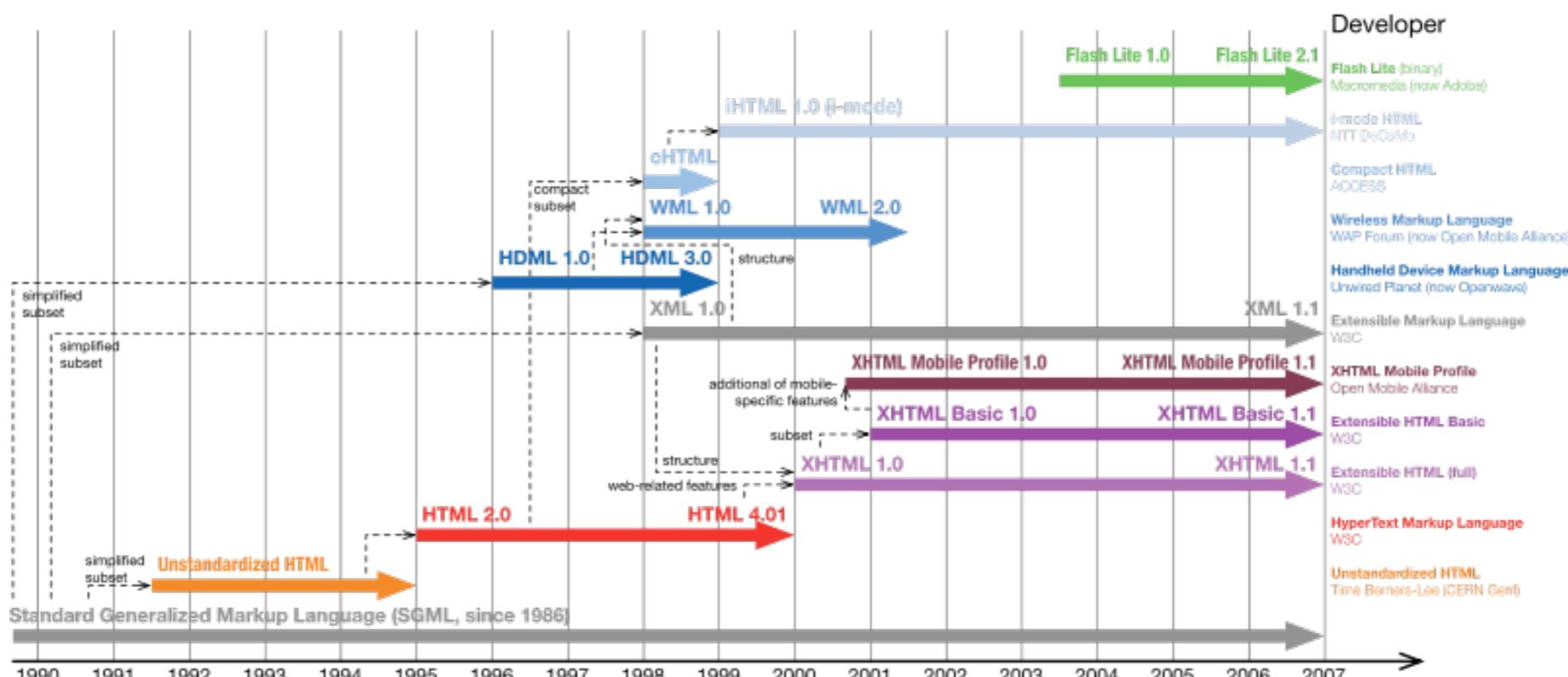
4.3.

HTML y CSS para apps web móviles

Lenguajes de marcado

- Históricamente hay diferentes **variantes de HTML y CSS** específicas para móviles
 - XHTML Basic (subconjunto de XHTML) y CSS MP (del W3C)
 - XHTML MP (Mobile Profile, ampliación de XHTML Basic) y WAP CSS (de la OMA-Open Mobile Alliance)
- En la actualidad se suele usar **simplemente HTML5** igual que en el escritorio

https://en.wikipedia.org/wiki/XHTML_Mobile_Profile

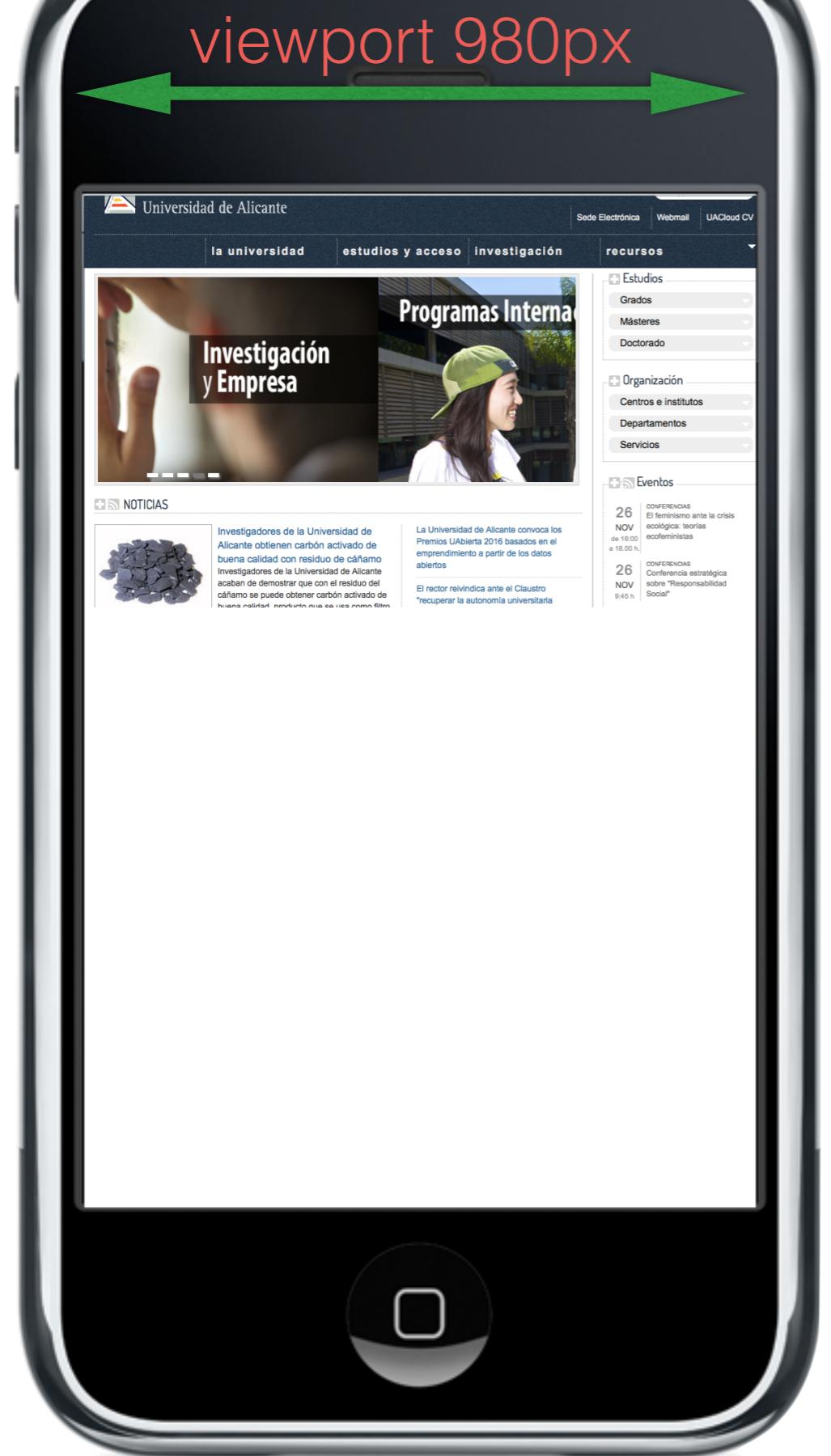


320px



990px

A screenshot of a desktop browser displaying the University of Alicante website. At the top right, there are language links: 'ESPAÑOL', 'VALENCIÀ', and 'ENGLISH'. Below them is a search bar with the placeholder 'Buscar web | personas'. To the right of the search bar are links for 'Sede Electrónica', 'Webmail', and 'UACloud CV'. The main content area features a large image of a smiling person wearing a green cap, with the text 'Programas Internacionales' overlaid. Below this, there are three sections: 'Estudios' (Grados, Másteres, Doctorado), 'Organización' (Centros e institutos, Departamentos, Servicios), and 'Eventos' (two conference entries: 'El feminismo ante la crisis ecológica: teorías ecofeministas' on November 26 and 'Conferencia estratégica sobre "Responsabilidad Social"' on November 26). A green double-headed arrow at the bottom spans the width of the desktop browser, labeled '990px'.



El viewport

- El dispositivo simula que tiene una resolución distinta a la real
 - El iPhone original tenía 320px de ancho, pero escalaba las páginas tomando como referencia 980px
 - Esto se hizo para que las webs aparecieran como en el escritorio (y no “cortadas” en vertical)
 - Por motivos históricos/prácticos se ha conservado la idea



<http://developer.apple.com/library/ios/#documentation/AppleApplications/Reference/SafariWebContent/UsingtheViewport/UsingtheViewport.html>

Un pixel no siempre es un pixel

- “Tipos” de pixels:
 - Device pixels: los físicos
 - CSS pixels: los que se usan en las reglas CSS (`screen.width`, `screen.height` reportan CSS pixels, no físicos)
- P.ej. el iPhone 12 Pro Max reporta que tiene 428 px de ancho en vertical (CSS pixels) cuando en realidad tiene 1284 físicos (<https://www.mydevice.io/#compare-devices>)
- `window.devicePixelRatio`: ¿Cuántos píxeles reales es 1 pixel CSS? (en el ejemplo anterior, 3)
- ¿Para qué se pueden usar los pixels “extra”?
 - Para mostrar texto más nítido
 - Para servir imágenes de “alta resolución” vs. “resolución estándar”: tag `<picture>`, atributo `srcset` del tag `` (<https://developers.google.com/web/fundamentals/design-and-ux/responsive/images?hl=es>)

Controlar el viewport

- Etiqueta `<meta name="viewport">` en la cabecera
- 2 “modos de uso”:
 - Para webs con “ancho fijo”, especificar el tamaño

```
<head>
  <meta name="viewport" content="width=640">
</head>
```

- Para webs “fluidas”, indicar que se use el ancho del dispositivo

```
<head>
  <meta name="viewport" content="width=device-width">
</head>
```

Explicación más detallada: <https://web.dev/responsive-web-design-basics/>

<https://without-vp-meta.glitch.me/>

Ejemplo sin `<meta>`

<https://with-vp-meta.glitch.me/>

Ejemplo con `<meta>`

Responsive web

The mobile responsive web interface for UACloud features a header bar with the UACloud logo, a search icon, a user profile icon, and a notification bell icon with a red badge showing '8'. Below the header, a banner encourages users to download the app for Android or iOS. The main content area is divided into sections: 'Últimas' (Last) and 'Últimas' (Recent). The 'Últimas' section contains cards for 'Mi alumnado', 'Tutorías', 'Moodle UA', and 'Solicitudes UA'. The 'Últimas' section contains cards for 'Programa Redes-ICE', 'Docencia dual', 'Programa acción tutorial', 'Solicitudes UA', 'Partes de Mantenimiento', 'Pago de recibos', and 'AlumniUA'. Each card includes a small icon and a timestamp.

COLOMINA PARDO, OTTO

8

Ven a la app de la UA
Recuerda que puedes descargarla para [Android](#) o [iOS](#).

Mi alumnado
21/11/2020
20:50:50

Tutorías
21/11/2020
20:50:37

Moodle UA
20/11/2020
16:11:05

Solicitudes UA
20/11/2020
13:59:11

Últimas

Programa Redes-ICE

Docencia dual

Programa acción tutorial

Solicitudes UA

Partes de Mantenimiento

Pago de recibos

AlumniUA

The mobile responsive web interface for UACloud features a header bar with the UACloud logo, a user profile icon, and a notification bell icon with a red badge showing '8'. Below the header, a banner shows recent activity: 'Mi alumnado' (21/11/2020, 20:50:50), 'Tutorías' (21/11/2020, 20:50:37), and 'Moodle UA' (20/11/2020, 16:11:05). The main content area is divided into sections: 'Últimas' (Last) and 'Últimas' (Recent). The 'Últimas' section contains cards for 'Solicitudes UA' (20/11/2020, 13:59:11), 'Programa Redes-ICE', 'Docencia dual', and 'Programa acción tutorial'. The 'Últimas' section contains cards for 'Solicitudes UA', 'Partes de Mantenimiento', and 'Pago de recibos'. Each card includes a small icon and a timestamp.

Mi alumnado
21/11/2020
20:50:50

Tutorías
21/11/2020
20:50:37

Moodle UA
20/11/2020
16:11:05

Solicitudes UA
20/11/2020
13:59:11

Últimas

Programa Redes-ICE

Docencia dual

Programa acción tutorial

Solicitudes UA
20/11/2020
13:59:11

Partes de Mantenimiento

Pago de recibos

Media queries

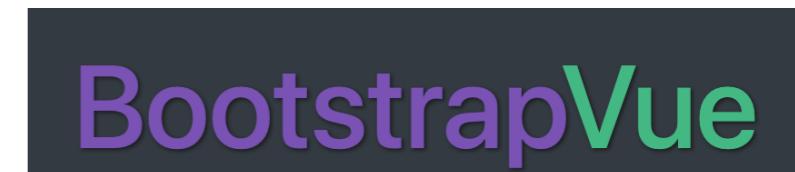
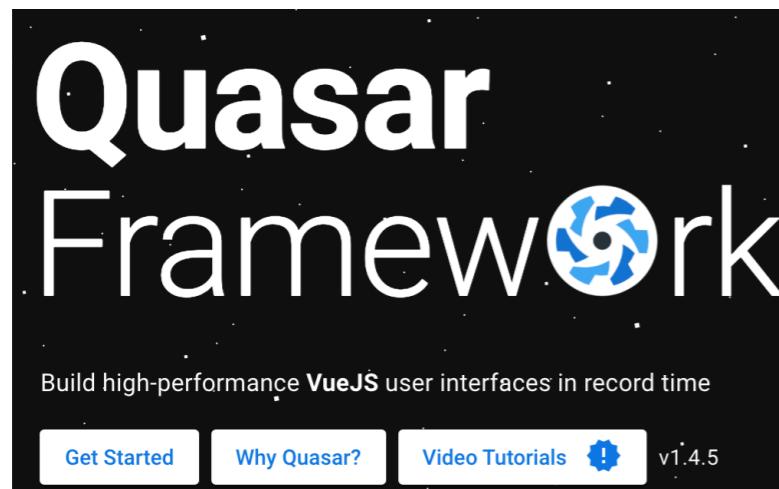
- Reglas CSS aplicables solo a ciertos tamaños de pantalla

```
<!-- vincular con una u otra hoja de estilo dependiendo de la resolución horizontal -->
<link type="text/css" rel="stylesheet" media="screen and (max-device-width:480px)"
      href="smartphone.css" />
<link type="text/css" rel="stylesheet" media="screen and (min-device-width:481px)"
      href="desktop.css" />

<!-- también se puede poner en el CSS “empotrado” en el HTML -->
<style>
    @media screen and (max-device-width:480px) {
        body {background-color: red;}
    }
</style>
```

Componentes para Frameworks JS

- Con la popularidad de los frameworks JS al estilo React, Vue, Angular,... se han extendido las bibliotecas de componentes de UI especialmente adaptados a móvil para estos frameworks



Tema 4: Intro a las apps web en dispositivos móviles

4.4.

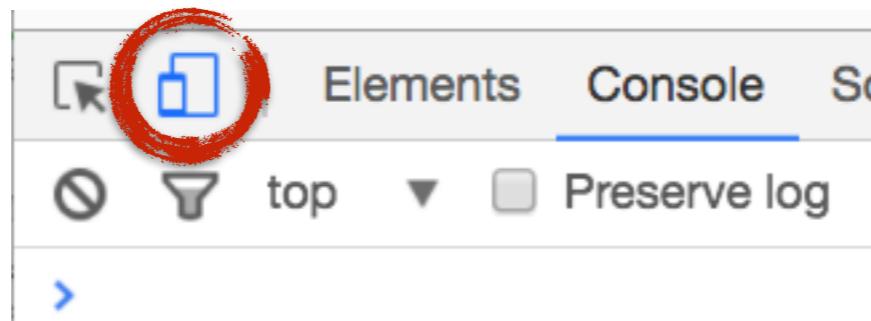
**APIs Javascript en
móviles**

Algunos APIs útiles en móviles

- Interacción con el hardware
 - **Cámara/Micrófono**
 - **Touch**
 - **Acelerómetro y giroscopio**
 - **Geolocalización**
- Uso sin conexión al servidor
 - **localStorage**
 - **Web SQL / IndexedDB**: bases de datos en el cliente
 - **service worker**: entre otras cosas, permite el funcionamiento *offline*
- Otros
 - **Battery status**
 - **Vibration**
 - **Payment Request API**
 - **Speech**: síntesis y reconocimiento
 - ...

Para probar/depurar estos APIs (Chrome)

- Opción 1: en la versión de escritorio, herramientas para desarrolladores (la emulación es limitada)



- Opción 2: en un emulador
 - <http://emilypporta.com/how-to-actually-run-the-chrome-web-inspector-with-an-emulated-android-device/>
- Opción 3: en un dispositivo real
 - <https://developers.google.com/web/tools/chrome-devtools/remote-debugging/>

Video/Audio en tiempo real

- Hay un **gran número de APIs relativamente recientes** para obtener video/audio de la cámara del usuario, capturar imágenes estáticas, grabar el *stream*, usar todo esto para comunicación entre navegadores (*chat*)...
- Ver los **estándares propuestos** en el grupo de interés del W3C “Web Real-Time Communications”: https://www.w3.org/TR/tr-groups-all#tr_Web_Real-Time_Communications_Working_Group
- De menor a mayor complejidad
 - HTML media capture
 - Media capture (`getUserMedia`)
 - MediaRecorder

HTML media capture

- Permite **seleccionar imagen/video/audio** o **capturarlo** de la cámara/micrófono usando un `<input type="file">`

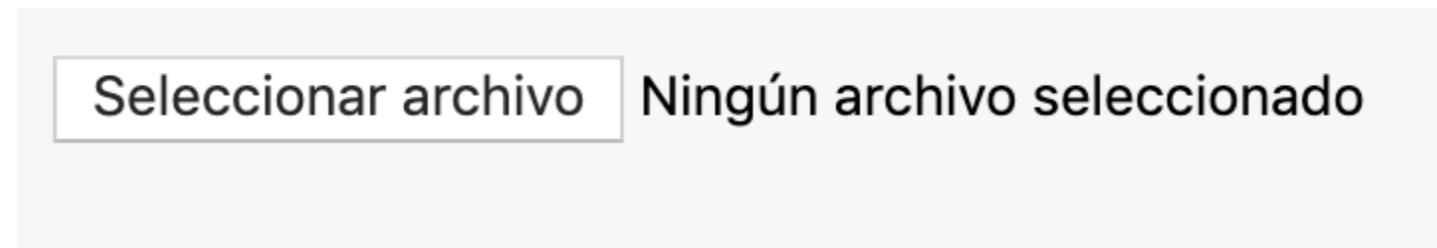
```
<input type="file" accept="image/*" capture>
<input type="file" accept="video/*" capture>
<input type="file" accept="audio/*" capture>
```

<https://jsbin.com/yeqilej/1/edit?html.js.output>

- Combinando esto con Javascript podemos:
 - Mostrar la imagen en la página en un tag `` ([ejemplo](#))
 - Manipularla usando el API Canvas (API de dibujo 2D)
 - Subir la imagen al servidor con AJAX
 - ...

Problemas del media capture

- El **aspecto** del `<input type="file">` no es muy configurable
- La captura de cámara/micrófono no es compatible con escritorio (solo la selección de archivos)



Media Capture and Streams (a.k.a. `getUserMedia`)

```
<video id="cam_video">Aquí aparecerá el video</video>
```

```
navigator.mediaDevices.getUserMedia({
  video: true
}).then(function(stream) {
  var video = document.getElementById('cam_video');
  video.srcObject = stream;
  video.play();
})
```

<https://codepen.io/ottocol/pen/yPWoWP?editors=1010>

En el objeto pasado a `getUserMedia` se puede pedir si queremos solo video, también audio, qué resoluciones preferimos y qué cámara si hay más de una, por ejemplo

```
{
  audio: true,
  video: {
    width: { min: 1280 },
    height: { min: 720 }
  }
}
```

(más info en <https://developer.mozilla.org/es/docs/Web/API/MediaDevices/getUserMedia>)

Capturar imágenes

```
<video id="player" controls autoplay></video>
<button id="capture">Capturar</button>
<canvas id="snapshot" width=320 height=240></canvas>
```

```
var player = document.getElementById('player');
var snapshotCanvas = document.getElementById('snapshot');
var captureButton = document.getElementById('capture');

var handleSuccess = function(stream) {
    // Attach the video stream to the video element and autoplay.
    player.srcObject = stream;
};

captureButton.addEventListener('click', function() {
    var context = snapshotCanvas.getContext('2d');
    // Draw the video frame to the canvas.
    context.drawImage(player, 0, 0, snapshotCanvas.width,
                      snapshotCanvas.height);
});

navigator.mediaDevices.getUserMedia({video: true})
    .then(handleSuccess);
```

De <https://developers.google.com/web/fundamentals/media/capturing-images/?hl=es>
Ver demo en <https://codepen.io/ottocol/pen/zYYQXWq?editors=1010>

Grabar streams: MediaRecorder

```
<video id="cam_video"></video>
```

```
navigator.mediaDevices.getUserMedia({
  video: true
}).then(function (stream) {
  var recorder = new MediaRecorder(stream);
  recorder.addEventListener('dataavailable', function(e) {
    video = document.getElementById("cam_video")
    video.src = URL.createObjectURL(e.data);
    video.loop = true
    video.play();
  });
  recorder.start();
  setTimeout(function(){
    recorder.stop();
    console.log("Grabado!!")
  }, 2000);
})
```

<https://codepen.io/ottocol/pen/RxwoNB?editors=1010>

Más ejemplos en <https://developers.google.com/web/fundamentals/media/recording-video?hl=es>

Detectores de formas

- **Caras, textos, códigos QR**
- Estándar en “incubación”, por ahora solo implementado en Chrome (hay que activar un flag del navegador)

```
<video id="video"></video>

const video = document.getElementById('video')
const faceDetector = new FaceDetector()
faceDetector.detect(video)
    .then(caras => {
        console.log("Num. caras detectadas: " + caras.length)
        caras.forEach(cara => {
            cara.landmarks.forEach(landmark => {
                if (landmark.type == 'eye') {
                    console.log("Ojo en " + landmark.locations[0].x + "," +
                               landmark.locations[0].y)
                }
            })
        })
    })
})
```

Demo: <https://codepen.io/ottocol/pen/OOexmR>

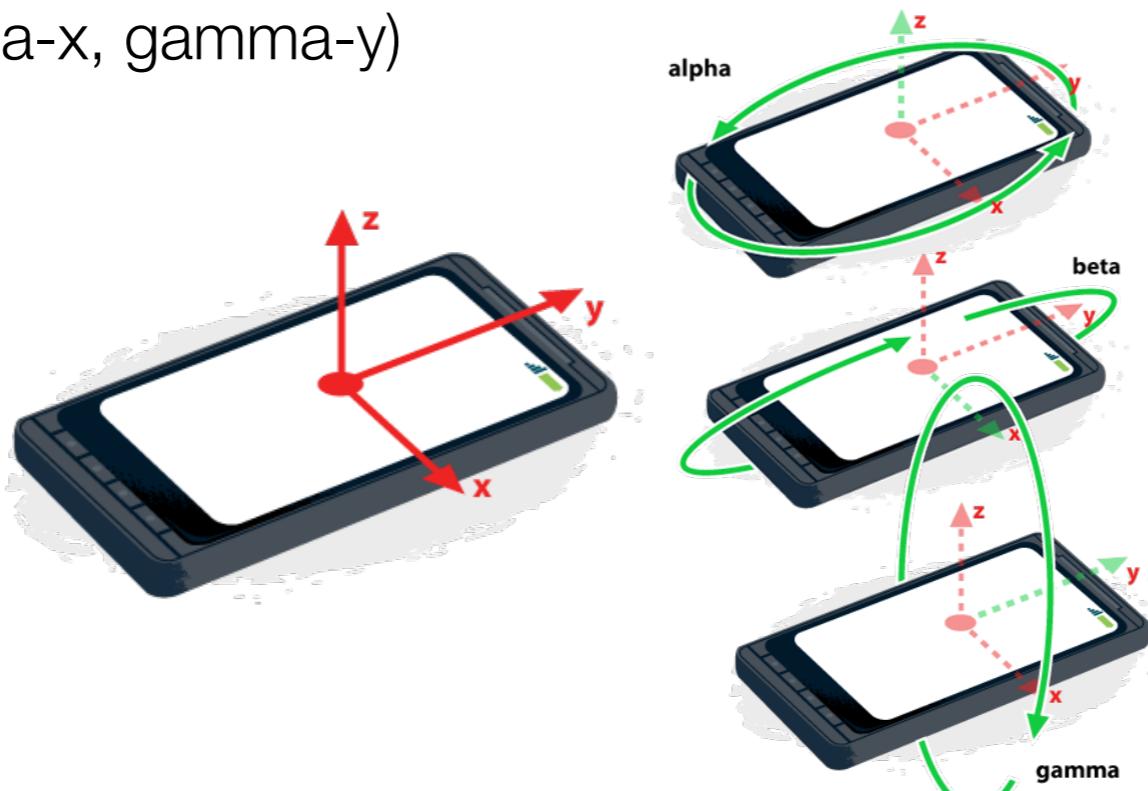
Buen artículo de introducción al API (ejemplo base de la demo anterior) <https://blog.arnellebalane.com/introduction-to-the-shape-detection-api-e07425396861>
Tutorial de web.dev: <https://web.dev/shape-detection/>

Touch API

- **Eventos** `touchstart`, `touchmove`, `touchend`
 - Similares a los correspondientes de ratón: `mousedown` (pulsado botón), `mousemove`, `mouseup` (soltado botón)
- **Propiedades** del evento:
 - Array “`touches`”. En dispositivos sin *multitouch* este array solo tiene una posición
 - Cada posición del array tiene su `target` (elemento DOM tocado), coordenadas (`clientX`, `clientY`), la “forma del dedo” (`radiusX`, `radiusY`, `rotationAngle`) etc
- Tutorial: <http://www.html5rocks.com/es/mobile/touch/>
- No implementa “gestos” (`swipe`, `pinch-to-zoom`, ...), pero sí hay muchas librerías que lo hacen, por ejemplo hammer.js

Orientación en 3D

- Con el **acelerómetro** y/o **giroscopio** se puede detectar el movimiento (evento ‘devicemotion’) y la posición actual en 3D (evento ‘deviceorientation’)
 - Los eventos se disparan x veces por segundo (típicamente 50-60)
 - **deviceorientation** informa de ángulos actuales con respecto a los ejes 3D (alpha-z, beta-x, gamma-y)



- **devicemotion** informa de aceleración y velocidad de rotación

Tutorial y demo: <https://code.tutsplus.com/tutorials/an-introduction-to-the-device-orientation-api--cms-21067>

Orientación de la pantalla

- Si no necesitamos posición 3D sino solo detectar cambios de orientación entre modo **vertical** (*portrait*) y **horizontal** (*landscape*)
- `screen.orientation`: <https://developer.mozilla.org/en-US/docs/Web/API/Screen/orientation>
- Detectar cambios: evento **orientationchange** del objeto **window**

```
window.addEventListener('orientationchange', function(){
    //solo puede ser 0,90,180,270
    console.log("Angulo actual: " + screen.orientation.angle)
    console.log("Tipo de orientacion: " + screen.orientation.type)
})
```

Almacenamiento offline

- **Almacenamiento local:** bases de datos en el cliente
 - localStorage
 - Web SQL (ya sin soporte en iOS, solo Android)
 - IndexedDB (iOS/Android)
- **Cache** de recursos de red (HTML, CSS, JS) para que la aplicación se pueda seguir usando en la medida de lo posible
 - cache manifest (*deprecated*)
 - service workers (*lo veremos la semana que viene*)

Ejemplo: PouchDB

- Un wrapper de IndexedDB, ya que este es un API de “demasiado bajo nivel”
- <https://codepen.io/astol/pen/qbbOXE?editors=1010>

```
var db = new PouchDB('todos');
var remoteCouch = false;
```

Crear BD

```
function addTodo(text) {
  var todo = {
    _id: new Date().toISOString(),
    title: text,
    completed: false
  };
  db.put(todo, function callback(err, result) {
    if (!err) {
      console.log('Successfully posted a todo!');
    }
  });
}
```

Insertar documento

(Otro ejemplo, no el de los TO-DOS)
Para poder hacer queries sobre un campo se tiene que crear un índice para el mismo

```
db.createIndex({
  index: {
    fields: ['name', 'age']
  }
}).then(function () {
  return db.find({
    selector: {
      name: 'mario',
      age: {$gt: 21}
    }
  });
});
```

Otros APIs...

- Interacción por voz
 - **Speech Synthesis**: soporte en iOS y Android [[demo](#)]

```
let utterance = new SpeechSynthesisUtterance("Hola, ¿qué tal lleváis la clase de ADI?");  
speechSynthesis.speak(utterance);
```

- **Speech Recognition**: soporte solo en Android, el reconocimiento se hace *online*, no es local [[demo](#)]

Geolocalización

- Devuelve la **posición geográfica** del usuario (latitud, longitud)
- **Métodos** de localización
 - Hay métodos con alta precisión (GPS) y baja (a partir de la dirección IP o usando la red GSM)
 - El método exacto por el que se está calculando la posición es transparente al desarrollador Javascript (aunque se puede indicar si queremos alta o baja precisión)
- Lo único que nos da el API son las **coordenadas**. Necesitaremos algún servicio adicional si queremos dibujar un **mapa** con la posición, etc. (p.ej. Google Maps)
- Al ser un API relativamente antiguo funciona con **callbacks**, no con promesas

Ejemplo sencillo

- Sin chequeo de errores ni opciones de localización
- obtener la posición `navigator.geolocation.getCurrentPosition()` pero no la devuelve directamente. Hay que pasarle el nombre de una función *callback* (recibirá la posición en un parámetro)
 - La posición recibida es un objeto con dos campos: `coords` (con la info básica: latitud, longitud, etc) y `timestamp`
 - Antes de obtener la posición el navegador va a pedir permiso al usuario. Si no se concede, el *callback* no se ejecuta

```
navigator.geolocation.getCurrentPosition(function(pos){  
    alert("Estás en (" + pos.coords.latitude + " , " + pos.coords.longitude + ")");  
})
```

Gestión de errores

- Podemos pasar un segundo *callback* a `getCurrentPosition`: una función que se llamará si se ha producido algún error
 - Por ejemplo el usuario no ha dado permiso, o no hay dispositivos de localización
 - El callback de error recibe como argumento un objeto con dos campos. El más interesante es `code`, un código de error: 1:permiso denegado, 2:No se puede calcular la posición, 3:Timeout, 0>Error desconocido

```
navigator.geolocation.getCurrentPosition(mostrarPosicion, verError);

function mostrarPosicion(pos) {
    //este es el callback del ejemplo de la transparencia anterior
}

function verError(error) {
    if (error.code == 1)
        alert("No has dado permiso para ver tu posición")
}
```

Opciones de localización

- Tercer parámetro (opcional) de getCurrentPosition: objeto con tres campos:
 - enableHighAccuracy (booleano): indica si queremos una localización de precisión (p.ej. GPS) o nos basta con una aproximada (p.ej. usando la red de móvil)
 - timeout (nº en milisegundos) tiempo que estamos dispuestos a esperar que el dispositivo nos dé una posición. Pasado este tiempo se generará un error de timeout
 - maximumAge (nº en milisegundos) si el dispositivo tiene en cache una posición con una antigüedad inferior a esta, nos vale, no es necesario que calcule la actual.

```
//queremos alta precisión  
//pero nos vale con la posición de hace un minuto  
navigator.geolocation.getCurrentPosition(mostrarPosicion, verError,  
{enableHighAccuracy: true, maximumAge:60000});
```

Algunas referencias

- **Google Web Fundamentals:** consejos de diseño, info sobre APIs en desktop y móvil <https://developers.google.com/web/fundamentals?hl=es>
- **Mozilla Development Network:** documentación, artículos, tutoriales (<https://developer.mozilla.org/en-US/docs/Web/API>)
- **Dive into HTML5:** (diveintohtml5.info/) libro publicado por O'Reilly, versión gratuita on-line
- **Pocket Javascript:** *Blog con ejemplos detallados, artículos* <http://www.pocketjavascript.com>