

Tema 6: Apps web en dispositivos móviles

Parte I: Introducción y APIs Javascript

Tema 5: Apps web en dispositivos móviles

6.1.

Tipos de aplicaciones en móviles

(Según la plataforma de desarrollo)

Aplicaciones nativas

Desarrolladas con el SDK nativo de la plataforma y en los lenguajes soportados por ellas (*Kotlin/Java en Android, Swift 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, micrófono, 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 alguna tienda de apps oficial (App Store)
 - Algunos APIs no accesibles, aunque cada vez menos. Proyecto Fugu, sigue la pista de la *brecha* entre APIs nativos/APIs Web

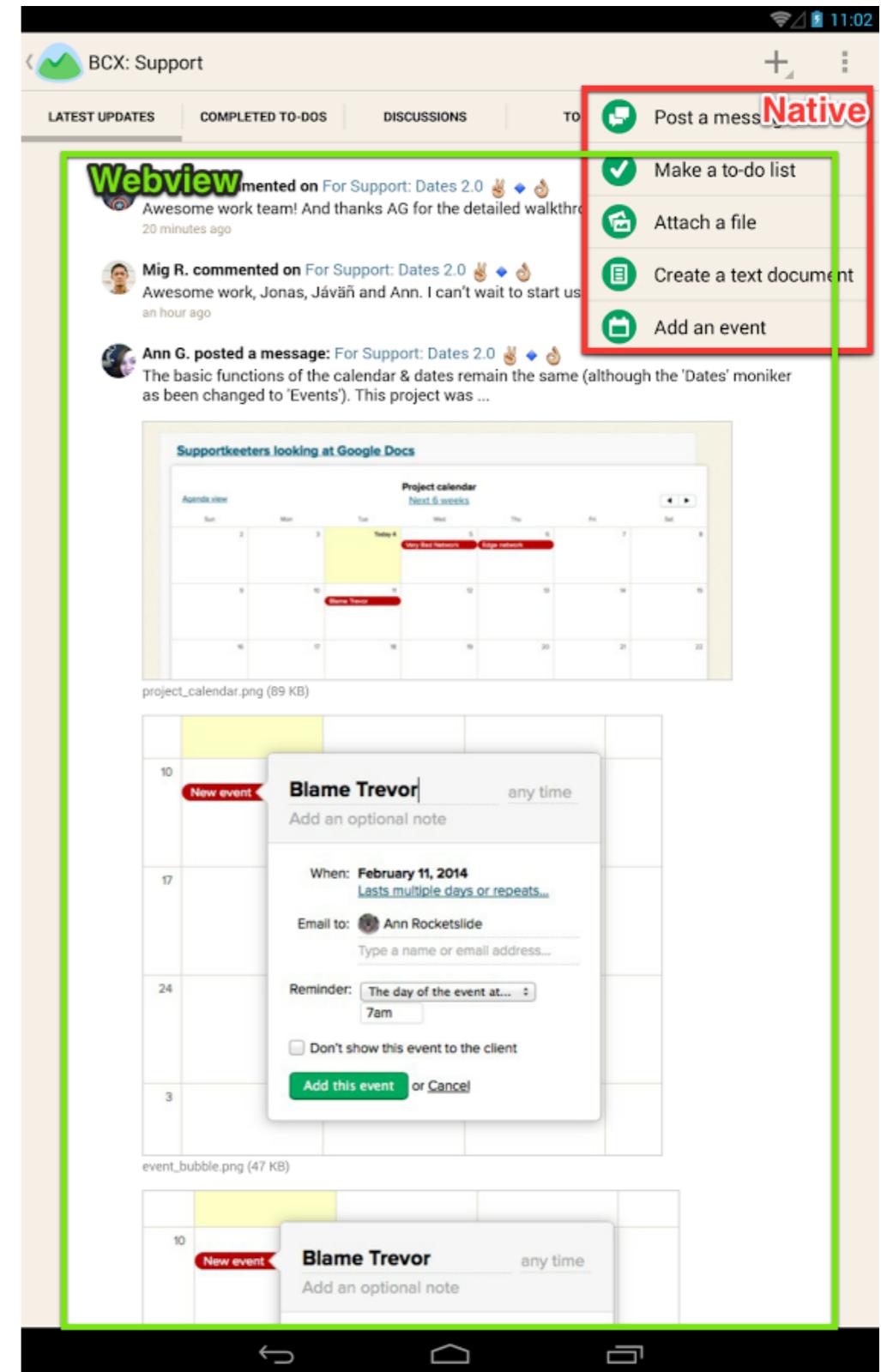
Ejemplo de apps web: PWAs

- **Progressive Web Apps:** son aplicaciones web pero dan la impresión de ser nativas
- Se pueden instalar 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)



Ejemplo: Capacitor

- <https://capacitorjs.com/>. La **plataforma híbrida** más conocida en la actualidad
 - Muy similar a Apache Cordova, aunque más moderno
- 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
- Podemos crear plugins 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 multiplataforma

- 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 (más info)
 - En JS: React native (React), NativeScript (Angular, Vue, Typescript, *vanilla* JS)
 - En otros lenguajes: C# - .NET MAUI, Dart - Flutter , Kotlin - Kotlin Multiplatform Mobile



Kotlin Multiplatform



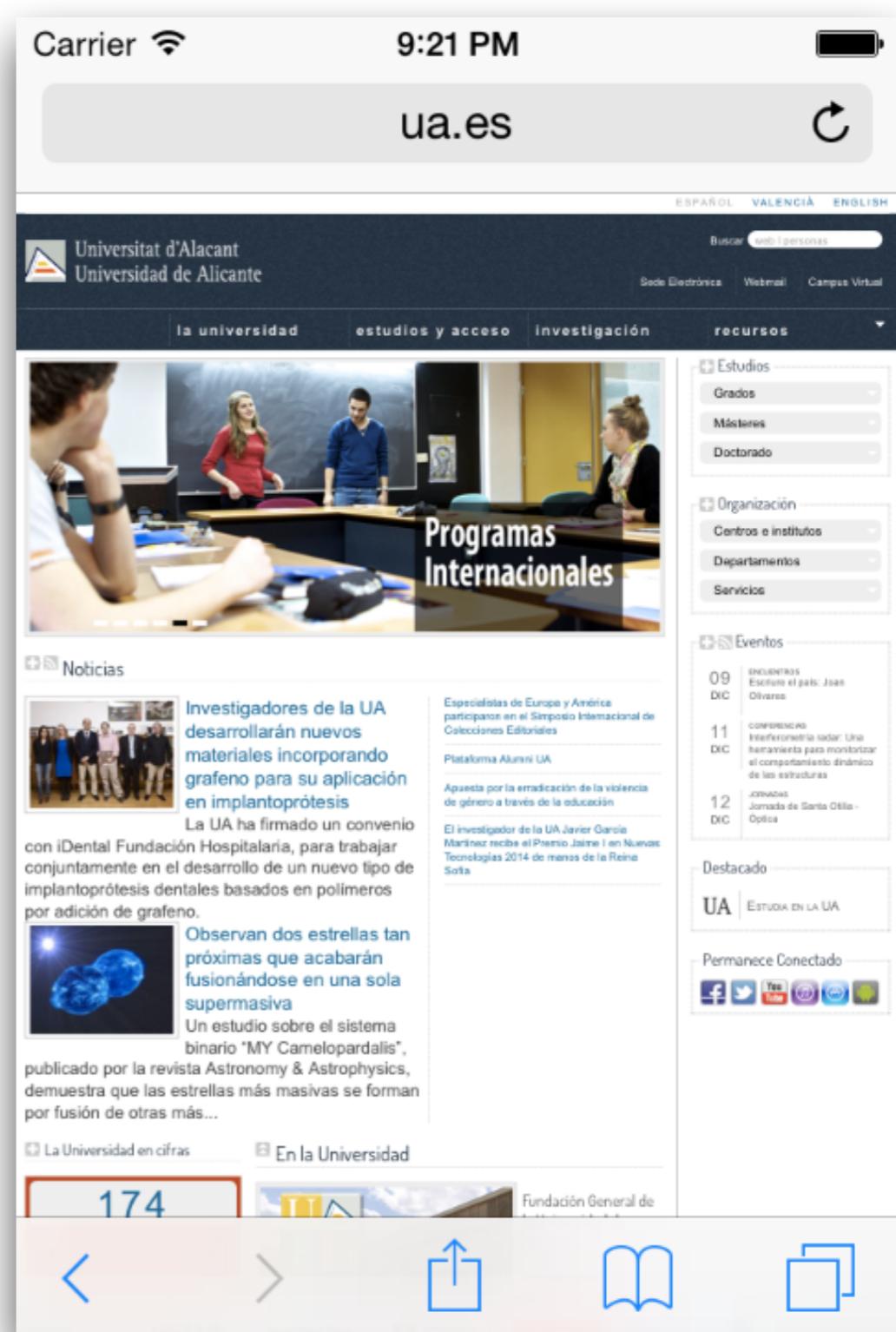
Tema 5: Apps web en dispositivos móviles

6.2.

Principios generales de
diseño de apps web
móviles

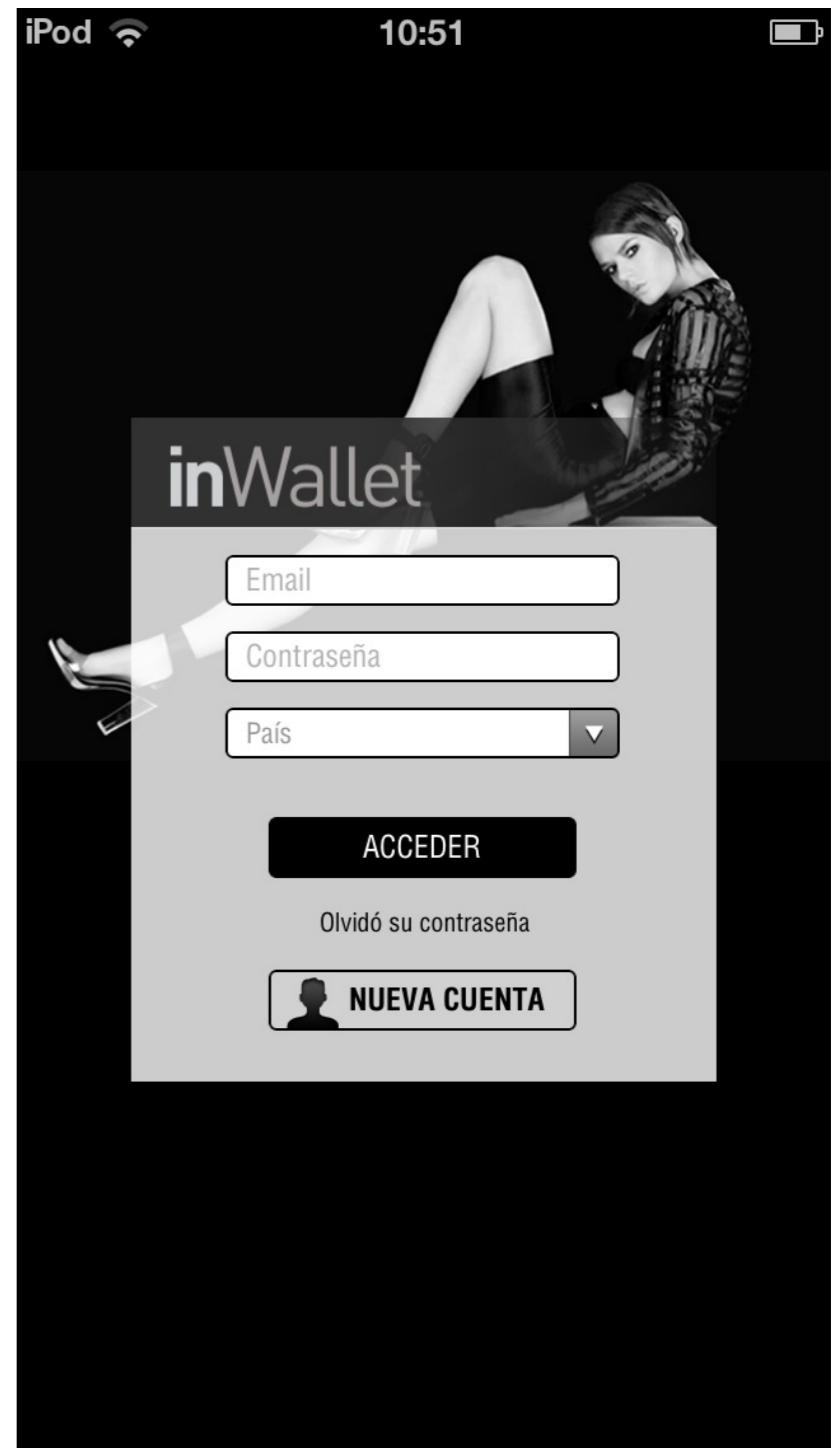
La pantalla

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



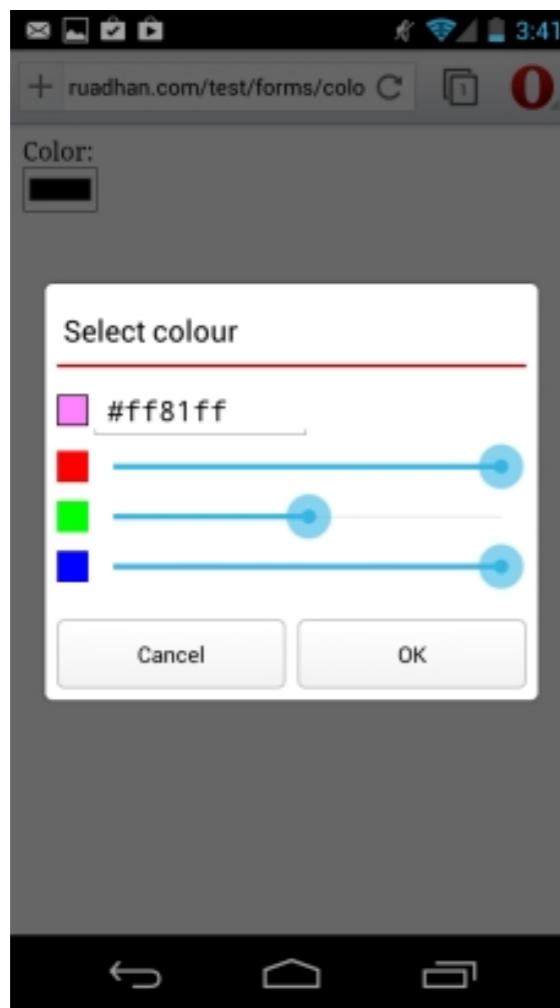
La entrada de datos es tediosa

- 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.

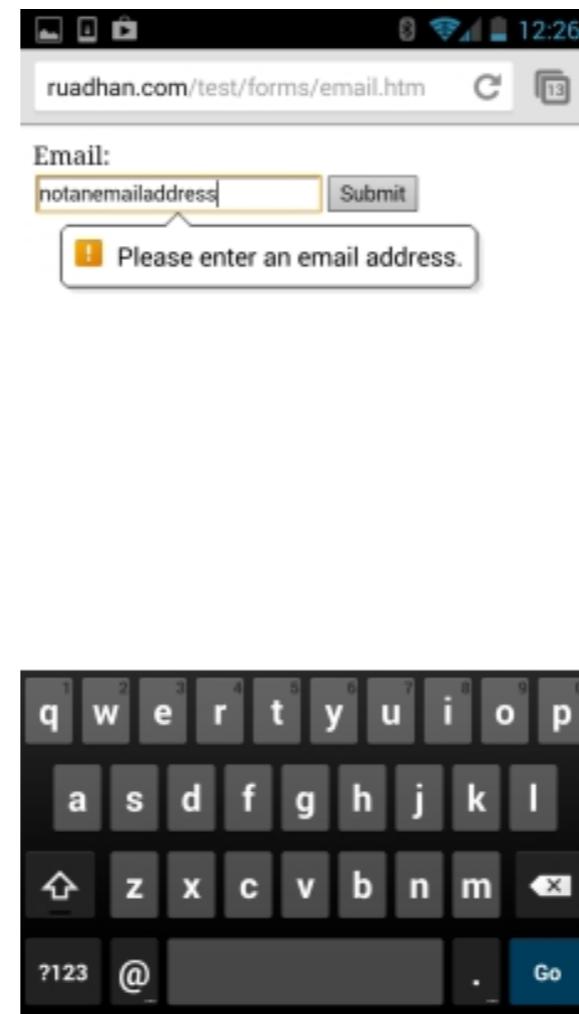


Facilitar la entrada de datos

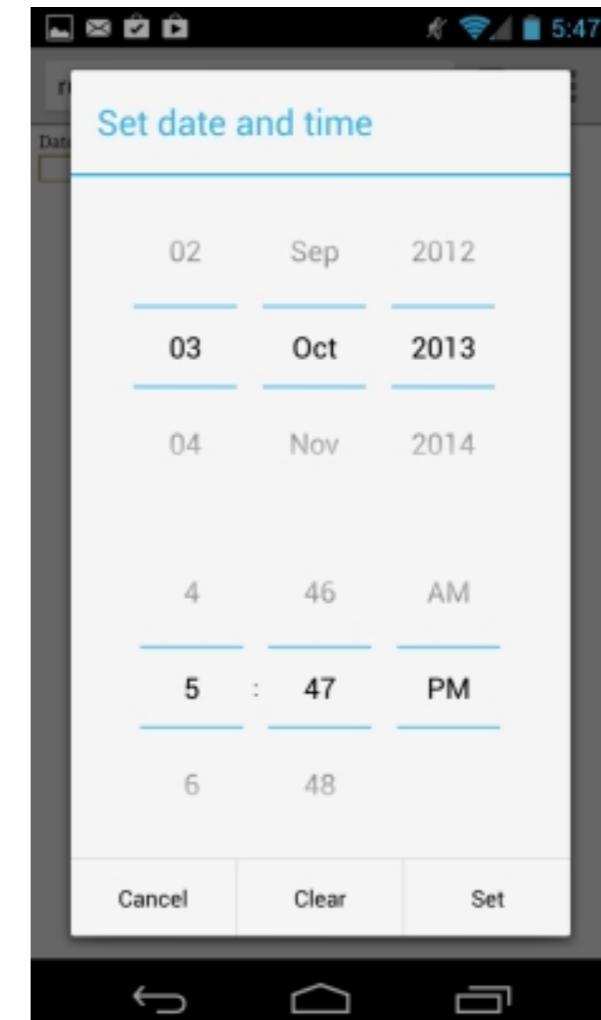
- Usar <input> del type más apropiado (number, date, email, color,...)



<input type="color"/>



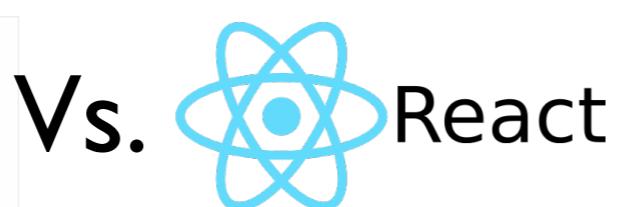
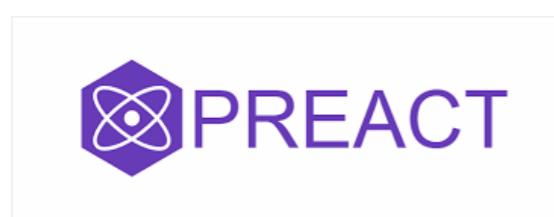
<input type="email"/>



<input type="datetime-local"/>

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



Pros of Preact

- Fast & Lightweight: Preact is just 3.5 kb in size and renders quickly, making it the best framework for building high-performing lightweight apps.
- Compatible: It is highly compatible with React API and supports the same ECMAScript, making it efficient enough to integrate with an existing React project for enhanced performance.

- Usar rendering en el servidor (SSR, SSG) como vimos la semana pasada
- No malgastar recursos hardware (p.ej. precisión en la localización, como luego veremos)

Code splitting

- Usar el `import()` “dinámico” de ES6

```
import { createRouter, createWebHistory } from 'vue-router';

const routes = [
  {
    path: '/',
    name: 'Home',
    component: () => import('./views/Home.vue'),
  },
  {
    path: '/about',
    name: 'About',
    component: () => import('./views/About.vue'),
  },
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});
```

- Los *bundlers* como Vite o Webpack colocan cada import dinámico en un archivo independiente (*chunk*) que se cargará solo cuando se necesite

Más consejos

- En web.dev, de Google: <https://web.dev/learn/pwa/foundations/>
- Charla “[Principles of mobile app design](#)”, en Google IO 2016

The screenshot shows a web browser window with the URL <https://web.dev/learn/pwa/foundations/>. The page title is "Learn PWA!". The navigation bar includes "Learn PWA!" and "Share". Below the navigation, there's a breadcrumb trail: "web.dev > Learn > Learn PWA!". A timestamp "003" is visible. The main content title is "Foundations". A descriptive paragraph states: "All Progressive Web Apps are, at their core, modern websites, so it's important that your website has a solid foundation in responsive design, mobile and everything first, intrinsic design, and web performance." At the bottom, there's a "On this page" section with a dropdown arrow.



Tema 5: Apps web en dispositivos móviles

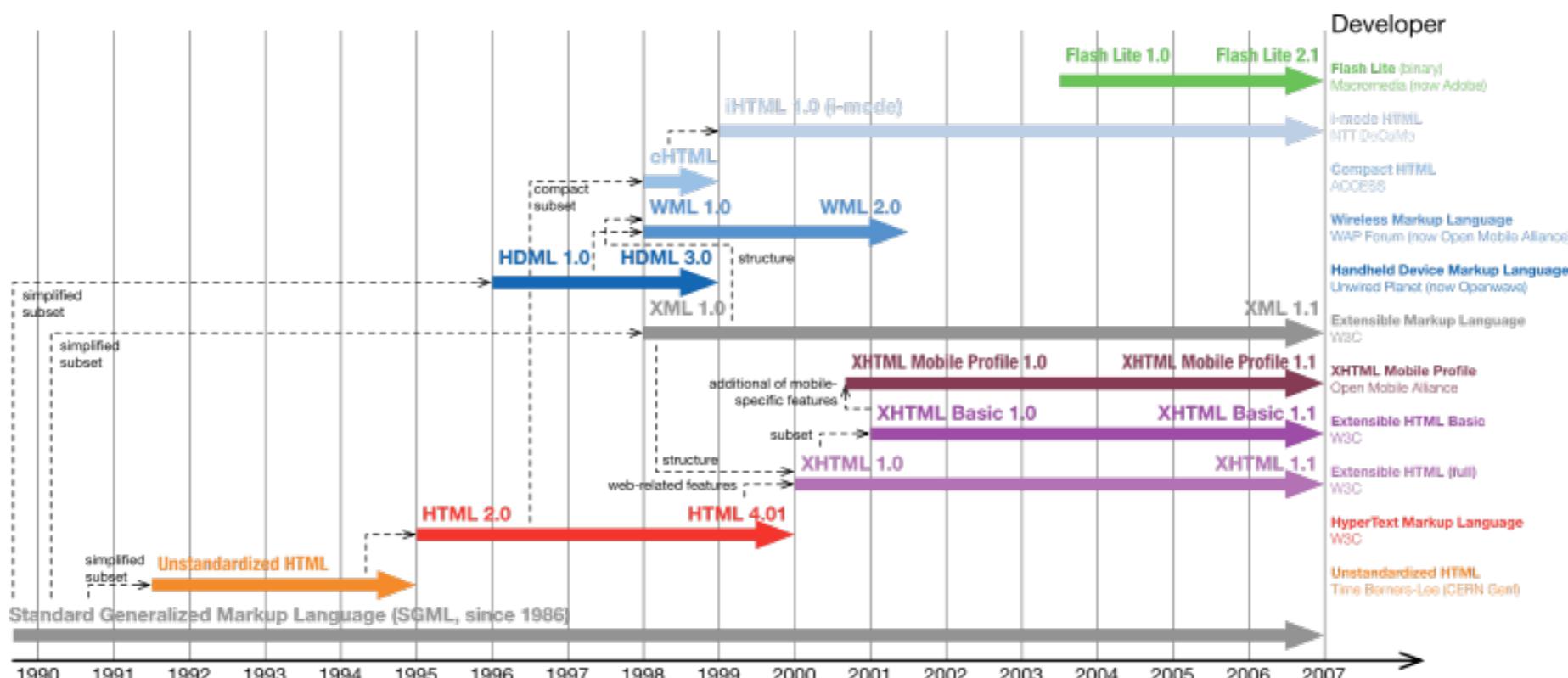
6.3.

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



Volvamos por un momento al 2007



320px



Universitat d'Alacant
Universidad de Alicante

la universidad



Investigación
y Empresa

NOTICIAS



Investigadores de la Universidad de Alicante obtienen carbón activo de una planta de tratamiento de aguas residuales mediante el uso de un filtro



ESPAÑOL VALENCIÀ ENGLISH

Buscar web | personas

Sede Electrónica Webmail UACloud CV

Estudios y acceso investigación recursos

Programas Internacionales

Estudios

- Grados
- Másteres
- Doctorado

Organización

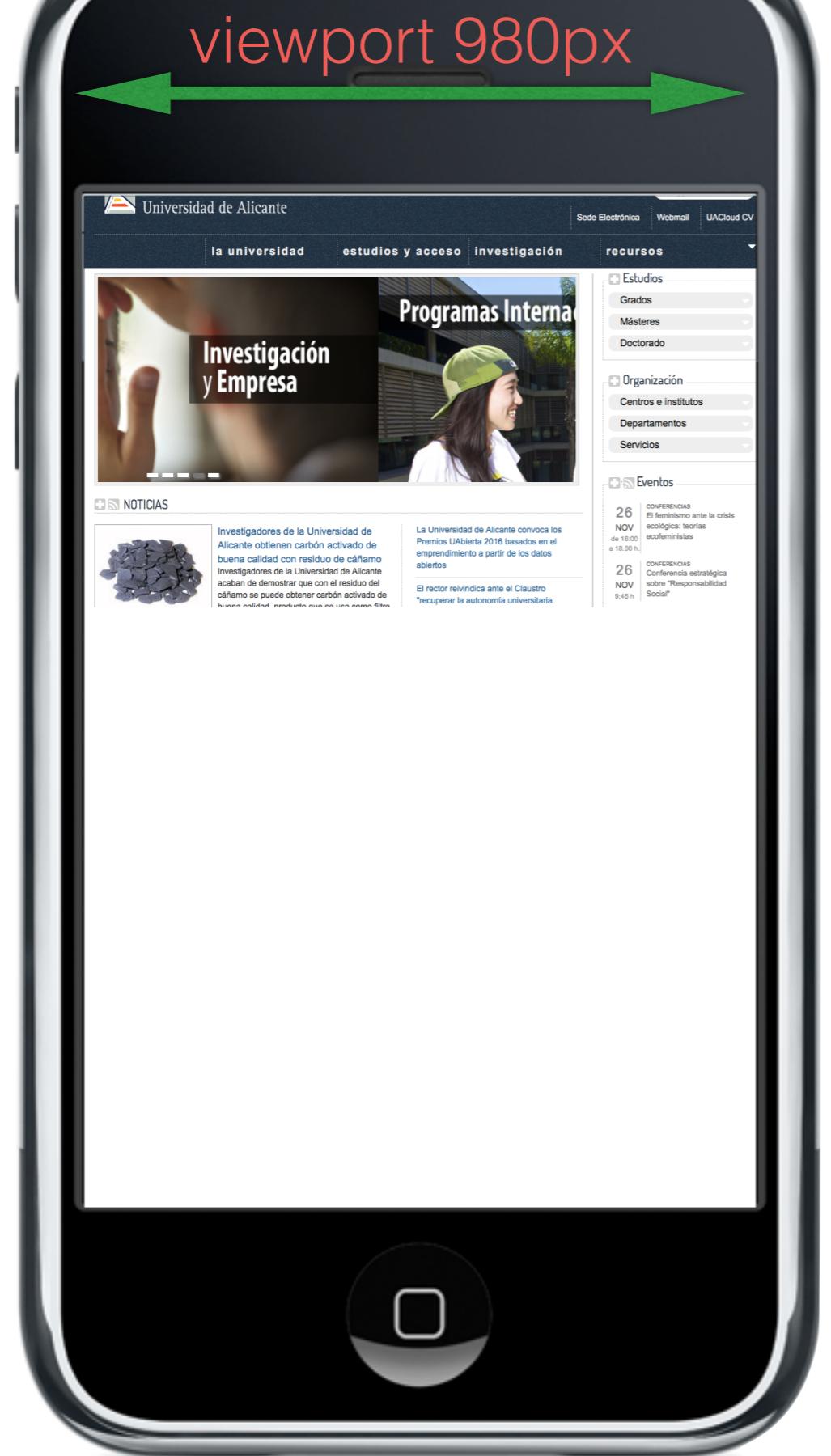
- Centros e institutos
- Departamentos
- Servicios

Eventos

26 NOV CONFERENCIAS
El feminismo ante la crisis ecológica: teorías ecofeministas
de 16:00 a 18.00 h.

26 NOV CONFERENCIAS
Conferencia estratégica sobre "Responsabilidad Social"

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>

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>`

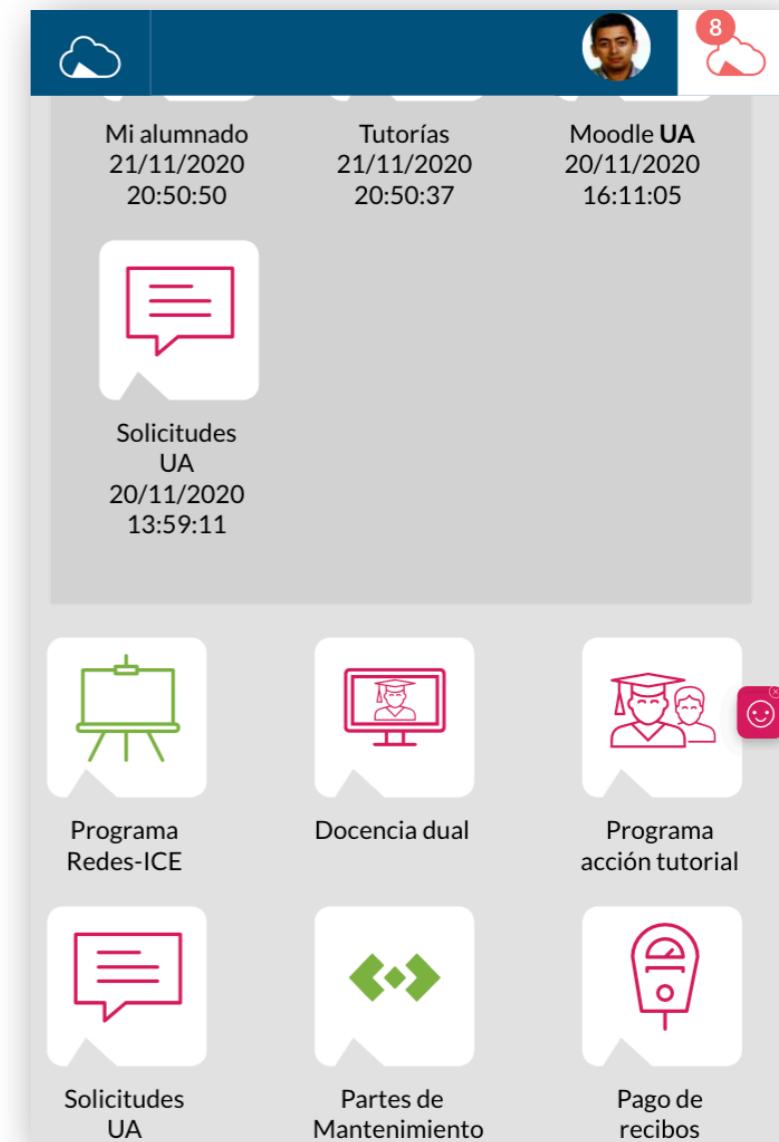
Un pixel no siempre es un pixel

- “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 16 Pro Max reporta que tiene 440 px de ancho en “portrait” (CSS pixels) cuando en realidad tiene 1320 físicos <https://yesviz.com/viewport/>)
- `window.devicePixelRatio`: ¿Cuántos píxeles reales es 1 pixel CSS? (en el ejemplo anterior, 3)
- ¿Para qué se usan los píxeles “extra”?
 - Para mostrar texto más nítido (lo hace el navegador de manera “nativa”)
 - Para servir imágenes de “alta resolución” vs. “resolución estándar”: tag `<picture>`, atributo `srcset` del tag `` (<https://web.dev/responsive-images/>)

Responsive web



The mobile responsive web interface for UACloud features a dark blue header bar with the UACloud logo, a user profile icon, and a notification bell icon showing 8 notifications. Below the header, a message encourages users to download the app for Android or iOS. The main content area has a light gray background and includes a "Últimas" (Last) section with four cards: "Mi alumnado" (21/11/2020 20:50:50), "Tutorías" (21/11/2020 20:50:37), "Moodle UA" (20/11/2020 16:11:05), and "Solicitudes UA" (20/11/2020 13:59:11). At the bottom, there is a row of seven icons representing various services: Programa Redes-ICE, Docencia dual, Programa acción tutorial, Solicitudes UA, Partes de Mantenimiento, Pago de recibos, and AlumniUA.



The mobile responsive web interface for UACloud features a dark blue header bar with the UACloud logo, a user profile icon, and a notification bell icon showing 8 notifications. Below the header, a message encourages users to download the app for Android or iOS. The main content area has a light gray background and includes a "Últimas" (Last) section with four cards: "Mi alumnado" (21/11/2020 20:50:50), "Tutorías" (21/11/2020 20:50:37), "Moodle UA" (20/11/2020 16:11:05), and "Solicitudes UA" (20/11/2020 13:59:11). At the bottom, there is a row of seven icons representing various services: Programa Redes-ICE, Docencia dual, Programa acción tutorial, Solicitudes UA, Partes de Mantenimiento, Pago de recibos, and AlumniUA.

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" />

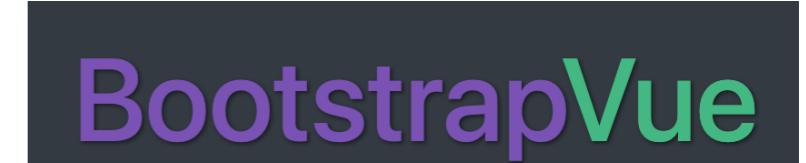
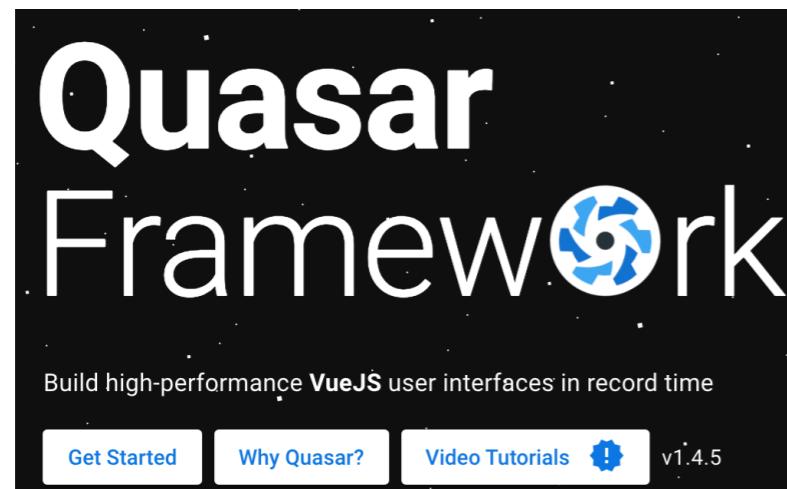
<!-- también se puede poner en el CSS “empotrado” en el HTML -->
<style>
  @media screen and (max-device-width:480px) {
    body {background-color: red;}
  }
</style>
```

Sintaxis de media queries

- **Expresiones:**
 - min- (>=). p.ej. min-device-width:640px
 - max- (<=). p.ej. max-device-width:640px
 - : (==). p.ej. device-width:640px
- **Operadores:** not, and, or, only (only screen serían dispositivos que solo soportan media="screen", típicamente móviles)
- Algunas **características** comprobables
 - device-width, device-height: medidas del dispositivo
 - width, height: medidas del viewport
 - orientation (puede ser landscape o portrait)
 - resolution (típicamente en dpi)
 - aspect-ratio, device-aspect-ratio

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 5: Apps web en dispositivos móviles

6.4.

APIs Javascript en
móviles



Aclaración: los APIs que veremos a continuación no son exclusivos de dispositivos móviles, pero en estos es cuando podemos estar razonablemente seguros de tener el hardware necesario para que funcionen (cámara, acelerómetro, GPS,...)

Algunos APIs útiles en móviles

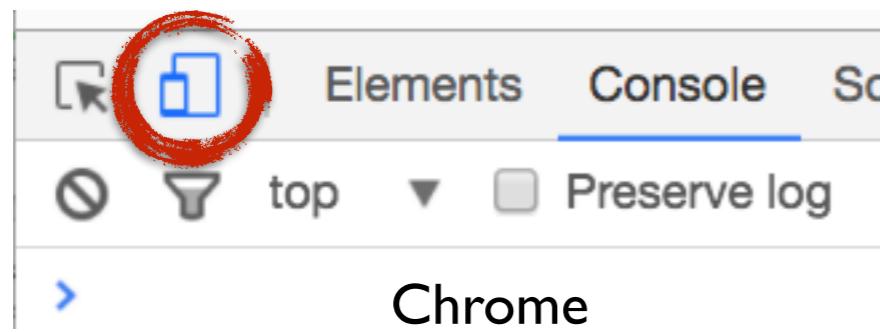
- Interacción con el hardware
 - **Cámara/Micrófono**
 - **Touch**
 - **Acelerómetro y giroscopio**
 - **Geolocalización**
 - **Battery status**
 - **Vibration**
- Uso sin conexión al servidor
 - **localStorage**
 - **Web SQL / IndexedDB:** bases de datos en el cliente
 - **service worker:** entre otras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

<https://whatwebcando.today/>

Sitio interesante con APIs soportadas por tu navegador actual, documentación y demos

Para probar/depurar estos APIs

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



- Opción 2: en un emulador
 - <https://stackoverflow.com/questions/30901688/how-to-use-a-web-inspector-with-emulated-android-device-avd>
- Opción 3: en un dispositivo real
 - Chrome: <https://developer.chrome.com/docs/devtools/remote-debugging?hl=es-419>
 - Firefox: <https://developer.mozilla.org/en-US/docs/Tools/about:debugging>

Video/Audio en tiempo real

- Hay **varios 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 (*videochat*)...
- Ver los **estándares propuestos** en el grupo de interés del W3C “Web Real-Time Communications” (algunas demos)
- 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">`

```
<!-- con "capture" se captura en ese momento, si no, se toma de la galería -->
<input type="file" accept="image/*" capture>
<input type="file" accept="video/*">
<input type="file" accept="audio/*">
```

- 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 `fetch`
- ...



¡¡Esto no funciona en el escritorio, solo en navegador móvil!!

Media Capture and Streams: conceptos previos

- MediaStream stream con audio y/o vídeo, tiene 1 o más *tracks*
 - Podemos asignarlo a una etiqueta <video>/<audio> con la propiedad `srcObject`
- MediaStreamTrack 1 única pista de audio o vídeo
- Blob: datos binarios, usado para archivos, imágenes,...
 - `URL.createObjectURL(blobInstance)`: pasar de Blob a URL para
- Canvas: área rectangular para dibujar en pantalla
 - Context: el “sitio” donde dibujamos, objeto que implementa todos los métodos para dibujar

Media Capture and Streams

(a.k.a. getUserMedia)

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

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

<https://codepen.io/ottocol/pen/yPVoWP?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>)

¡Primero chequear el soporte!

- Chequear que el navegador actual soporta el API
- Típicamente comprobar que el objeto/método que queremos usar no es `undefined`

```
if (navigator?.mediaDevices?.getUserMedia) {  
    //soporta el API  
    ...  
}
```

Capturar fotos

- `ImageCapture`, actualmente solo Android (Chrome/Android Browser)

```
<video id="video" style="height: 180px; width: 240px;"></video>
<button id="boton">Tomar Foto</button>
<img id="imagen" width="240" height="180">
```

```
var stream

document.addEventListener("DOMContentLoaded", async function(){
  stream = await navigator.mediaDevices.getUserMedia({video:true})
  var video = document.getElementById("video")
  video.srcObject = stream
  video.play()
})

document.getElementById("boton").addEventListener("click", async function(){
  var capt = new ImageCapture(stream.getVideoTracks()[0])
  var foto = await capt.takePhoto()
  var img = document.getElementById("imagen")
  img.src = URL.createObjectURL(foto)
})
```

Capturar imágenes “legacy”

```
<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');

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

var handleSuccess = function(stream) {
    player.srcObject = stream;
};

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>

Capturar streams

- MediaStream Recording API: MediaRecorder

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

```
let stream = await navigator.mediaDevices.getUserMedia({video: true})  
  
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(); ← Dispara el evento 'dataavailable'  
    console.log("Grabado!!")  
}, 2000);
```

Obtenemos el stream de video de la cámara, ponemos un MediaRecorder a grabar con “**start**” y 2 segundos después paramos la grabación (**stop**) y para verla de forma sencilla, la convertimos de Blob a URL y se la ponemos a un <video> HTML ya en la página

Blob

<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 de barras**
- Estándar en “incubación”, Android/últimas versiones de iOS

En Chrome desktop hay que activar un flag del navegador (Experimental Web Platform Features). En Safari Desarrollo > Comutación de funciones > buscar y marcar“shape detection”

```
<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>
Demo: <https://whatpwacando.today/face-detection/>

Demo de códigos de barras: <https://codepen.io/ottocol/pen/WNEBjj?editors=1111>

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/>

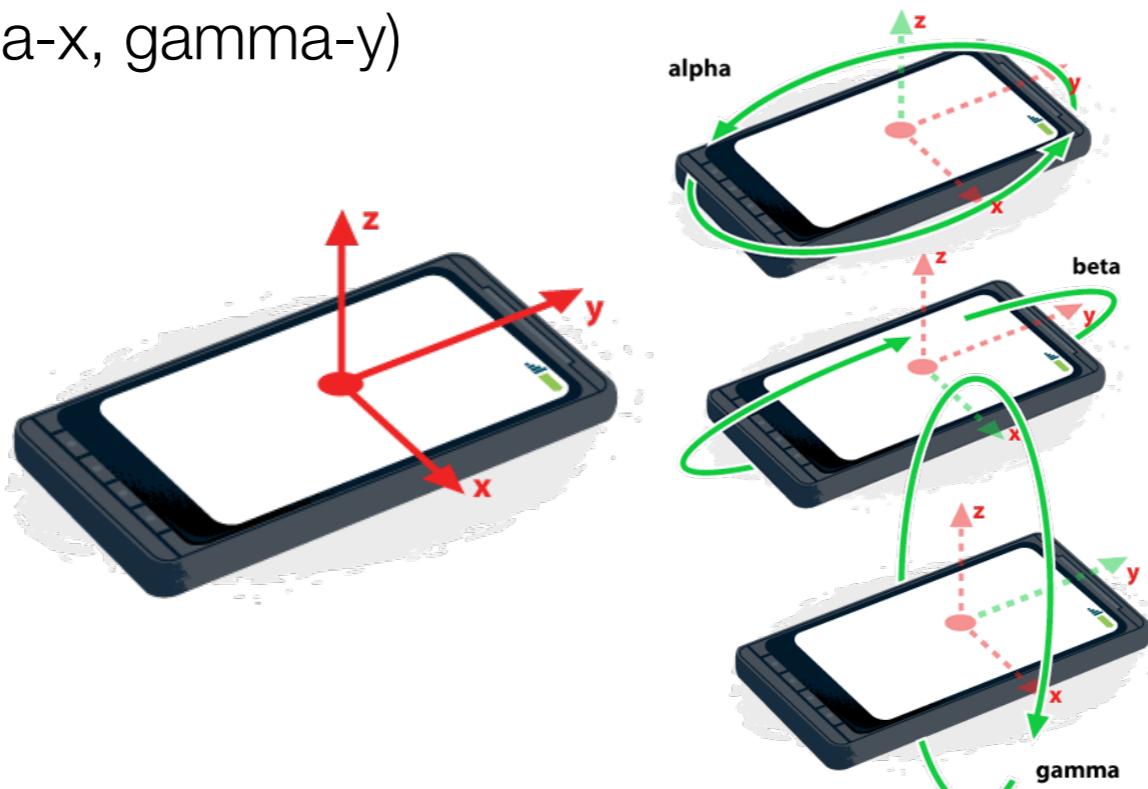
Orientación de la pantalla

- 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)
})
```

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

Almacenamiento offline

- **Almacenamiento local:** bases de datos en el cliente
 - localStorage (ya lo vimos, sencillo de usar pero no permite hacer búsquedas como en una BD)
 - Web SQL (“deprecated”, 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 éste es un API de “demasiado bajo nivel”
- Ejemplo TO-DO: <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!');
    }
  });
}
```

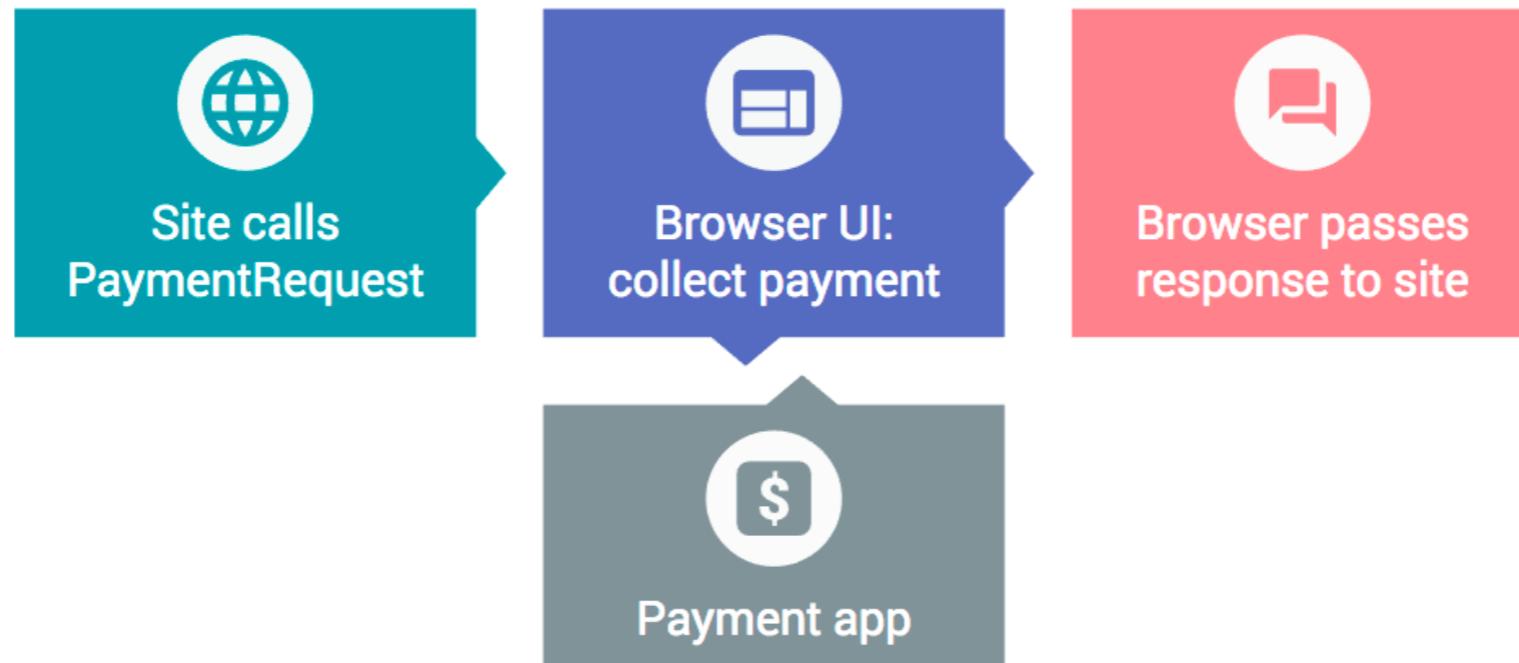
Insertar documento

(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...

- **Payments API:** simplificar la introducción de los datos de pago del usuario para hacer una compra online



<https://developers.google.com/web/fundamentals/payments/?hl=es>

Web Share

- Como el compartir  de las apps nativas. Se puede compartir una URL, texto, o conjunto de File

```
const shareData = {  
    title: 'UA',  
    text: 'Ven a estudiar a la UA!',  
    url: 'http://www.ua.es'  
}  
  
// El share debe ser disparado por un evento generado por el usuario  
document.getElementById("boton").addEventListener('click', async () => {  
    try {  
        await navigator.share(shareData)  
        console.log('OK')  
    } catch(err) {  
        console.log('Error: ' + err)  
    }  
});  
;
```

Interacción por voz

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

```
let utterance = new SpeechSynthesisUtterance("Yo no soy ni voo a ser tu bizcochito");
speechSynthesis.speak(utterance);
```

- **Speech Recognition:** 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 o la librería JS Leaflet)
- Al ser un **API “antiguo”** funciona con **callbacks**, no con promesas

Ejemplo en <https://stackblitz.com/edit/vitejs-vite-5zfj7w?file=index.html>

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, precisión, 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){  
    console.log("Estás en (" + pos.coords.latitude + ", " + pos.coords.longitude + ")");  
    console.log("con precisión de " + pos.coords.accuracy + " m.");  
})
```

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

- **Web dev:** consejos de diseño, info sobre APIs en desktop y móvil
<https://web.dev/learn/>
- **Mozilla Development Network:** documentación, artículos, tutoriales (<https://developer.mozilla.org/en-US/docs/Web/API>)
- El sitio de **Maximiliano Firtman**, artículos, charlas, etc sobre apps móviles web <https://firt.dev/learn/>