

Parte III: Desarrollo en dispositivos móviles

Tema 4:

Introducción a las Aplicaciones web en dispositivos móviles

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 (*Java, Kotlin, Swift, Obj-C,...*)

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



Aplicaciones web

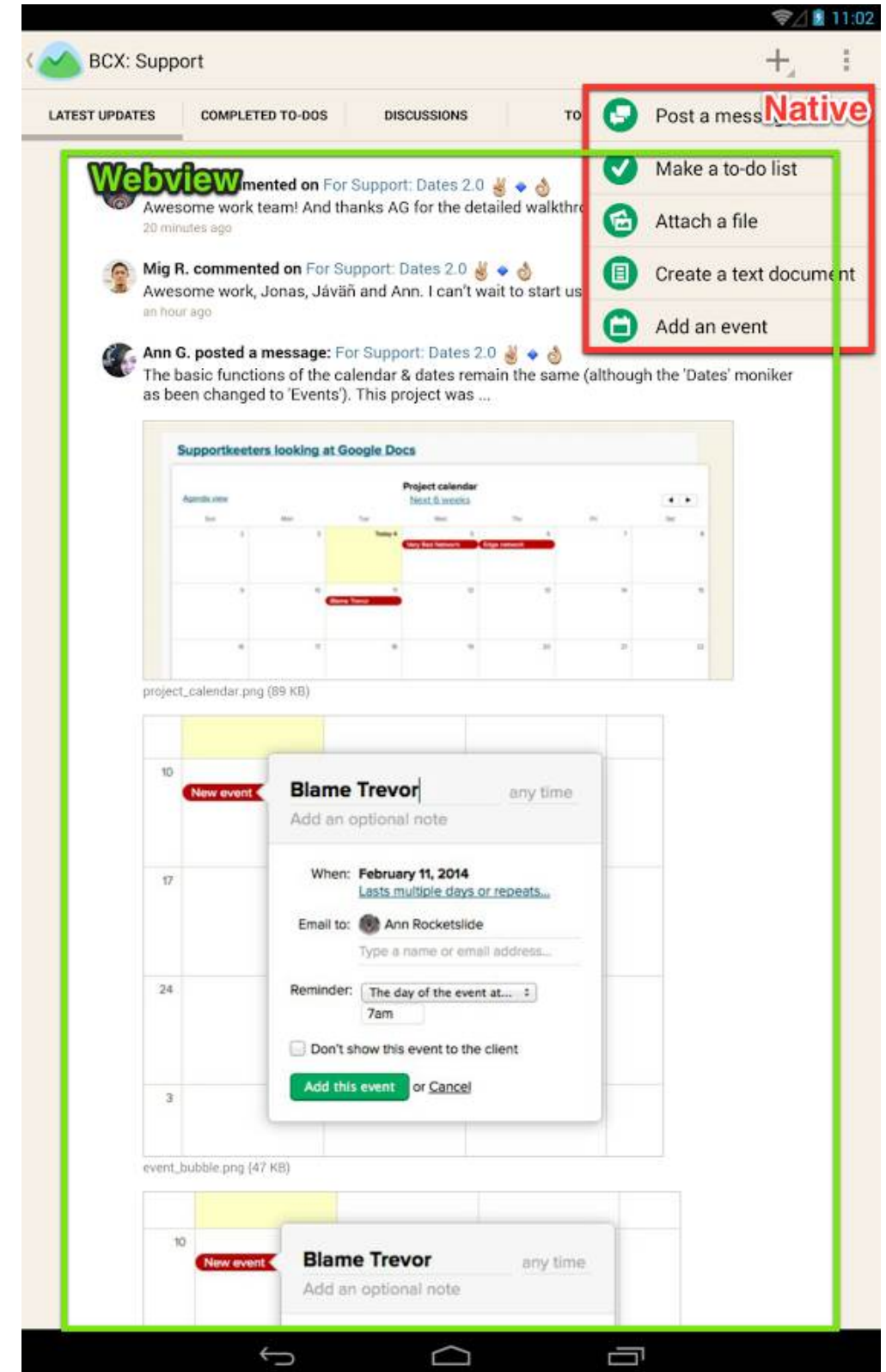
Código en **Javascript**, interfaz en **HTML/CSS**

- Ventajas:
 - Portabilidad
 - 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
 - Algunos APIs no accesibles (p.ej. acceso a la agenda de contactos)
 - No se pueden vender en las tiendas de apps oficiales, ya que son sitios web

Aplicaciones híbridas

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

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



Apache Cordova

- <https://cordova.apache.org/>
- La **plataforma híbrida** más conocida. La versión comercial es **phonegap**



Reusable code across platforms



Support for offline scenarios



Access native device APIs

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
 - JS, HTML: React native, NativeScript
 - C#: Xamarin

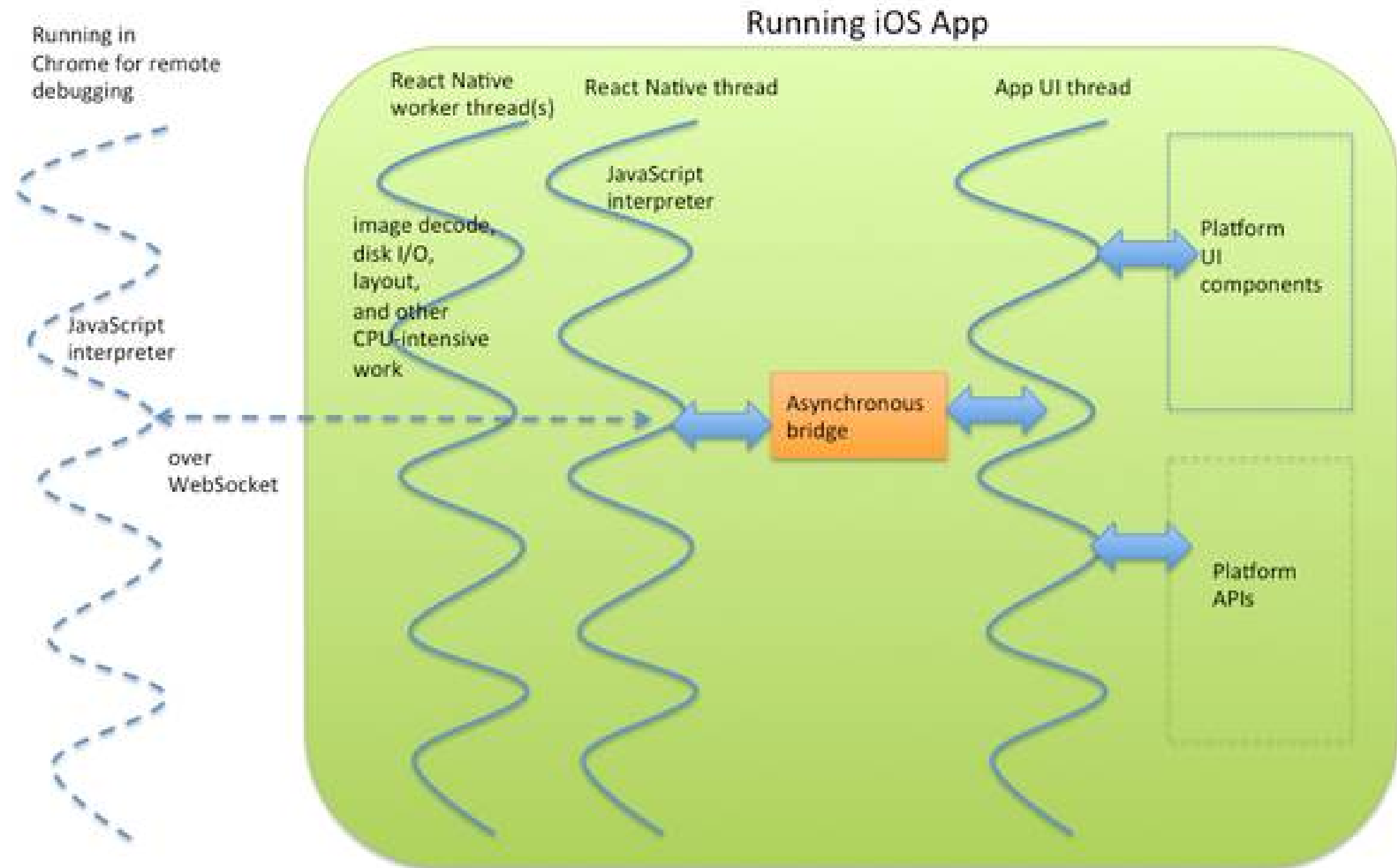


Ejemplo React Native

```
export default class HolaReactNative extends Component {
  saludo() {
    Alert.alert('Hey', 'Qué tal???');
  }
  render() {
    let img = {
      uri: "https://web.ua.es/secciones-ua/images/layout/logo-ua.jpg"
    }
    return (
      <View style={styles.container}>
        <Text style={styles.welcome}>
          Hola React Native!
        </Text>
        <Image source={img}
          style={{width:300, height:200}}
          resizeMode="contain"/>
        <Button onPress={this.saludo} title="Saludar"/>
      </View>
    );
  }
}
```

<https://snack.expo.io/@ottocol/hola-react-native>

Arquitectura de React Native



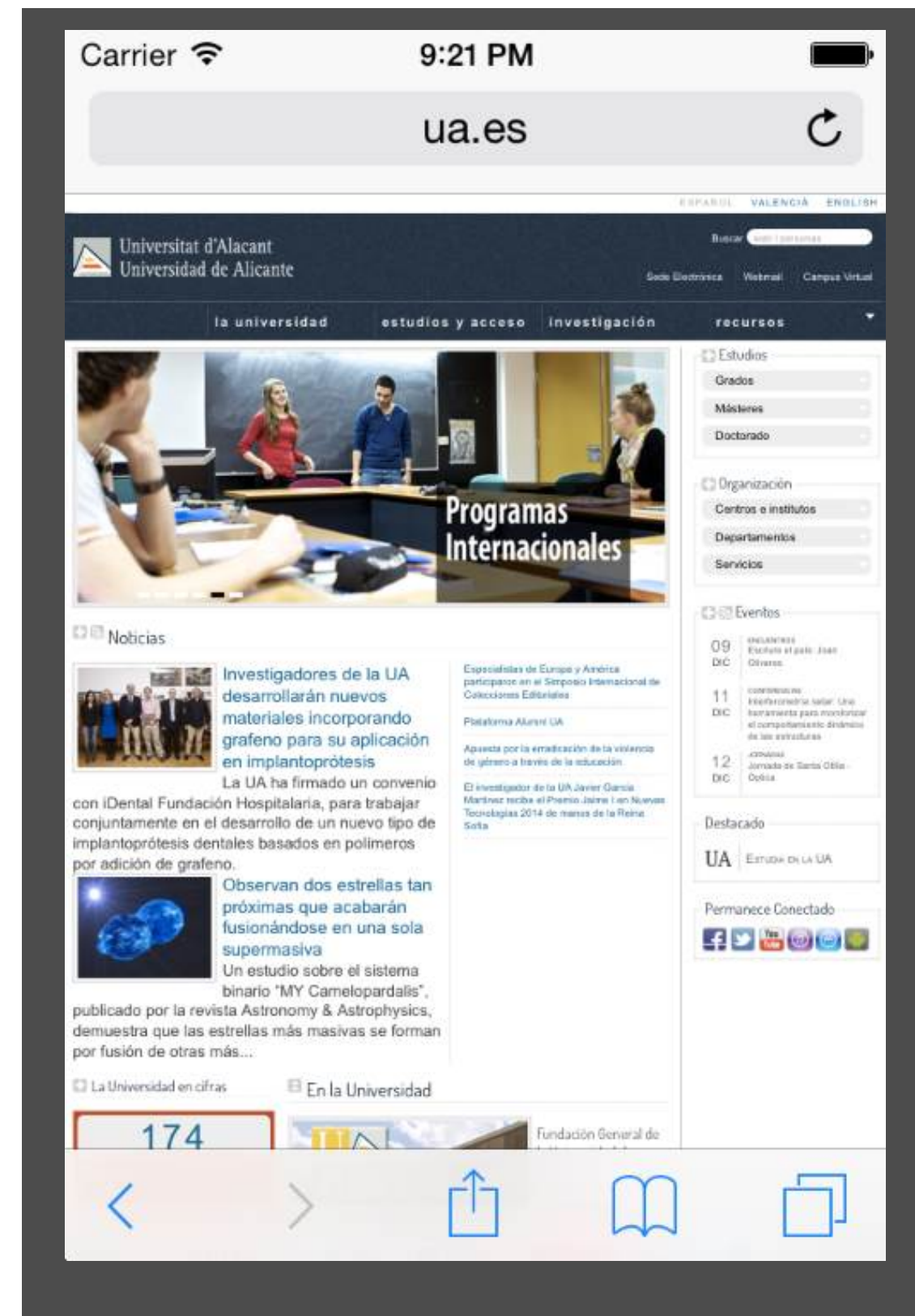
Tema 4: Intro a las apps web en dispositivos móviles

4.2.

Principios generales de
diseño de apps web
móviles

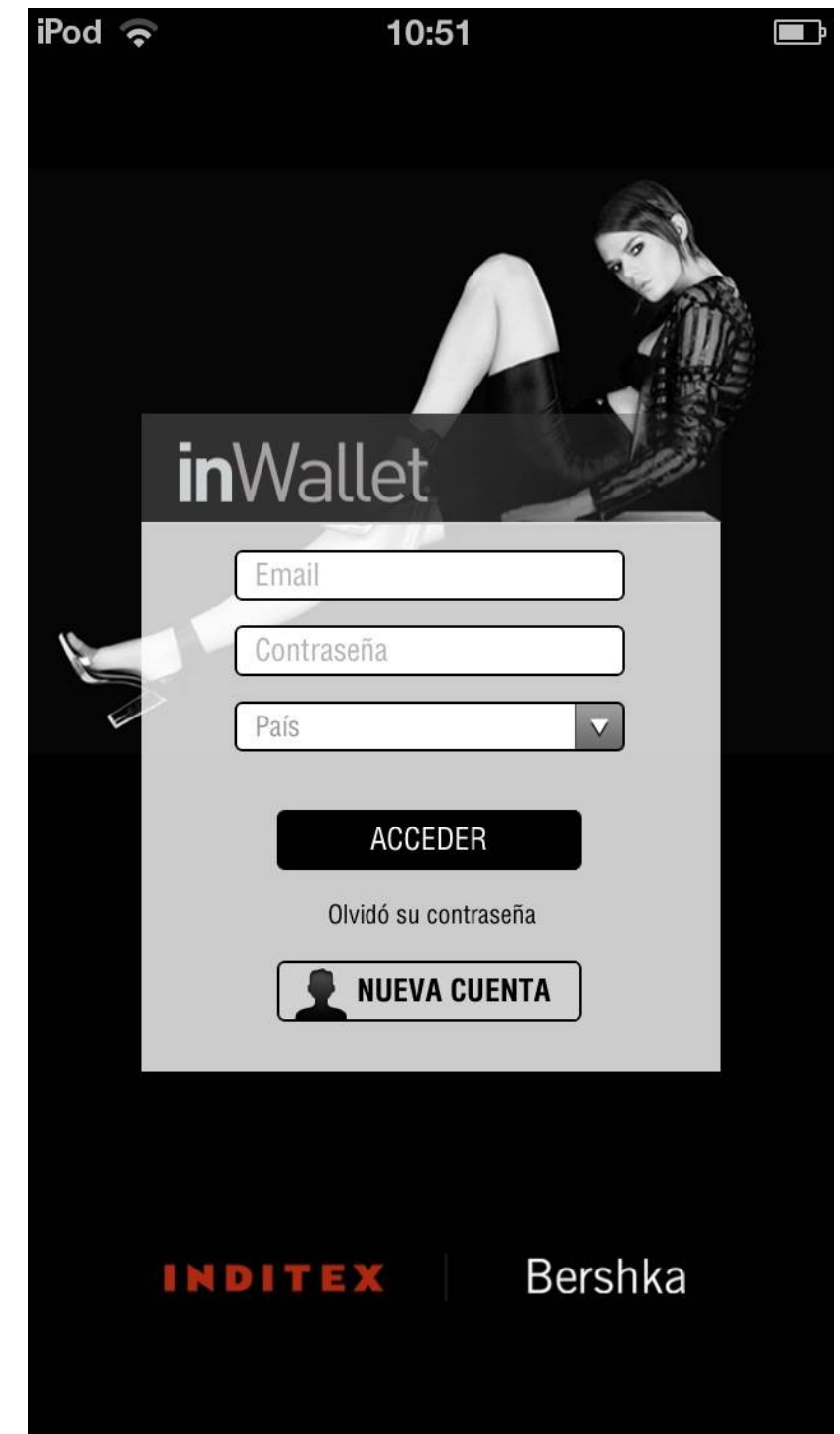
La pantalla

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



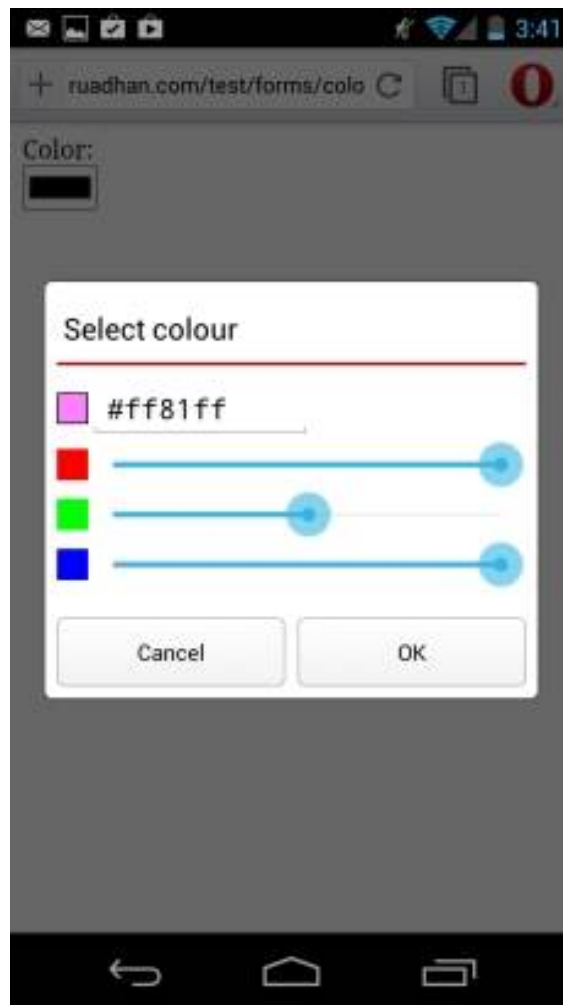
La entrada de datos es tediosa

- 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.

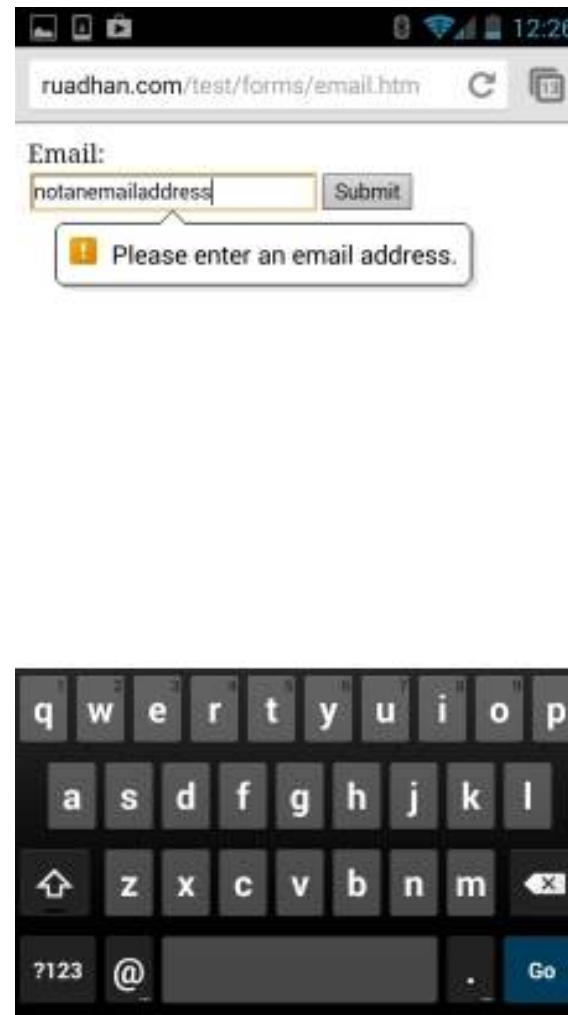


Facilitar la entrada de datos

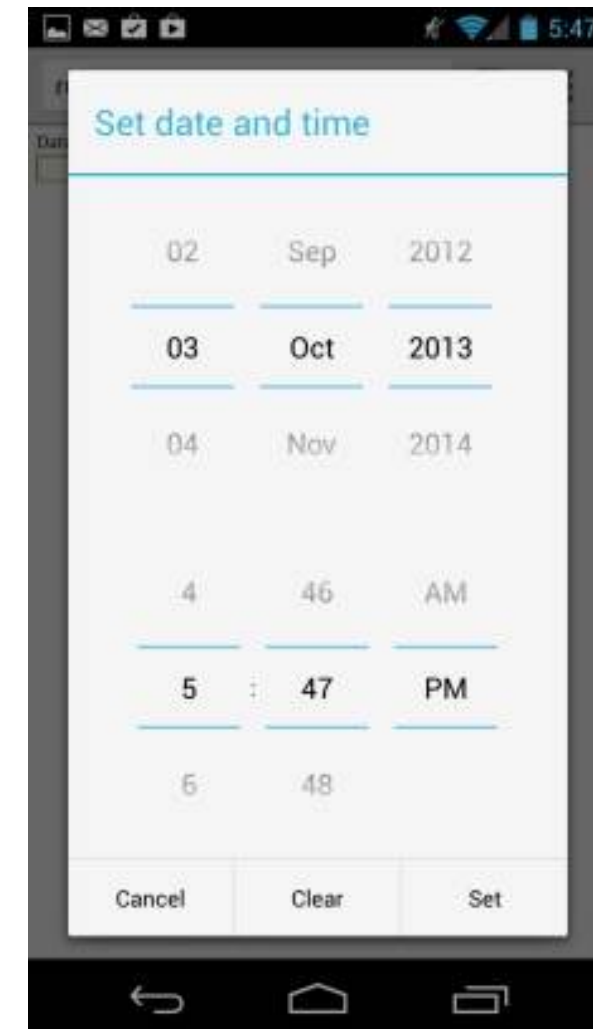
- Usar `<input>` del type más apropiado (number, date, email, color,...)



`<input type="color"/>`



`<input type="email"/>`



`<input type="datetime-local"/>`

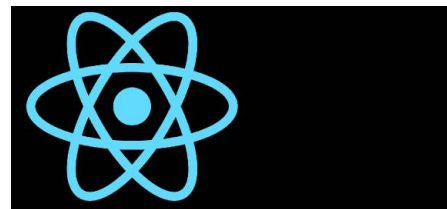
<https://mobiforge.com/design-development/html5-mobile-web-forms-and-input-types>

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



Vs.



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)

<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)

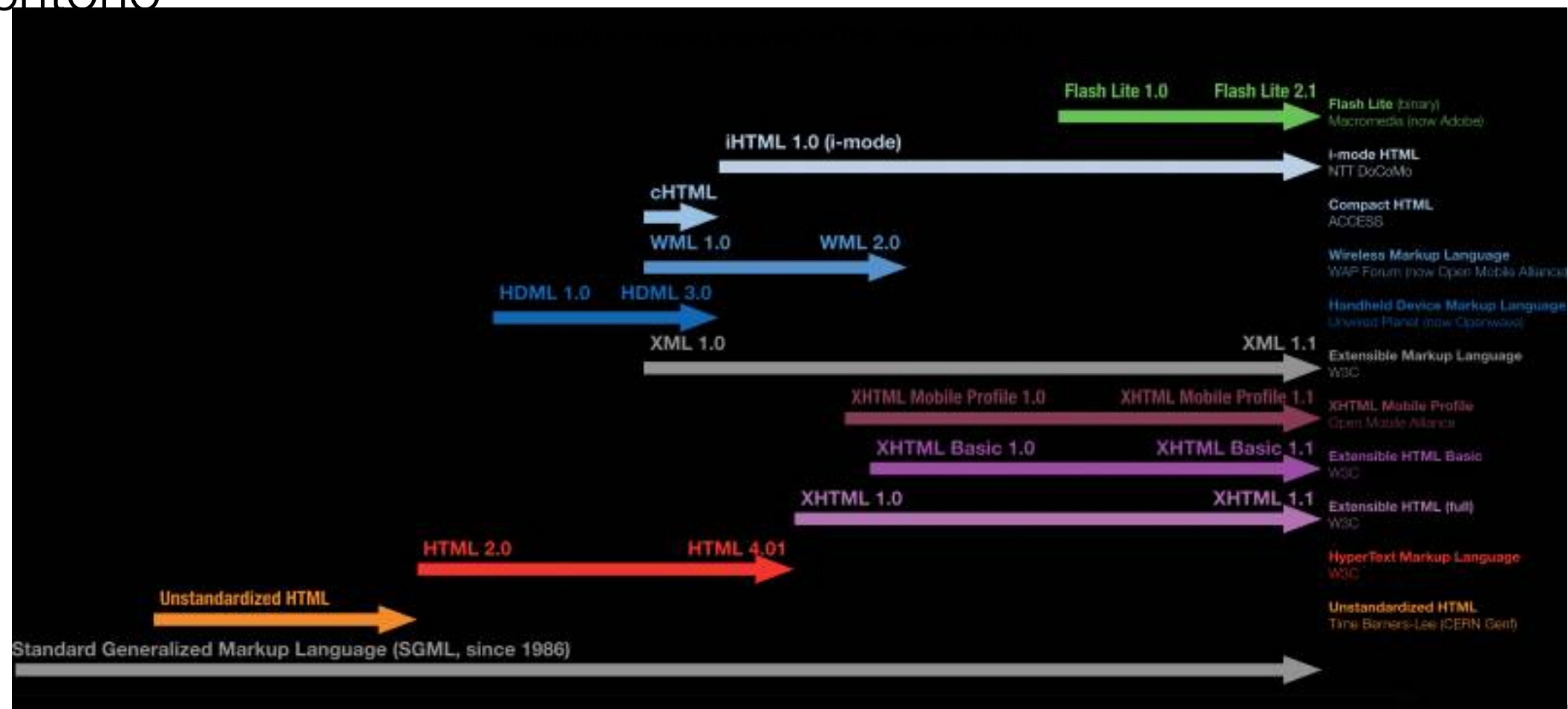
Tema 4: Intro a las apps web en dispositivos móviles

4.3.

HTML y CSS para apps
web móviles

Lenguajes de marcado

- 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





990px

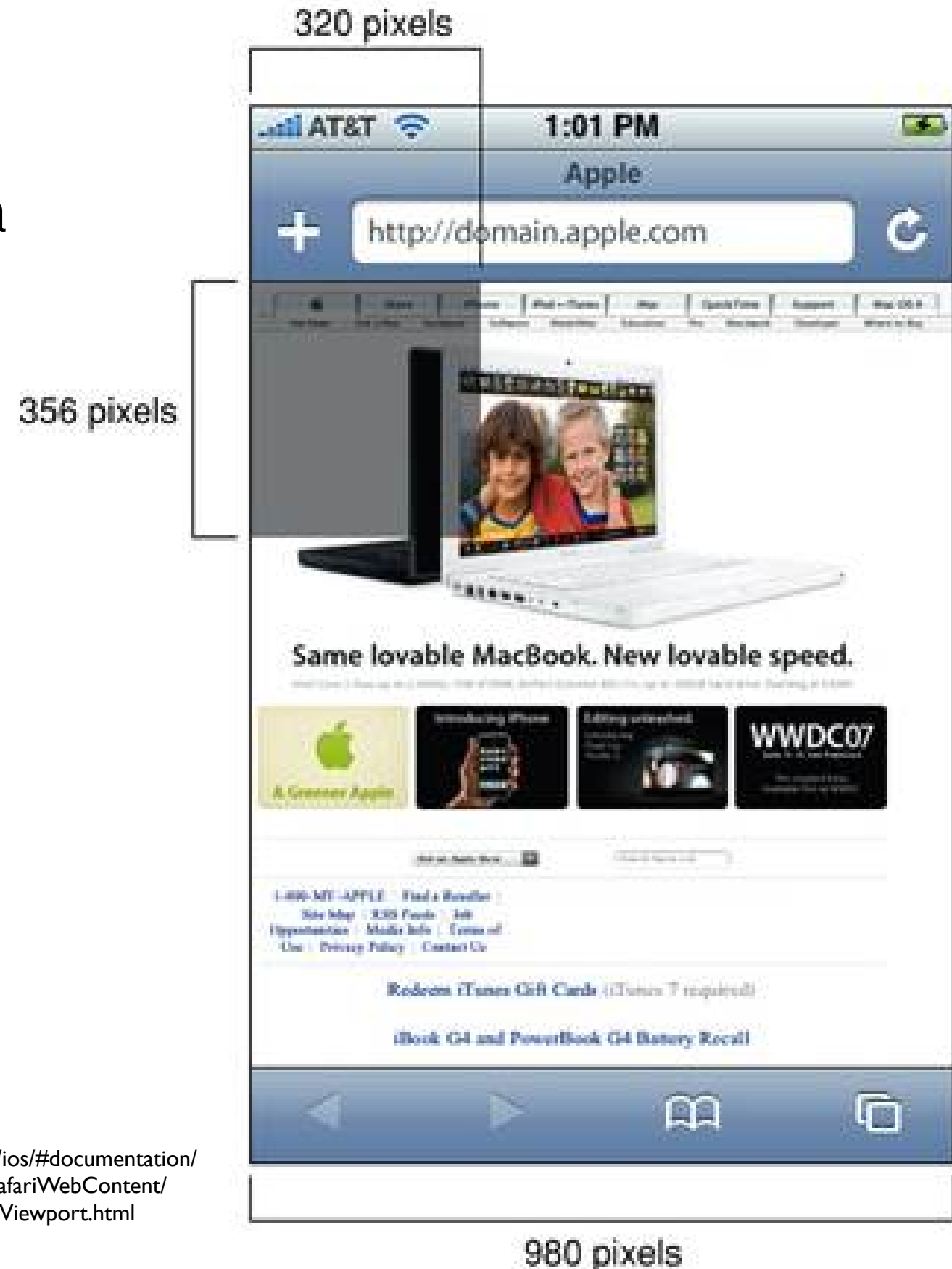




El *viewport*

- 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>
```

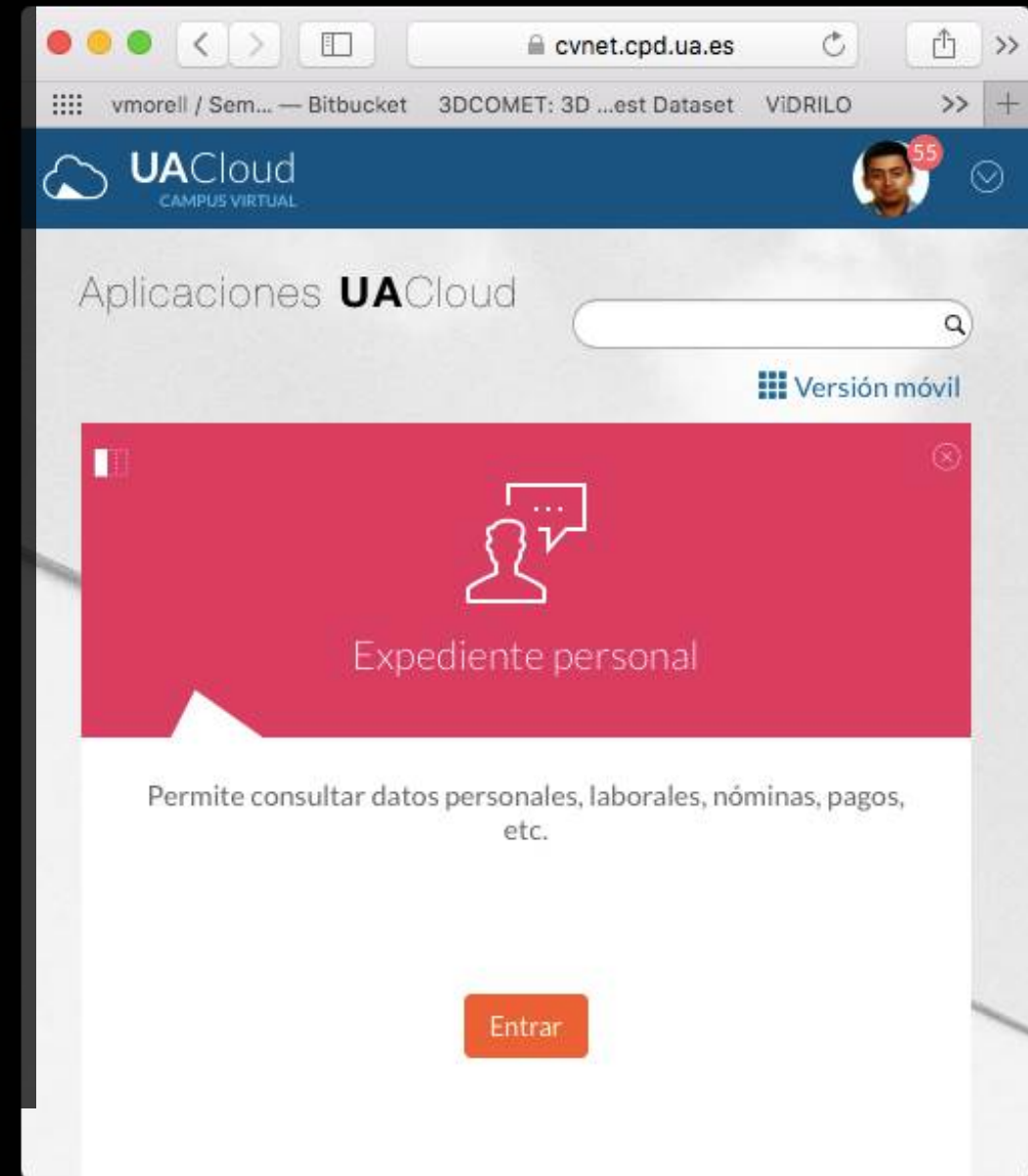
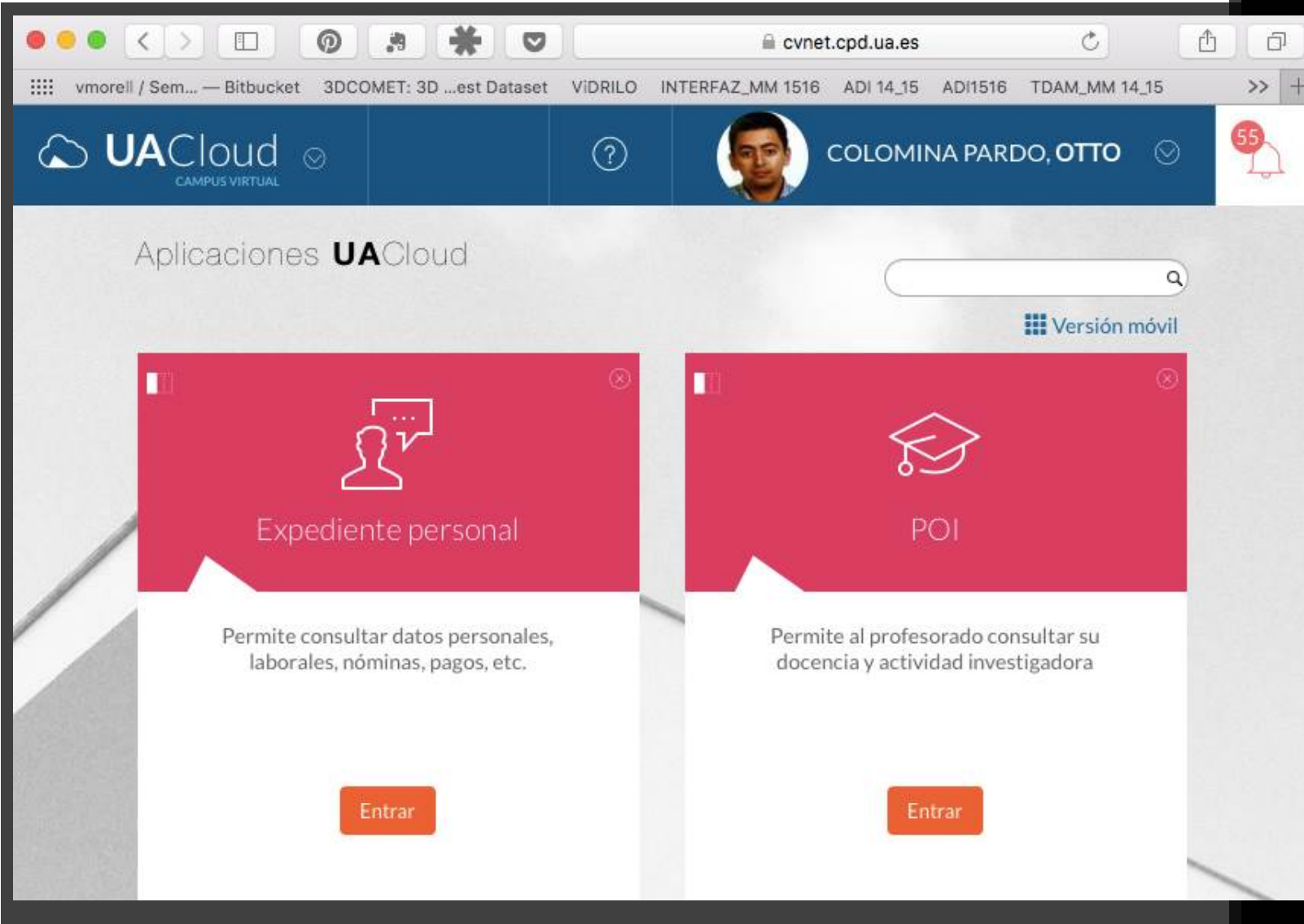
- Para webs “fluidas”, indicar que no se debe hacer escalado

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

“Retina” Web

- “Tipos” de pixels:
 - Device pixels: los del hardware
 - CSS pixels: los que se usan en las reglas CSS (screen.width, screen.height reportan CSS pixels, no físicos)
- Como hemos visto, un dispositivo puede reportar que tiene 320 px de ancho (CSS pixels) cuando en realidad podría tener 640
- `window.devicePixelRatio`: ¿Cuántos píxeles reales es 1 pixel CSS?
- ¿Para qué se usan los pixels “extra”?
 - Para mostrar texto más nítido
 - Podemos usar esta información para servir imágenes de “alta resolución” vs. “resolución estándar”

Responsive web



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" />

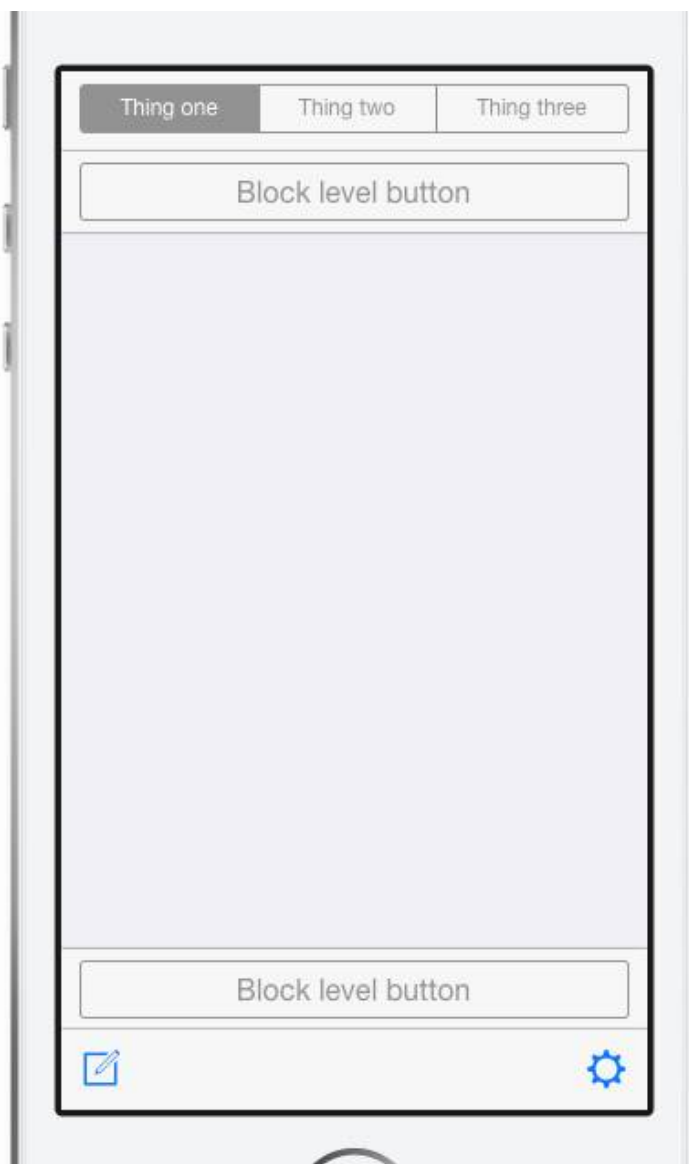
<!-- también se puede poner en el CSS "empotrado" en el HTML -->
<style>
    @media screen and (max-device-width:480px) {
        body {background-color: red;}
    }
</style>
```

Sintaxis de media queries

- **Expresiones:**
 - min- (\geq). p.ej. min-device-width:640px
 - max- (\leq). 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

Frameworks web móviles

- Definimos los elementos de pantalla con etiquetas HTML convencionales, con clases CSS o atributos HTML propios del framework
- En la inicialización, el *framework* les asigna un aspecto (CSS) y un “comportamiento” (Javascript) especiales



```
<!-- Segmented control in standard bar fixed to top -->
<nav class="bar bar-standard">
  <div class="segmented-control">
    <a class="control-item active">Thing one</a>
    <a class="control-item">Thing two</a>
    <a class="control-item">Thing three</a>
  </div>
</nav>
<!-- Block button in standard bar fixed below top bar -->
<div class="bar bar-standard bar-header-secondary">
  <button class="btn btn-block">Block level button</button>
</div>
<!-- Block button in standard bar fixed above the footer -->
<div class="bar bar-standard bar-footer-secondary">
  <button class="btn btn-block">Block level button</button>
</div>
<!-- Icons in standard bar fixed to the bottom of the screen -->
<div class="bar bar-standard bar-footer">
  <a class="icon icon-compose pull-left"></a>
  <a class="icon icon-gear pull-right"></a>
</div>
```

Algunos frameworks



<http://propertycross.com/> una *app* de ejemplo implementada con muchos frameworks distintos, para que sea más sencillo compararlos y evaluarlos (AVISO: ya no está mantenida) 😓

Comparación de frameworks: <http://mobile-frameworks-comparison-chart.com>

Tema 4: Intro a las apps web en dispositivos móviles

4.4.

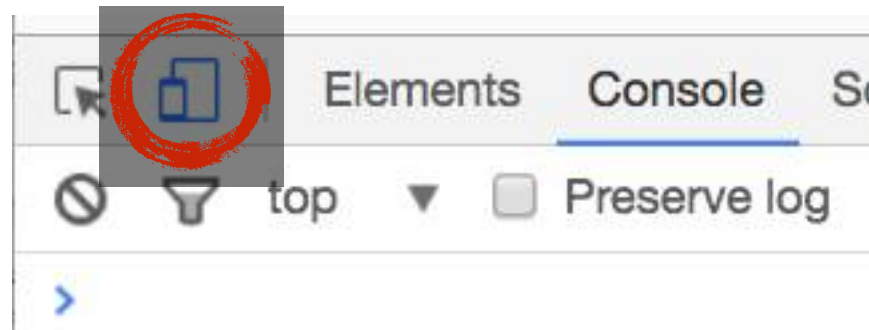
APIs Javascript en
móviles

Algunos APIs útiles en móviles

- Interacción con el hardware
 - **Touch**
 - **Cámara/Micrófono**
 - **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
 - **Payment Request API**
 - **Speech:** síntesis (iOS, Android) y reconocimiento (Android) del habla

Para probar estos APIs (Chrome)

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



- Opción 2: en un emulador
 - <http://emilyporta.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/>

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`

Video/Audio en tiempo real

- Hay un **gran número de APIs bastante recientes** para obtener video/audio de la cámara del usuario, capturar imágenes estáticas, grabar el *stream*, usar esto para comunicación entre navegadores...
- 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

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>
```

- Combinando esto con Javascript podemos:
 - Mostrar la imagen en la página, 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

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

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

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

Capturar imágenes

```
<video id="cam_video">Video</video>
<button id="startbutton">Take photo</button>
<canvas id="canvas" style="display:none">
</canvas>
<img id="photo" alt="The screen capture will appear in this box.">
```

```
function takepicture() {
    var context = canvas.getContext('2d');
    if (width && height) {
        canvas.width = width;
        canvas.height = height;
        context.drawImage(video, 0, 0, width, height);

        var data = canvas.toDataURL('image/png');
        photo.setAttribute('src', data);
    } else {
        clearphoto();
    }
}
```

https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Taking_still_photos

Grabar streams

```
<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>

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>

Algunas referencias

- Charla: *Real time front-end alchemy, or: capturing, playing, altering and encoding video and audio streams, without servers or plugins!*, Soledad Penadés, AtTheFrontend Conference, 2016
 - Video: <https://vimeo.com/168545600>
 - Traspas y demos: https://soledadpenades.com/files/t/2016_rtalchemy/
- Tutorial: *Record Audio and Video with MediaRecorder* <https://developers.google.com/web/updates/2016/01/mediarecorder>
- Repositorio de ejemplos WebRTC: <https://webrtc.github.io/samples/>
- <https://blog.arnellebalane.com/introduction-to-the-shape-detection-api-e07425396861>

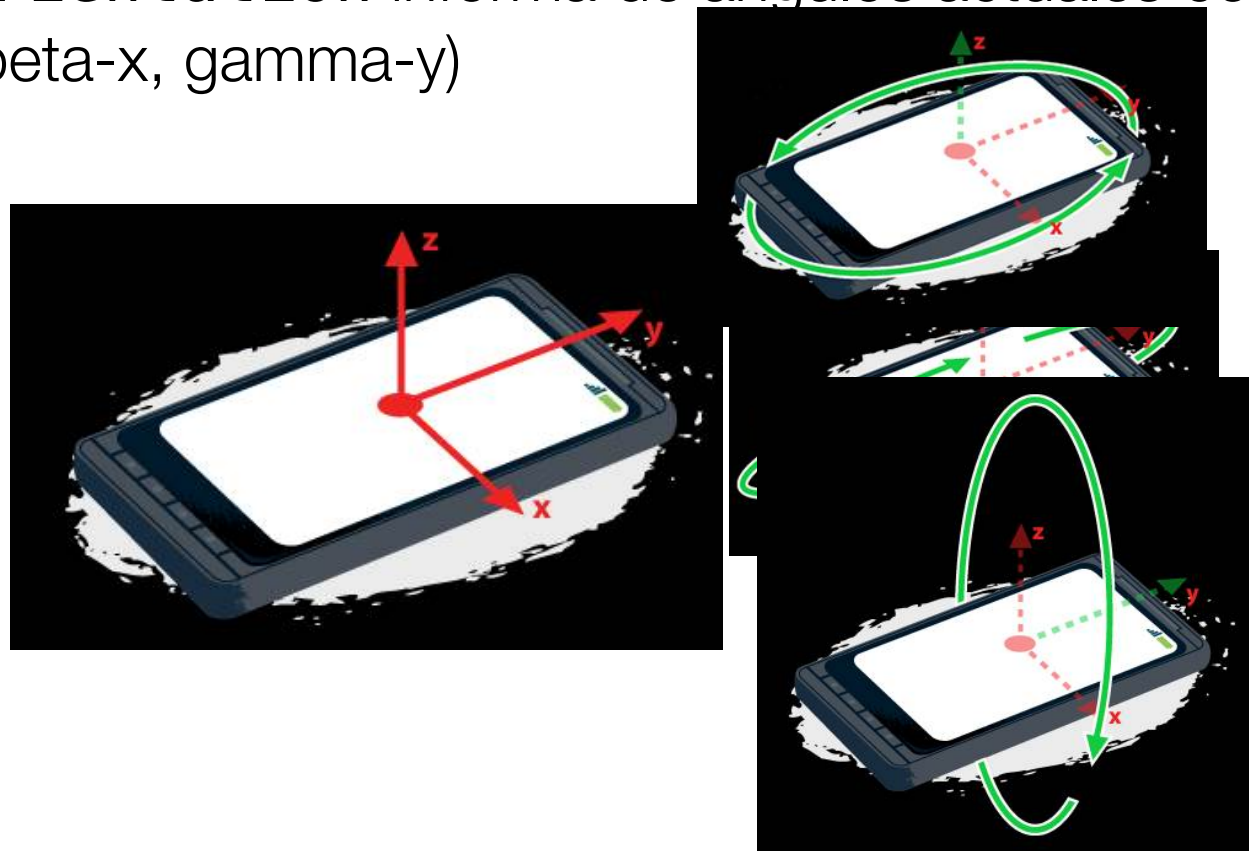
Orientación

- Todos los dispositivos móviles tiene **acelerómetros** para detectar cambios de orientación entre modo vertical (*portrait*) y horizontal (*landscape*)
- Evento **orientationchange**

```
window.addEventListener('orientationchange', function(){
    console.log("Angulo actual: " + screen.orientation.angle)
    console.log("Tipo de orientacion: " + screen.orientation.type)
})
```

Orientación

- 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>

Almacenamiento offline

- **Almacenamiento local:** bases de datos en el cliente
 - localStorage
 - Web SQL
 - IndexedDB
- **Cache** de recursos de red (HTML, CSS, JS) para que la aplicación se pueda seguir usando en la medida de lo posible
 - cache manifest
 - service workers

Bases de datos en el cliente

- Hay dos estándares
 - Web SQL: una base de datos relacional (SQLite), accesible con SQL
 - IndexedDB: una base de datos de pares clave-valor (tipo NoSQL)
- El estándar apoyado oficialmente es **IndexedDB**. Web SQL se ha dejado de mantener, no habrá versiones futuras
- Ahora mismo Web SQL funciona en iOS y Chrome for Android, y en casi todos los navegadores (pero no en Firefox, ni Edge).

Web SQL

- Es **SQL**, inmediatamente familiar para millones de desarrolladores
- API **asíncrono**. Todas las operaciones usan *callbacks*
- **Transaccional**: hay que hacer una transacción para todas las operaciones SQL (incluyendo SELECT)
- Todas las implementaciones existentes usan **SQLite** www.sqlite.com
- Limitaciones: 5Mb por aplicación (salvo en iOS)

Ejemplo Web SQL

```
//abrir BD(nombre, versión, nombre_ampliado, tamaño en bytes)
var bd = openDatabase('Tareas', '1.0', 'tareas pendientes',5*1024*1024);

//crear una tabla: asíncrono y transaccional, como todas las llamadas
bd.transaction(function(trans) {
    //ejecutar SQL(sentencia_SQL, array_con_parámetros)
    trans.executeSql('CREATE TABLE IF NOT EXISTS ' +
        'tarea(ID INTEGER PRIMARY KEY ASC, texto TEXT, ' +
        'prioridad INTEGER)', []);
});

//insertar datos
var texto = prompt("Introduce nueva tarea");
var prioridad = parseInt(prompt("Introduce su prioridad"));
bd.transaction(function(trans) {
    trans.executeSql('INSERT INTO tareas(texto,prioridad) VALUES (?,?)',
        [texto, prioridad])
});
```

Ejemplo modificado: <http://jsbin.com/tabiyuh/edit?html,js,output>

Ejemplo Web SQL (II)

```
//obtener datos
bd.transaction(function(trans) {
    //a executeSQL le podemos pasar un callback que se ejecutará en caso
    //de éxito, y otro en caso de error
    trans.executeSql('SELECT * FROM tareas', [], verDatos, error)
});

//el callback automáticamente recibe la transacción y, si es un SELECT,
//los registros resultado de la consulta
function verDatos(trans, rs) {
    //rows son los registros en sí
    for(var i=0; i<rs.rows.length; i++)
        console.log("Tarea: " + rs.rows.item(i).texto + "\n" +
            "Prioridad: " + rs.rows.item(i).prioridad)
}

function error() {
    alert("Se ha producido un error con la BD");
}
```

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)

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(mostrarPosicion);
function mostrarPosicion(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) {
    ...
}

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});
```