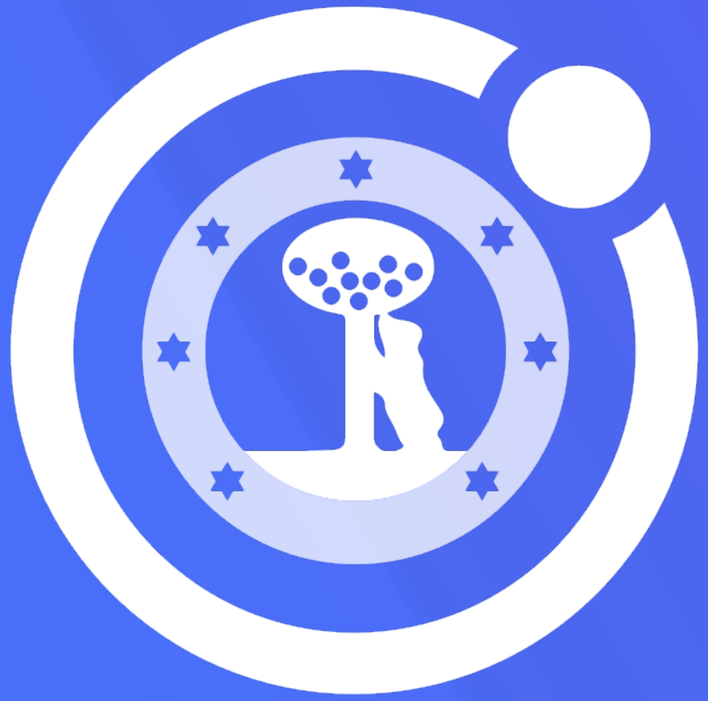
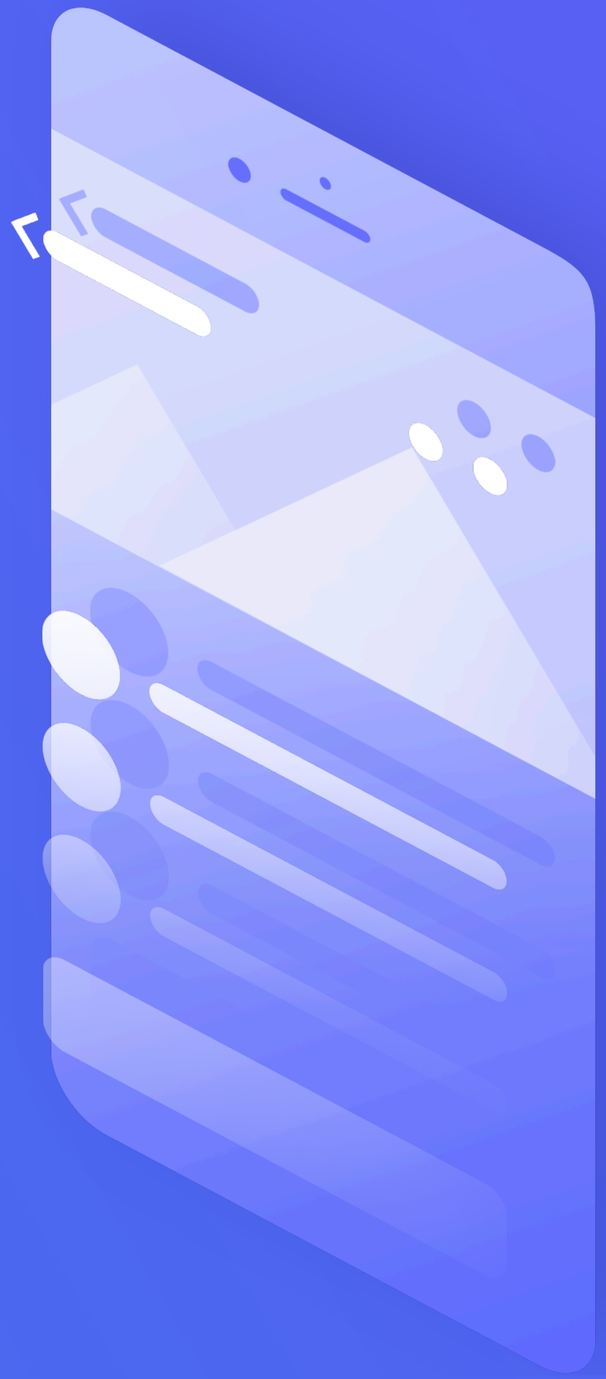


IONICmadrid

ionicmadrid

Taller de Introducción a Capacitor

Rodrigo Fernández
@FdezRomero



Sobre mí

Rodrigo Fernández (@FdezRomero)
CTO y Co-fundador de Savelist

Desarrollador Full-stack Javascript
Usando Ionic desde 2014

SAVELIST


capacitor

Repaso rápido a Capacitor

- **Sucesor de Apache Cordova.**
- API para funcionalidades nativas más comunes.
- Retrocompatibilidad con plugins de Cordova.
- API para Plugins más simple.
- Gestiona dependencias con npm, CocoaPods y Gradle.
- Proyectos estándar de Xcode y Android Studio.
- Sin hooks, se modifica directamente en el proyecto.
- iOS 10+, Android 5.0+ (API 21)

Comprobar dependencias

- Node.js 8.6.0+
  `node -v`
- npm 5.6.0+
  `npm -v`
- Ionic CLI 3.20.0+
  `ionic -v`

Instalar Node.js LTS

`npm i -g npm`

`npm i -g ionic`

Comprobar dependencias (Android)

- Java 8 JDK (no funciona con Java 9)

- Android Studio
  - Android SDK Platform 27+
  - Android SDK Tools 26.0.1+

Comprobar dependencias (macOS)

- Xcode 9

- Xcode CLI Tools
  `xcode-select --install`

- CocoaPods
  `sudo gem install cocoapods`
  `pod repo update`

Crear un proyecto de Ionic 3

```
ionic start taller-capacitor
```

Seleccionar ionic-angular 2/3

Cordova: **No**

Ionic Pro SDK: **No**

Abrir el proyecto de Ionic

Abrir en Visual Studio Code:

```
code taller-capacitor
```

Puedes abrir la carpeta manualmente con cualquier otro editor de código.

Eliminar plugins de Cordova

El starter tabs viene con 2 plugins por defecto que vamos a eliminar, ya que vamos a usar Capacitor.

```
npm rm @ionic-native/splash-screen
@ionic-native/status-bar
```

src/app/app.module.ts

Eliminamos los imports de **StatusBar** y **SplashScreen**:

```
import { StatusBar } from '@ionic-
native/status-bar';
import { SplashScreen } from '@ionic-
native/splash-screen';
```

src/app/app.module.ts

Eliminamos los servicios de los providers y guardamos.

```
providers: [
  StatusBar,
  SplashScreen,
  {provide: ErrorHandler, useClass:
IonicErrorHandler}
]
```

src/app/app.component.ts

Eliminamos los imports de **StatusBar** y **SplashScreen**:

```
import { StatusBar } from '@ionic-
native/status-bar';
import { SplashScreen } from '@ionic-
native/splash-screen';
```

src/app/app.component.ts

Eliminamos los servicios del constructor y su contenido.

```
constructor(platform: Platform, statusBar: StatusBar, splashScreen:
SplashScreen) {
  platform.ready().then(() => {
    // Okay, so the platform is ready and our plugins are available.
    // Here you can do any higher level native things you might need.
    statusBar.styleDefault();
    splashScreen.hide();
  });
}
```

Instalar Capacitor en el proyecto

```
npm install @capacitor/core @capacitor/cli
```

Capacitor queda instalado localmente, con lo que podemos tener diferentes versiones en cada proyecto.

Inicializar Capacitor

```
npx cap init
```

npx es un nuevo comando de npm 5+ para ejecutar comandos instalados localmente en **node_modules**.

cap es el comando del CLI de Capacitor.

Inicializar Capacitor

Nos preguntará lo siguiente:

- **App Name:** TallerCapacitor
- **App Package ID:** es.ionicmadrid.tallercapacitor

Se creará el fichero **capacitor.config.json** con la configuración.

Utilizar el plugin Camera de Capacitor

Capacitor ya dispone de los plugins más usados, sin necesidad de instalarlos. La mayoría tienen una API muy similar a los plugins *core* de Cordova.

El plugin de **Camera** también funciona en la web, gracias a la API de **getUserMedia** de **Media Devices** y **WebRTC**.

src/pages/home/home.ts

Importamos los objetos **Plugins**, **CameraResultType** y **CameraSource** de Capacitor:

```
import { Plugins, CameraResultType, CameraSource }
from '@capacitor/core';
```

Y creamos una constante **Camera** con el contenido de **Plugins.Camera**:

```
const { Camera } = Plugins;
```

src/pages/home/home.ts

Como estamos usando Angular y vamos a recibir una URI con la imagen desde **Camera**, importamos también **DomSanitizer** y **SafeResourceUrl**:

```
import { DomSanitizer, SafeResourceUrl } from '@angular/platform-browser';
```

DomSanitizer es un servicio con métodos a los que podemos pasarles URLs y otros valores que Angular normalmente bloquearía por ser potencialmente inseguros para limpiarlos y permitir su uso. **SafeResourceUrl** es el tipo que devuelve la función **bypassSecurityTrustResourceUrl()**.

No tiene nada que ver con Capacitor y es exclusivo de Angular, pero lo necesitamos para poder mostrar la imagen que tomemos más adelante en nuestro ejemplo.

src/pages/home/home.ts

Declaramos la propiedad **image** en la clase **HomePage**, que contendrá la URI que apunte a la imagen de la cámara:

```
export class HomePage {
  image: SafeResourceUrl;
```

E injectamos el servicio **DomSanitizer** en el constructor:

```
constructor(public navCtrl: NavController, public
sanitizer: DomSanitizer) {
```

src/pages/home/home.ts

Por último creamos la función **takePicture**, que usaremos para abrir la cámara y obtener la URI de la imagen. Este ejemplo usa **async/await** para esperar y obtener el resultado de la promesa de **Camera.getPhoto()**.

```
async takePicture() {
  const image = await Camera.getPhoto({
    quality: 90,
    allowEditing: true,
    resultType: CameraResultType.Uri,
    source: CameraSource.Camera
  });

  this.image = this.sanitizer.bypassSecurityTrustResourceUrl(image &&
image.webPath);
}
```

Camera.getPhoto()

En este ejemplo hemos utilizado **resultType: CameraResultType.Uri**, que devuelve una URI a la imagen y tiene mejor rendimiento.

También es posible usar **CameraResultType.Base64** si queremos que nos devuelva un *string* con el contenido de la imagen para enviarlo a un servidor.

Con **source: CameraSource.Camera** especificamos que origen de la imagen sea la cámara, pero también podemos usar **Photos** o **Prompt**.

Todas las opciones del plugin **Camera** están disponibles en la documentación de Capacitor.

src/pages/home/home.html

En el template de la página reemplazamos el contenido de **ion-content** por el siguiente:

```
<ion-content padding>
  <img [src]="image" class="picture">
  <button ion-button
color="primary" (click)="takePicture()">Take
Picture</button>
</ion-content>
```

src/pages/home/home.scss

Opcionalmente, hacemos que la imagen que viene de la cámara ocupe todo el ancho de la pantalla añadiendo:

```
page-home {
  .picture {
    display: block;
    width: 100%;
  }
}
```

src/index.html

Como estamos usando Capacitor en lugar de Cordova, eliminamos el script que carga **cordova.js**:

```
<!-- cordova.js required for cordova apps (remove if not needed)
-->
// <script src="cordova.js"></script>
```

Y añadimos el siguiente script, que carga los componentes de **@ionic/pwa-elements** necesarios para mostrar la UI de la cámara en nuestra PWA:

```
<script src='https://unpkg.com/@ionic/pwa-elements@0.0.11/dist/
ionicpwaelements.js'></script>
```

Ejecutar la PWA

```
ionic serve
```

Ionic hará un build de nuestra app, la servirá en **localhost:8100** y abrirá nuestro navegador por defecto.

Hacer una foto con la cámara

En el tab **Home**, pulsamos el botón **Take Picture** que hemos añadido antes y se abrirá la interfaz de la cámara según la plataforma.

Es posible que nos pida permiso para acceder a la cámara y al sistema de ficheros la primera vez.

Si hacemos una foto y la aceptamos, vemos que aparece el resultado al principio de **HomePage**.

Añadir la plataforma Android

```
npx cap add android
```

Capacitor creará un proyecto nativo en la carpeta `android` e instalará las dependencias necesarias.

Añadir la plataforma iOS (sólo macOS)

```
npx cap add ios
```

Capacitor creará un proyecto nativo en la carpeta `ios` e instalará las dependencias necesarias.

Actualizar el código de nuestra app

Generamos un build de nuestra app en **www**:

```
ionic build
```

Actualizamos **www** con los proyectos nativos y los plugins instalados, si los hay:

```
npx cap sync
```

Ejecutar la app en Android

```
npx cap open android
```

Capacitor abrirá Android Studio, que lanzará Gradle y puede que descargue algunas dependencias. Cuando termine, pulsamos el botón **Run** y seleccionamos un AVD para el emulador, o un dispositivo Android que tengamos conectado.

Ejecutar la app en iOS

```
npx cap open ios
```

Capacitor abrirá el workspace de Xcode. Pulsamos sobre el proyecto **App** en el navegador y seleccionamos un equipo de desarrollo para el certificado. Luego elegimos un dispositivo iOS del simulador o uno que tengamos conectado y pulsamos el botón de **Run**.

Utilizar el plugin Device de Cordova

En Capacitor también podemos instalar plugins de terceros, incluso plugins de Cordova* gracias a su compatibilidad.

Aunque Capacitor ya incluye el plugin **Device**, vamos a instalar **cordova-plugin-device** y a usarlo con **Ionic Native 5**.

* No todos los plugins de Cordova se han probado con Capacitor. Si encuentras un plugin que no funciona bien, por favor busca o abre un issue en su GitHub.

Instalar cordova-plugin-device y Ionic Native 5

Instalamos el plugin, el paquete base de Ionic Native en *beta* (v5) y el específico para el plugin de **Device**:

```
npm install cordova-plugin-device
@ionic-native/core@beta @ionic-native/
device@beta
```

src/app/app.module.ts

Importamos el servicio **Device** de Ionic Native. A partir de la v5 hay que añadir **/ngx** al final de la ruta para indicar que queremos el servicio de Angular:

```
import { Device } from '@ionic-native/device/ngx';
```

Y lo añadimos a los providers:

```
providers: [
  Device,
  {provide: ErrorHandler, useClass: IonicErrorHandler}
]
```

src/pages/about/about.ts

Importamos el servicio **Device** de Ionic Native:

```
import { Device } from '@ionic-native/device/ngx';
```

Y lo injectamos en el constructor de **AboutPage**:

```
constructor(public navCtrl: NavController, public device: Device)
{
```

Observa que aunque es un plugin de Cordova, no hace falta esperar al evento **deviceready** ni a que resuelva **platform.ready()**. El plugin está disponible inmediatamente gracias a Capacitor.

src/pages/about/about.html

En el template de la página añadimos lo siguiente dentro de **ion-content**:

```
<ion-content padding>
  <p>Model: {{ device?.model }}</p>
  <p>UUID: {{ device?.uuid }}</p>
</ion-content>
```

Actualizar el código de nuestra app

Generamos un nuevo build de nuestra app en **www**:

```
ionic build
```

Actualizamos **www** con los proyectos nativos y el nuevo plugin que hemos instalado:

```
npx cap sync
```

Ejecutar la app finalizada

Abrir proyecto Android:

```
npx cap open android
```

Abrir proyecto iOS:

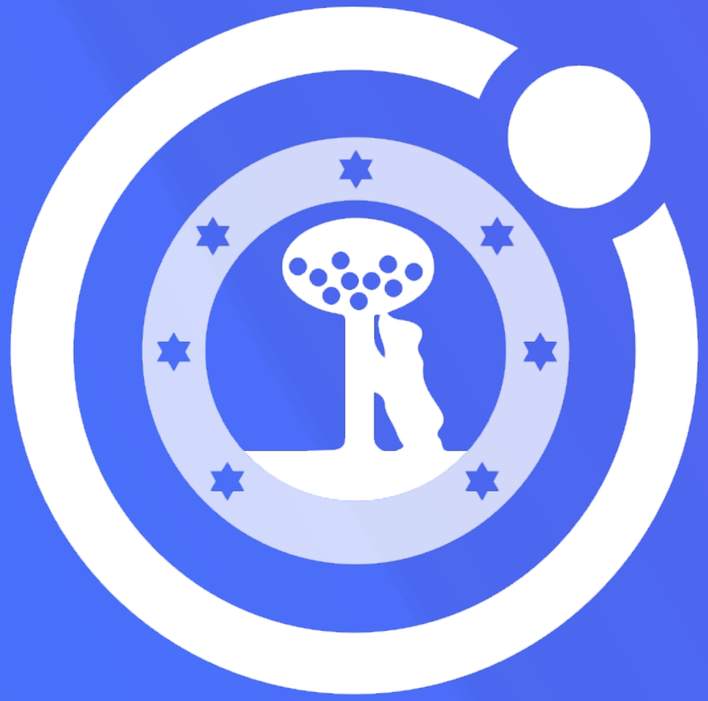
```
npx cap open ios
```

Al ejecutar la app, en el tab **About** deberíamos poder ver el modelo y UUID de nuestro dispositivo.

Repositorio en Github

bit.ly/tallercapacitor

Gracias



IONICmadrid