



UA.MASTER MOVILES

MÁSTER UNIVERSITARIO EN DESARROLLO DE SOFTWARE
PARA DISPOSITIVOS MÓVILES

PROGRAMACIÓN HIPERMEDIA PARA DISPOSITIVOS MÓVILES

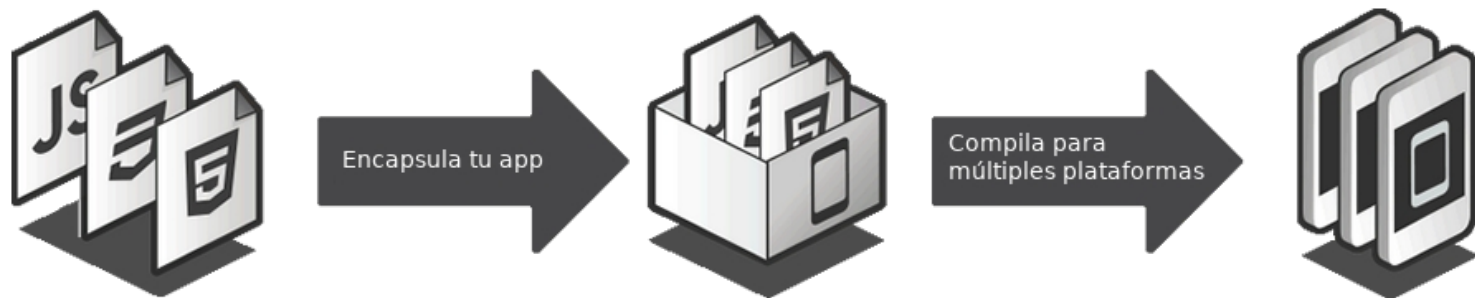
Desarrollo de aplicaciones híbridas con
Ionic + Capacitor

1. Introducción
2. Instalación
3. Interfaz de línea de comandos
4. Gestión de plataformas
5. Compilación y ejecución
6. Plugins
7. Publicar una App

¿QUÉ SON LAS APLICACIONES HÍBRIDAS?



- Esquema básico de funcionamiento:
 - 1) Desarrollar la aplicación usando estándares Web.
 - 2) Combinar la aplicación Web con una App nativa, esto nos dará acceso a las características nativas de los dispositivos móviles.
 - 3) Configurar y compilar la aplicación para cada una de las plataformas para las que queramos generar la aplicación nativa.



CAPACITOR

- Actualmente Ionic integra por defecto Capacitor al crear un nuevo proyecto:

```
$ ionic capacitor
```

- Si al ejecutar un comando nos aparece el siguiente error:

```
[ERROR] capacitor integration is disabled in the default  
project.
```

- Podemos ejecutar la siguiente instrucción para solucionarlo:

```
$ ionic integrations enable capacitor
```

- Opciones de la interfaz de línea de comandos:

`$ ionic capacitor add` → Añadir plataformas

`$ ionic capacitor build` → Compilar

`$ ionic capacitor run` → Ejecutar en un dispositivo

`$ ionic capacitor sync` → Copiar y actualizar las plataformas

`$ ionic capacitor open` → Abre el IDE de Android Studio o Xcode

`$ ionic capacitor update` → Actualiza las plataformas instalados

- Para obtener más información sobre algún comando añadimos la opción “`--help`”, por ejemplo:

`$ ionic capacitor add --help`

- Para añadir nuevas plataformas ejecutamos:

```
$ ionic capacitor add [<plataforma>]
```

- **IMPORTANTE:** Es necesario instalar cada SDK por separado y que Capacitor lo pueda encontrar.

	Mac	Linux	Windows
Android	✓	✓	✓
iOS	✓		

- Para instalar y configurar Android:
<https://ionicframework.com/docs/developing/android>
- Para instalar y configurar iOS:
<https://ionicframework.com/docs/developing/ios>

Nuevas carpetas añadidas al proyecto:

- `android/` → Código del proyecto Android.
- `ios/` → Código del proyecto iOS.
- `resources/` → Iconos y *splashscreen* específicos de las plataformas.
- **`src/`** → Código fuente principal de nuestra aplicación.
- `www/` → Código web compilado.

IMPORTANTE:

- Al compilar se sigue un orden: `src` → `www` → `android`
- **No modificar** las carpetas de los proyectos directamente.

- Para compilar y ejecutar para una plataforma usamos:

```
$ ionic capacitor run <platform>
```

- Por ejemplo:

```
$ ionic capacitor run android
```

- Opcionalmente podemos añadir:

```
--livereload 0 -l
```

- Para que esta opción funcione correctamente es necesario añadir también la opción “--external”:

```
$ ionic capacitor run android -l --external
```

- Para ejecutar una plataforma Android, además de tener instalado Android Studio y el SDK, es necesario disponer de la última versión de **Java**.

- Para que el CLI de Capacitor la encuentre:

```
JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
```

- Si tenemos varias versiones instaladas:

```
$ sudo update-alternatives --config java
```

```
There are 3 choices for the alternative java (providing /usr/bin/java).
```

Selection	Path

* 0	/usr/lib/jvm/java-17-openjdk-amd64/bin/java
1	/usr/lib/jvm/java-11-openjdk-amd64/bin/java
2	/usr/lib/jvm/java-8-oracle/jre/bin/java

- Posteriormente, al ejecutar el comando `run` nos dará a elegir entre los emuladores configurados y los dispositivos reales conectados.
- Para elegir uno es importante comprobar la versión de Android de la plataforma en el fichero `“android/variables.gradle”`:

```
minSdkVersion = 22  
compileSdkVersion = 32
```

Para inspeccionar los mensajes de consola de una aplicación emulada o nativa tenemos tres opciones:

- Abrir la siguiente dirección en una pestaña de Google Chrome:

```
chrome://inspect
```

- Abrir el panel LogCat dentro de Android Studio.
- Ejecutar el comando “adb logcat” (carpeta `platform-tools` del SDK).

- Cuando se pulsa el botón físico *Back* de los dispositivos Android, se desapila la pantalla actual. Pero, si **no** hay ninguna pantalla anterior, no ocurrirá nada.
- Para solucionar esto tenemos que instalar:

```
npm install @capacitor/app  
npx cap sync
```

- Y escribir el siguiente código:

```
import { Platform } from '@ionic/angular';  
import { App } from '@capacitor/app';  
  
@Component({ ... })  
export class HomePage implements OnInit {  
  
  constructor(private platform: Platform) {  
    this.platform.backButton.subscribeWithPriority(-1, () => {  
      App.exitApp();  
    });  
  }  
}
```

Más info en: <https://ionicframework.com/docs/developing/hardware-back-button>

PLUGINS

- 31 *plugins* oficiales:

<https://capacitorjs.com/docs/apis>

Action Sheet, App Launcher, App, Background Runner, Barcode Scanner, Browser, Camera, Clipboard, Cookies, Device, Dialog, Filesystem, Geolocation, Google Maps, Haptics, Http, InAppBrowser, Keyboard, Local Notifications, Motion, Network, Preferences, Push Notifications, Screen Orientation, Screen Reader, Share, Splash Screen, Status Bar, Text Zoom, Toast, Watch

- *Plugins* de la comunidad para Capacitor:

<https://github.com/capacitor-community/>

- Awesome Cordova plugins:

<https://capacitorjs.com/docs/plugins/cordova>

Pasos para su uso:

1. Salvo el *plugin* básico de Capacitor, el resto hay que **instalarlos**:

```
npm install @capacitor/<nombre-plugin>
npx cap sync
```

2. Si usamos “`--livereload`” hay que **reiniciar** la ejecución.
3. Esto nos añadirá nuevas clases y métodos a Ionic que podremos usar

desde TypeScript:

1. Importar.
2. Seguir las instrucciones del *plugin*.

- Las funciones proporcionadas por el *plugin* pueden ser de dos tipos:
 - 1) Obtener un dato de manera puntual.
 - 2) *Listener* de eventos.
- Ejemplos:

```
import { Network } from '@capacitor/network';
```

```
@Component({...})
```

```
export class HomePage
```

```
{
```

```
  constructor() {
```

```
    const obtenerCurrentNetworkStatus = async () => {
```

```
      const status = await Network.getStatus();
```

```
      console.log('Estado1:', status);
```

1)

```
    };
```

```
    obtenerCurrentNetworkStatus();
```

```
    Network.getStatus().then( status => { console.log('Estado2:', status ); });
```

2)

```
    Network.addListener('networkStatusChange', status => {
```

```
      console.log('Cambio de estado:', status);
```

```
    });
```

```
  }
```

```
}
```

Dos opciones
equivalentes
para obtener un dato
de manera puntual

Listener de eventos

- Para optimizar el rendimiento, los *listeners* de eventos de los *plugins* de Capacitor se ejecutan **fuera de las zonas de Angular**.
- Por este motivo, para asegurarnos de que Angular detecta el cambio en un valor, tenemos que gestionar las respuestas dentro de un bloque **NgZone.run**:

```
import { NgZone } from '@angular/core';

...

networkStatus: string = "?"

constructor(private ngZone: NgZone)
{
  Network.addListener("networkStatusChange", (status) => {
    this.ngZone.run(() => {
      this.networkStatus = status.connected ? "Online" : "Offline";
    });
  });
}
```

Más info en: <https://capacitorjs.com/docs/guides/angular>

- El *plugin* de Capacitor nos proporciona una API **web** con información base para determinar:
 - El tipo de plataforma en la que se está ejecutando la App.
 - Si la plataforma es nativa o no.
 - Si un *plugin* o característica *hardware* está disponible.

```
import { Capacitor } from '@capacitor/core';
```

```
ptfName = Capacitor.getPlatform();
```

```
isNative = Capacitor.isNativePlatform();
```

```
isAvailable = Capacitor.isPluginAvailable('Camera');
```

Métodos
estáticos

Más info en: <https://capacitorjs.com/docs/core-apis/web>

- El *plugin* “*device*” proporciona más información sobre el dispositivo, como su sistema operativo, versión, fabricante, etc.
- Primero tenemos que instalarlo:

```
npm install @capacitor/device  
npx cap sync
```

- Esto nos dará acceso a los siguientes métodos:

```
import { Device } from '@capacitor/device';  
  
...  
  
constructor() {  
  Device.getInfo().then( result => { this.deviceInfo = result; })  
  
  Device.getId().then( result => { this.deviceId = result; })  
  
  Device.getBatteryInfo().then( result => { this.battery = result; })  
}
```

Estos métodos devuelven:

- `Device.getId()`
 - ➔ `DeviceId: { uuid: string }`
- `Device.getBatteryInfo()`
 - ➔ `BatteryInfo: {
 batteryLevel: number, // Valor entre 0 y 1
 isCharging: boolean
}`
- `Device.getInfo()`
 - ➔ `DeviceInfo: {
 name: string,
 operatingSystem: OperatingSystem,
 osVersion: string,
 manufacturer: string,
 isVirtual: boolean,
 // ...
}`

`['ios', 'android',
'windows', 'mac',
'unknown']`

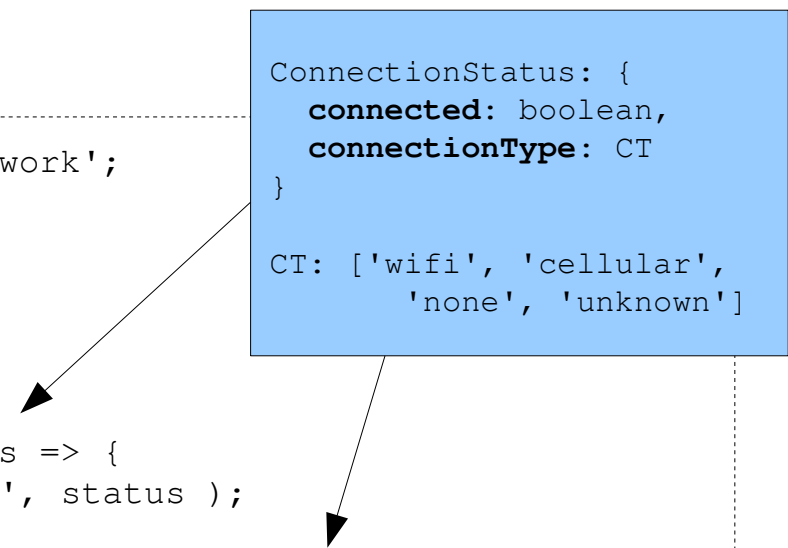
Más info en: <https://capacitorjs.com/docs/apis/device>

- Proporciona información sobre la conexión a Internet y su tipo.
- En primer lugar tenemos que instalarlo:

```
npm install @capacitor/network  
npx cap sync
```

- Uso:

```
import { Network } from '@capacitor/network';  
  
@Component({...})  
export class HomePage  
{  
  constructor()  
  {  
    Network.getStatus().then( status => {  
      console.log('Estado actual:', status );  
    });  
    Network.addListener('networkStatusChange', status => {  
      console.log('Cambio de estado:', status);  
    });  
    Network.removeAllListeners();  
  }  
}
```



```
ConnectionStatus: {  
  connected: boolean,  
  connectionType: CT  
}  
  
CT: ['wifi', 'cellular',  
     'none', 'unknown']
```

Más info en: <https://capacitorjs.com/docs/apis/network>

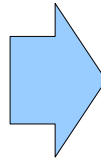
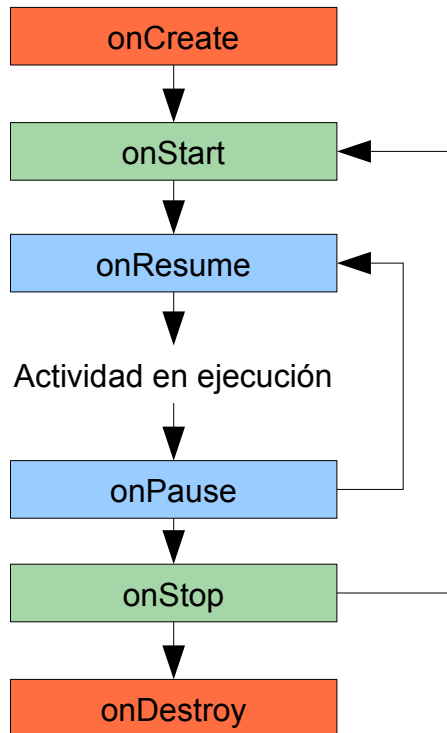
Este *plugin*, que ya hemos visto antes para el *back button*, también nos permite controlar los estados de la aplicación mediante la captura de eventos:

```
import { App } from '@capacitor/app';

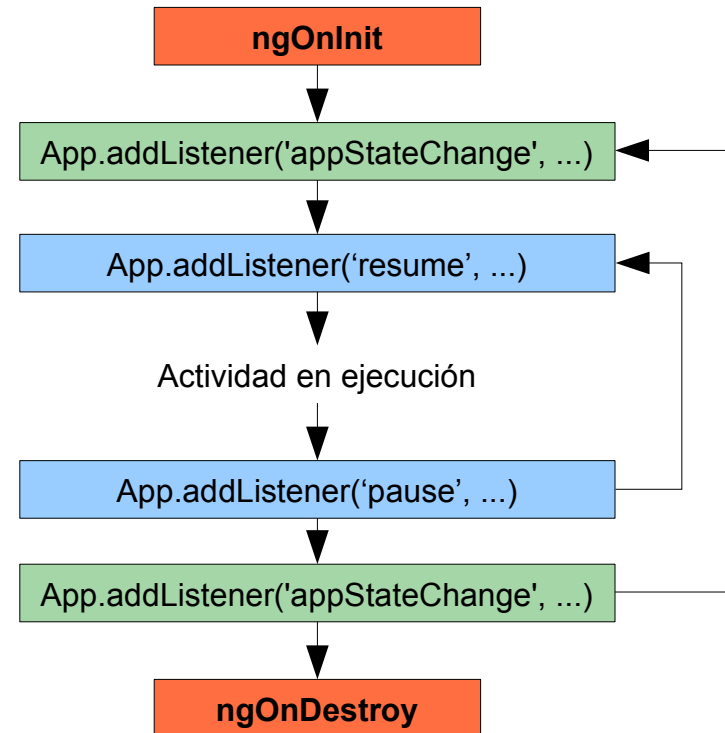
@Component({...})
export class HomePage {
  constructor(private platform: Platform)
  {
    App.addListener('resume', () => {
      console.log('onResume');
    });
    App.addListener('pause', () => {
      console.log('onPause');
    });
    App.addListener('appStateChange', ({ isActive }) => {
      console.log( isActive ? "onStart" : "onStop" );
    });
  }
}
```

Más info en: <https://capacitorjs.com/docs/apis/app>

Aplicación nativa



Ionic+Capacitor



PUBLICAR UNA APP

- Para modificar los iconos y *splash screens* de la aplicación tenemos que instalar la utilidad “capacitor-assets”:

```
$ npm install @capacitor/assets --save-dev
```

- Posteriormente crearemos la carpeta “resources” con los siguientes elementos:

```
resources/  
├── icon-only.png  
├── icon-foreground.png  
├── icon-background.png  
├── splash.png  
└── splash-dark.png
```

- Y por último, para generar los iconos y *splash screens* para las plataformas ejecutamos:

```
$ npx capacitor-assets generate
```

Más info en: <https://github.com/ionic-team/capacitor-assets>

1. Copiar la última versión del código:

```
npx cap copy && npx cap sync
```

2. Abrir el proyecto con Android Studio.

- Ir al menú “*Build*” y seleccionar la opción “*Generate Signed Bundle / APK*”
- Seguir las instrucciones para compilar y firmar la aplicación.

3. Subir la aplicación a Google Play.

➤ Actualizar una aplicación:

- Repetir los pasos anteriores, pero antes de compilar y firmar la aplicación de nuevo con Android Studio es necesario modificar el fichero “android/app/build.gradle” para **incrementar el valor** del “versionCode”.

¿PREGUNTAS?