

Programación de dispositivos móviles

TEMA III



PRIMEROS PASOS EN EL DESARROLLO DE APPS EN IONIC

3.1 Introducción

Ionic es un SDK completo de código abierto para desarrollo de aplicaciones móviles híbridas creado por Max Lynch, Ben Sperry y Adam Bradley en Drifty Co., en 2013.

La primera versión se lanzó en 2013, construida sobre AngularJS y Apache Cordova. En cambio, la última versión lanzada se reconstruye como un conjunto de Componentes Web, que permiten al usuario escoger cualquier framework de interfaz gráfica, como Angular, React o Vue.js. También permite el uso de componentes Ionic, sin ningún framework de UI en absoluto.

La plataforma Ionic proporciona herramientas y servicios para el desarrollo de aplicaciones móviles, aplicaciones de escritorio y PWA(Aplicaciones Web Progresivas), utilizando tecnologías tales como CSS, HTML5, JS. Se pueden distribuir a través de App Stores nativas (Apple Store y Google Play), para ser instaladas en dispositivos.

En el año 2017 Google creó el concepto de **PWAs** (Progressive Web Applications), que permitía desarrollar un tipo de aplicación web con más características de app móvil: icono de app para instalar en el escritorio, notificaciones push y capacidad de funcionamiento offline. Microsoft ha acogido esta tecnología, y la recomienda para las apps de su tienda de apps.

Historia de Ionic

La base de Ionic está desarrollada sobre Angular y Cordova, vio la luz en 2013 con la única intención de que los desarrolladores pudieran crear aplicaciones móviles híbridas con la particularidad y beneficios de los dos framework sobre los que fue construida.

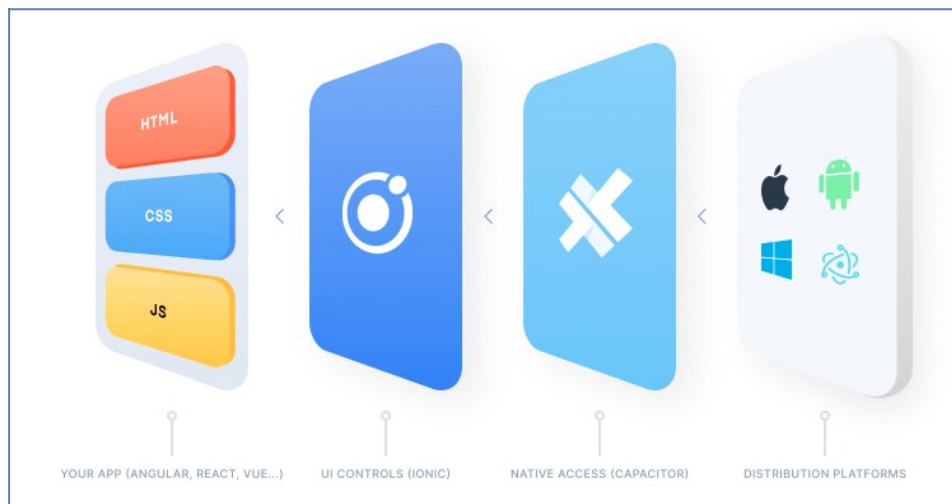
Una de las principales ventajas de trabajar con Ionic es que aprovecha todos los plugins (Hardware, software, imágenes, texto, códigos QR, etc) del marco de desarrollo móvil Cordova.

En 2016 se actualiza a la versión Ionic 2, en donde llega la modularización, una de las actualizaciones más completas de este framework, que permite separarlo por partes: core, angular, native, etc.

La actualización de Ionic 3 es prácticamente imperceptible, aunque si trae novedades en su rendimiento.

El nuevo cambio importante en la herramienta de desarrollo de aplicaciones móviles es su versión Ionic 4. Remplazar AngularJS por Angular moderno. El conjunto de componentes de esta herramienta utiliza elementos personalizados y las API DOM de Shadow disponibles en todos los navegadores modernos para dispositivos móviles y de escritorio.

Actualmente estamos en la versión 5.



El desarrollo de aplicaciones móviles con Ionic nos garantiza que la implementación del proyecto sea mucho más estable, sencilla y con una interfaz de usuario óptima.

Veamos la documentación <https://ionicframework.com/docs>

Mi primer ejemplo

Para crearlo

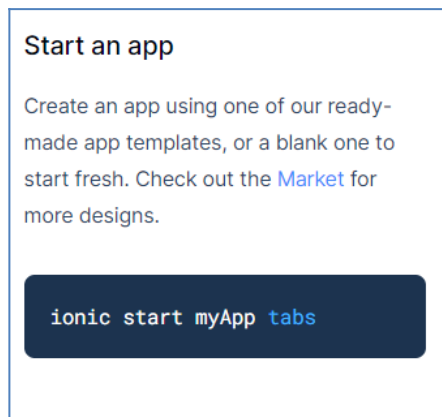
Utilizaremos la herramienta ionic cli para generarlo.

ionic start nombre template

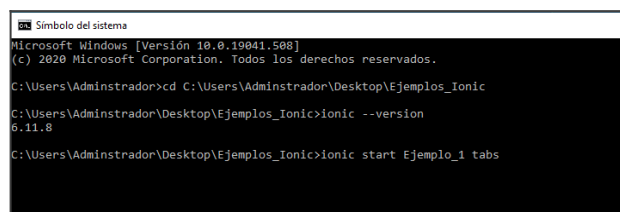
Donde nombre será el nombre de nuestro proyecto y template la plantilla base de nuestro proyecto. Se pueden crear varios tipos de proyectos:

- blank: plantilla vacía
- tabs: plantilla con tabs
- sidemenu: plantilla con menú lateral

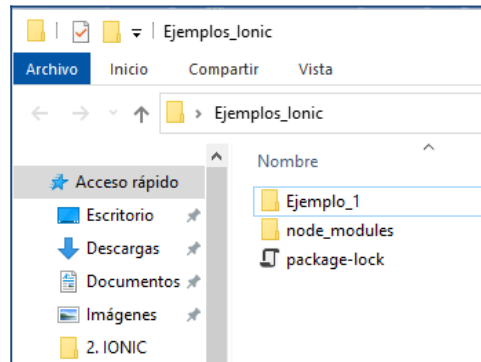
En general los crearemos en blanco pero esta primera vez vamos a crear un tab que incorpora muchos componentes



Ejecutar ese comando desde la carpeta en la que guardaremos nuestra aplicación



Tendremos que elegir framework angular, contestar afirmativamente en todos los pasos. Al final nos preguntará si queremos crear cuenta, de momento no lo haremos. El proceso lleva unos minutos. El resultado final



Ionic cli es la que genera el esqueleto del proyecto, con archivos y carpetas así como la configuración básica de herramientas que necesitaremos.

Para ejecutarlo

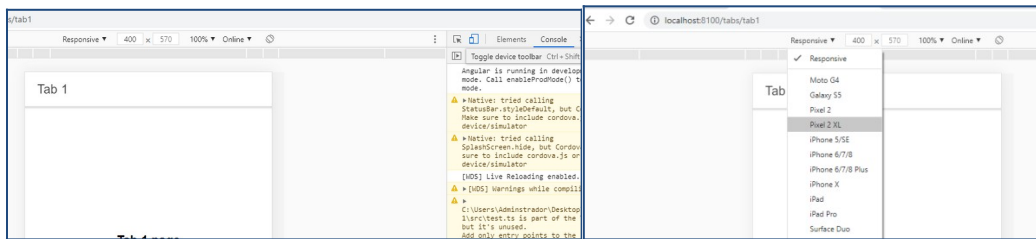
Para correr la aplicación: ionic serve en la carpeta del proyecto

Tardará un rato y finalmente

```
[ng] : Compiled successfully.  
[INFO] Development server running!  
      Local: http://localhost:8100  
      Use Ctrl+C to quit this process  
[INFO] Browser window opened to http://localhost:8100!
```

Este comando levanta un servidor local en un puerto y abre por defecto una aplicación web en el navegador.

Para ver la aplicación en formato dispositivo móvil, inspeccionar la página y elegir opción Consola



Veremos que la consola nos muestra Warnings, esto es porque hay cosas configuradas en la aplicación que sólo funcionan con el dispositivo físico. Se puede comentar el código para que no aparezcan pero realmente la app será desplegada en dispositivo móvil por lo que no merece la pena .

```
initializeApp() {  
  this.platform.ready().then(() => {  
    //this.statusBar.styleDefault();  
    //this.splashScreen.hide();  
  });  
}
```

appcomponents.ts

Elegir la opción de dispositivo sólo afecta a las dimensiones de la pantalla, el navegador no tiene simulador. La opción Responsive nos permite decidir esas dimensiones (tablet).

Tenemos otra posibilidad a la hora de visualizar nuestra app

ionic serve --lab

Es un entorno de pruebas en el que directamente aparecen simulados dos dispositivos, android e IOS. Si no tenemos instalado ese paquete nos preguntará si lo instala.

Para instalarlo nosotros directamente `npm i -D -E @ionic/lab`

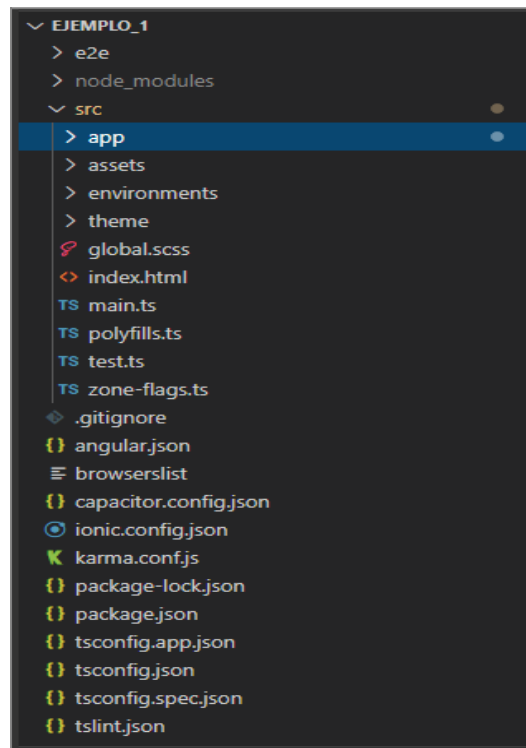
Personalmente prefiero la opción Consola del navegador web porque puedo ver errores.

Ejemplo_1: Sobre el proyecto creado, haz un Hola Mundo.

3.1.2 Estructura de un proyecto

La estructura de carpetas y archivos. El ciclo de la aplicación es muy parecido a angular. La página index.html carga el <app-root>, selector del app-component.

La mayor parte de los archivos de un proyecto son mantenidos de forma automática por el framework y su utilidad es la de crear y automatizar.



Veamos que es cada uno de ellos

- **e2e**: parte de pruebas unitarias y de integración
- **node**: todos los módulos de node, incluidos angular e ionic. Esta carpeta no se pasa a producción, porque incluye muchas cosas que sólo usaremos en desarrollo. Al pasar a producción, se empaquetan los módulos de ionic necesarios para correr la aplicación y sólo esos. Estos archivos son generados y mantenidos por el framework.
- **src**: nuestra aplicación, lo veremos en más detalle
- **gitignore**: archivo que le dice a git que archivos no serán parte de mi repositorio porque se generan mediante comandos

- **angular.json:** configuraciones de angular, rara vez lo modificaremos
- **package.json:** es el archivo de configuración de un proyecto de Node. Se definen y manejan características tan importantes como:
 - Nombre del proyecto.
 - Versión.
 - Dependencias.
 - Repositorios.
 - Autores.
 - Licencias.
- **browserlist:** informativo
- **.conf, .json:** archivos de configuración que rara vez modificaremos. A veces es necesario hacer algún cambio pero se hace a través de un comando .Por ejemplo el tsconfig.json es donde se especifican los archivos y opciones para que el compilador pueda trabajar.
- **tslint.json:** archivo de configuración que nos ayuda a programar de forma limpia. Por ejemplo configuramos que nos muestre error cuándo escribimos mal algo....Generalmente no se toca a no ser que algo en el editor no queramos que pase.

Ionic se actualiza bastante, esta lista a veces cambia pero no es un problema para los desarrolladores. Se puede consultar la documentación, es bastante buena

<https://ionicframework.com/docs>

Nuestra aplicación: src

- **assets:** recursos estáticos, imágenes...cuando pongamos en producción la aplicación serán transferidos.
- **environment:** por ejemplo podemos cambiar de desarrollo a producción
- **themes:** contiene los css o scss globales .
 - variables.css variables para poder utilizar en css sin tener que repetir o escribir sus valores

- **global.css:** añadiremos nuestras importaciones de css, incluso si es poco código podemos escribirlo directamente ahí
- **index.html:** punto de entrada de una aplicación web, renderiza la app.
- **pollyfills:** para habilitar configuración para dispositivos antiguos
- **test:** para las pruebas
- **app:** estarán nuestros archivos de la aplicación. En el ejemplo que creamos aparecerá lo necesario para cargar las tabs. Cuándo hagamos aplicaciones sin basarnos en una plantilla de ionic usaremos comandos tanto de angular como del propio ionic para no tener que escribir todas esas configuraciones.
- **main.ts:** el principal punto de entrada para tu aplicación. Compila la aplicación con el compilador y arranca el módulo raíz de la aplicación (AppComponent) para ejecutarse en el navegador o donde toque.

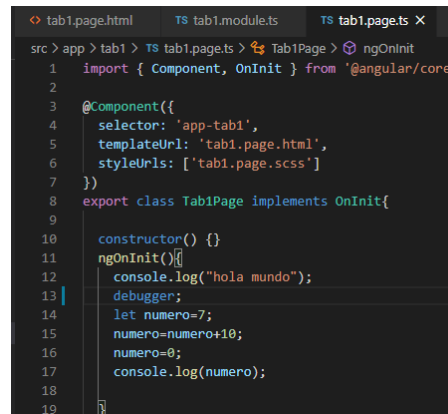
En el app-routing vemos que si el path está vacío se redirige al path home (como hacíamos en angular). Vemos que se hace una carga perezosa de dicho módulo

```
const routes: Routes = [  
  {  
    path: 'home',  
    loadChildren: () => import('./home/home.module').then( m => m.HomePageModule)  
  },  
  {  
    path: '',  
    redirectTo: 'home',  
    pathMatch: 'full'  
  },  
];
```

El template del componente home es la primera página que se mostrará

Ejemplo_2: "Depurando mi app"

Crear un proyecto de tabs, añadir código que escriba en la consola del navegador para poder depurarlo utilizando la instrucción debugger desde Visual Studio Code o bien la herramienta del propio navegador.



```
src > app > tab1 > TS tab1.page.ts > Tab1Page > ngOnInit
1  import { Component, OnInit } from '@angular/core'
2
3  @Component({
4    selector: 'app-tab1',
5    templateUrl: 'tab1.page.html',
6    styleUrls: ['tab1.page.scss']
7  })
8  export class Tab1Page implements OnInit{
9
10   constructor() {}
11   ngOnInit(){
12     console.log("hola mundo");
13     debugger;
14     let numero=7;
15     numero=numero+10;
16     numero=0;
17     console.log(numero);
18   }
19 }
```

3.1.3 Generar elementos de ionic : Generator

ionic generator es la herramienta que se utiliza para crear los distintos elementos de una app ionic:

La lista de elementos que podemos generar automáticamente con ionic generator son:

- **page:** Páginas.
- **component:** Los componentes son un conjunto de html, con su css y su comportamiento que podemos reutilizar en cualquier lugar sin tener que reescribir de nuevo todo.
- **service:** Los services son proveedores que se encargan del manejo de datos, bien extraídos de la base de datos, desde una api, etc. En versiones anteriores a la 4 de ionic se llamaban providers.
- **module:** Nos permite crear un módulo.
- **class:** Nos permite crear una clase.
- **directive:** Una directiva sirve para modificar atributos de un elemento.
- **guard:** Genera un “guardián” que nos permite crear una protección para permitir que los componentes solo se activen si se cumple alguna condición, como por ejemplo si el usuario está logueado.

➤ **pipe:** Los pipes nos permiten crear filtros para aplicar a la información que mostramos en la plantilla, por ejemplo podemos aplicar un filtro que convierta un texto en mayúsculas.

➤ **interface:** Nos permite crear una interfaz.

➤ **Enum:** genera una enumeración.

Páginas

Eliminamos la carpeta home porque crearemos nuestra página (una página es realmente un componente con carga perezosa y rutas)

```
ionic g page pages/inicio
```

- g: generar
- page: generamos una página
- pages/inicio: creará una carpeta pages y añadirá la página home

Si observamos el app-routing vemos que nos ha añadido la correspondiente ruta. Hará esto con todas las páginas que vayamos creando. No debemos olvidar redirigir a nuestro inicio cuándo el path esté vacío.

Navegar entre pantallas

Vamos a crear dos páginas nuevas: alert, action

```
ionic g page pages/alerta --spec=false
```

--spec=false es para indicar que no queremos generar el archivo de pruebas (.spec.ts).

Haremos un ejemplo en el que a través de dos botones podremos cargar esas dos páginas, también implementaremos regresar a la página inicial desde cada una de ellas:

Para cargar otra página, en el template de la página inicio, coloco un **ion-button** con su correspondiente **routerLink** para indicar la ruta que se cargará.

```
<ion-content>
  <ion-button routerLink="/alerta">
    Alerta!
  </ion-button>
</ion-content>
```

La ruta debe coincidir con el path del app-routing.

En el `<ion-toolbar>` queremos colocar una flecha para regresar a la página anterior. Esta zona en ionic está dividida en tres partes:

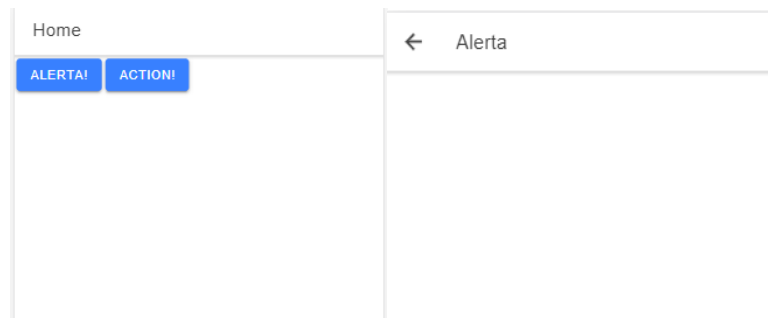
- start: el principio, donde está escrito el título
- la zona de en medio: no tiene nombre
- end: la parte de la derecha de la barra.

Colocaremos un `<ion-buttons>` y dentro un `<ion-back-button>`

```
<ion-header>
  <ion-toolbar>
    <ion-buttons slot="start">
      <ion-back-button></ion-back-button>
    </ion-buttons>

    <ion-title>Action sheet</ion-title>
  </ion-toolbar>
</ion-header>
```

El resultado de la aplicación será



Podríamos tener problemas cuándo no haya página a la que regresar, para eso añadimos en el ion-back-button el atributo `defaultHref="/"`, de esta manera le decimos que si no hay página anterior nos cargue igualmente ese botón(flecha) porque regresamos al inicio

Además de ese atributo, podemos cambiar el color, añadir texto y decidir si queremos que se vea en modo ios...

Podemos consultar todos estos atributos en la ayuda de ionic

<https://ionicframework.com/docs/components>

```
<ion-buttons slot="start">
  <ion-back-button
    defaultHref="/"
    text="Back"
    color="secondary"
    mode="ios">
  </ion-back-button>
</ion-buttons>
```

Podemos cambiar su posición a la derecha, en ese caso ponemos el botón después del título y con `slot="end"`.

Supongamos que queremos esta cabecera en ambas páginas, en lugar de repetir el código en ellas podemos crear un componente e inyectarlo.

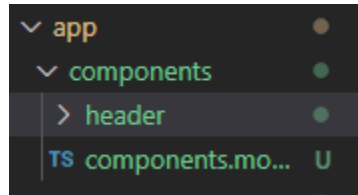
Módulos y componentes

Un componente se diferencia de una página en que no se le añade routing y tampoco carga perezosa. Es como si fuera el componente "plano" de angular.

- Para generar un módulo: `ionic g module path/nombre`
- Para generar un componente: `ionic g component path/nombre`

Si las carpetas existen no se crearán.

Siguiendo con nuestro ejemplo, generamos un módulo llamado components en una carpeta con ese nombre. En dicha carpeta generamos también un componente llamado header



Actualizo el módulo para declarar el componente y exportarlo puesto que lo usaremos fuera de este módulo en el que lo declaramos.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HeaderComponent } from './header/header.component';

@NgModule({
  declarations: [HeaderComponent],
  imports: [
    CommonModule
  ],
  exports: [HeaderComponent]
})
export class ComponentsModule { }
```

No debemos olvidar importar este módulo en todos aquellos módulos en cuyos componentes queramos usar el componente header.

El template del componente header será todo el código <ionic-header> que antes teníamos en ambas páginas, alert y action.

Ese sería el código de ambos templates (alert y action)

```
<app-header></app-header>

<ion-content>

</ion-content>
```

El template del header

```
<ion-header>
  <ion-toolbar>
    <ion-buttons slot="start">
      <ion-back-button
        defaultHref="/"
        text="Back"
        color="secondary"
        mode="ios">
    </ion-back-button>
    </ion-buttons>

    <ion-title>Action sheet</ion-title>
  </ion-toolbar>
</ion-header>
```

Para que nos funcione el `ion-back-button` tenemos que importar el `IonicModule` en el módulo que gestiona el componente cabecera.

Obviamente, tenemos que solucionar el tema del título porque no es el mismo para todas las páginas. Recibiremos mediante `@Input` el texto del dicho título.

```
<app-header titulo="Action Sheet"></app-header>

<ion-content>

</ion-content>

...
export class HeaderComponent implements OnInit {
  @Input() titulo: string;
  constructor() { }

  <ion-title>{{titulo}}</ion-title>
</ion-toolbar>
```

El atributo `titulo` en la primera imagen va sin `[]` porque lo voy a pasar como un `string` y no como una propiedad (tal y como vimos en angular)

Interfaces

Es muy común al utilizar servicios en angular y también en ionic, declarar una interfaz para definir el tipo de datos que nos proporciona el servicio. De esta manera evitamos declarar las propiedades de la clase de tipo any.

Si algo caracteriza a TypeScript, el lenguaje con el que se desarrolla en Angular, es el uso de tipos. Podemos usar tipos primitivos en variables, lo que nos ayudará a recibir ayudas en el momento en el que desarrollamos el código, pero también podemos definir tipos más complejos por medio de clases e interfaces.

Para definir una interfaz

```
interfaz nombre{  
  propiedad: tipo;  
  .....  
  .....  
  propiedad: tipo;  
};
```

Podemos declarar variables de ese tipo.