IONIC

Instalación

- Instalar node.js
- Instalar, a través de npm, ionic y cordova: *sudo npm install -g ionic cordova*
- Crear un proyecto nuevo con *ionic start new_nombre —v2*
- Y para lanzarlo tendremos que entrar en el directorio y ejecutar *ionic serve*

Estructura

- *src/app/*, configuración del componente raíz de angular
- *src/pages/*, páginas/vistas/actividades de nuestra aplicación
- *.json, ficheros de configuración de ionic, tslint…
- *config.xml*, la configuración de apache cordova en nuestro proyecto

GUIA IONIC Oficial

```
ionic start photo-gallery tabs --type=angular --capacitor
```

Ingresar al PATH

Next we'll need to install the necessary Capacitor plugins to make the app's native functionality work:

```
npm install @capacitor/camera @capacitor/storage @capacitor/filesystem
```

PWA Elements

Some Capacitor plugins, including the Camera API, provide the web-based functionality and UI via the Ionic PWA Elements library.

It's a separate dependency, so install it next:

```
$ npm install @ionic/pwa-elements
```

Run the App

Run this command next:

```
$ ionic serve
```

Componentes Básicos:

Ionic es una solución single source (sólo hay una base de código), no nativa (se lanza código JS en el navegador del dipositivo) basada en angular (2 o 4).

Las principales características de ionic (frente a hacer una web responsive con angular) son:

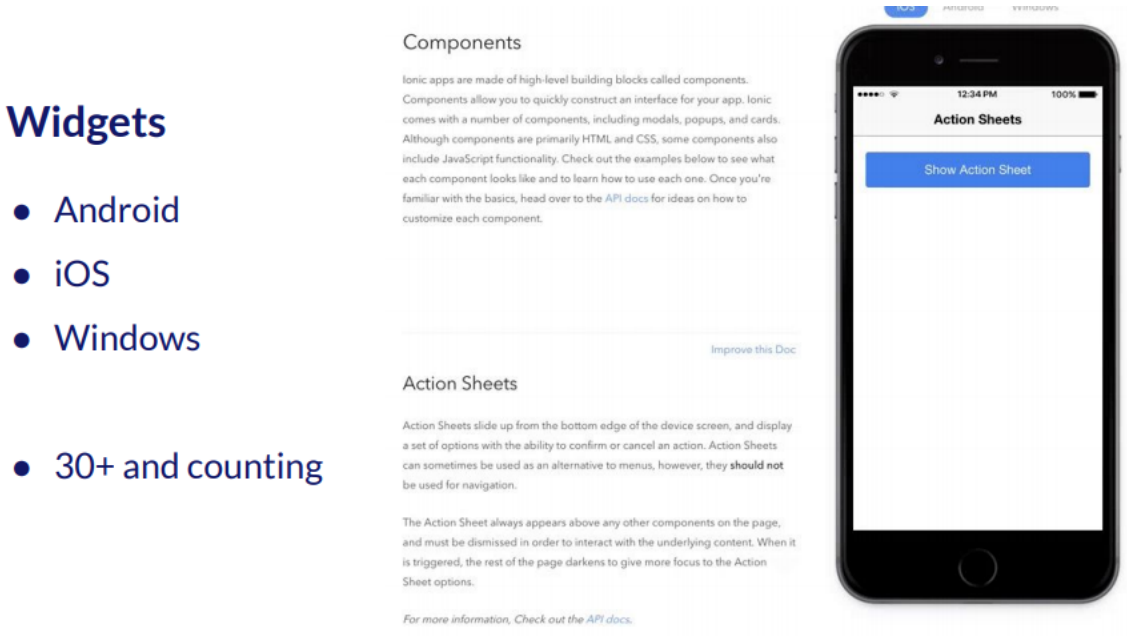
- Librería componentes con tema para iOS & Android
- Integración con Apache Cordova para instalación, puente con las funcionalidades nativas…

Este es el ejemplo de un componente de ionic:

```
import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';

@Component({
 selector: 'page-contact',
 templateUrl: 'contact.html'
})
export class ContactPage {

 constructor(public navCtrl: NavController) {
 }
}
```



Ionic nos va a aportar un conjunto de widgets.

Ejemplos

Inputs con label:

```
<ion-item>
   <ion-label floating>Search</ion-label>
   <ion-input type="text" #searchInput></ion-input>
 </ion-item>
```

Botones:

```
<button ion-button color="light" type="submit">Add Talk</button>
```

Selector de fecha:

```
<ion-item>
<ion-label stacked color="primary">Start Time</ion-label>
<ion-datetime displayFormat="hh:mm" pickerFormat="hh mm"></ion-datetime>
</ion-item>
```

Card:

```
<ion-card>
    <ion-card-header>{{talk.name}}</ion-card-header>
    <ion-card-content></ion-card-content>
</ion-card>
```

Lista

```
<ion-list inset>
  <button ion-item *ngFor="let item of items" (click)="itemSelected(item)">
    {{item}}
```

```
  </button>
</ion-list>
```

Slides:

```
<ion-slides pager>
<ion-slide style="background-color: green">
  <h2>Slide 1</h2>
</ion-slide>
```

Las páginas de ionic son componentes de angular. Podemos generar un componente con el CLI de ionic con *ionic generate*.

Los componentes encapsularan llamadas a servicios y el renderizado del resultado en la pantalla.

Podemos generar un servicio (*provider*) de la misma forma que el angular o creando un componente de ionic, con *ionic generate*.

Los servicios encapsularán lógica de negocio, que implique, probablemente, llamadas de red.

Crear una pantalla de creación en vez de listado implica retocar nuestro servicio para permitir enviar objetos JSON, añadir un formulario (y enviar la información usando cualquiera de las formas soportadas) y hacer la llamada al servicio.

Routing

Hasta ahora hemos creado componentes, embebidos dentro de nuestra página. Pero para soportar una navegación similar a la nativa de los dispositivos móviles tenemos que crear páginas cómo tal.

Para ello, usamos ionic CLI, lanzando *ionic generate* y marcando página.

Una página no es más que:

- Un componente anotado con @IonicPage
- Definido en declarations (y en exports/imports) en el módulo correspondiente
- Navegable en un modelo de stack de navegación

Podemos navegar haciendo uso de la inyección de *NavController* y llamando a:

```
this.navCtrl.push(NEW_PAGE)
```

Si quisiéramos enviar parámetros de una página a otra, podríamos hacerlo con:

```
this.navCtrl.push(NEW_PAGE, {parameters})
```

Para leer los parámetros en la página destino deberíamos inyectar *NavParams* y leerlo con:

```
this.navParams.get('parametro')
```

Resumen
Ionic Routing
ionic generate page TalkDetail
● Componente
● Definida en declarations & imports en el modulo
● Navegable con
this.navCtrl.push(NEW_PAGE, {parameters})
● navCtrl / navParams son inyectables

Plugin Interacción Nativa

Un wrapper por encima de Apache Cordova que nos permite acceder a características nativas con una sintaxis consistente, un API JS y un tratamiento de errores similar.

Ionic Native nos abstraerá de las características particulares de cada plugin de apache cordvoa.

Ionic Storage

- API uniforme para guardado persistente
- Basado en promesas
- Independiente del destino (IndexedDB, SQLite…)

Podemos usarlo instalando el plugin de apache cordova ejecutando: *cordova plugin add cordova-sqlite-storage —save*

Una vez instalado tenemos que proveerlo para poder inyectarlo en nuestros componentes. La forma breve de inyectar es:

```
providers: [...,Storage]
```

Para usarlo tenemos que inyectarlo y usuarlo a través de un API clave/valor:

```
storage.set('name', 'Max');
storage.get('name').then((v) => {})
```
Podemos también forzar una implementación para una plataforma concreta al proveer la dependencia:

```
providers: [{provide: Storage, useFactory: () =>
    new Storage(['sqlite', 'websql', 'indexeddb'], { name: '__mydb' }
}]
```

InAppBrowser

Disponible en ionic native, tenemos que instalar el plugin de apache cordova con:

```
cordova plugin add cordova-plugin-inappbrowser@1.7.1 --save
```
Y después instalamos el wrapper de ionic native:

```
npm install --save @ionic-native/in-app-browser
```
Para usarlo podemos inyectar *InAppBrowser*, proveerlo y usarlo con la siguiente sintaxis:

```
let browser = this.browser.create('...');
browser.show();
```
Para probarlo en un dispositivo local podemos lanzar nuestra aplicación ejecutando:

```
ionic cordova run android
```
NativeStorage

Plugin de apache cordova recubriendo las SharedPreferences de Android, un almacén en xml.

Instalamos los dos wrappers con:

```
ionic cordova plugin add --save cordova-plugin-nativestorage
npm install --save @ionic-native/native-storage
```
Para usarlo inyectamos *NativeStorage*, proveemos y usamos con:

```
this.nativeStorage.setItem(key, true);
```
Este código deberemos probarlo siempre en nuestra aplicación nativa porque si no fallará directamente. Para probarlo:

```
ionic cordova run android
```
Snackbar

Los *snacbkars* son un patrón de feedback de UI en Android que podemos utilizar a través de un plugin de apache cordova que *No* tiene wrapper de ionic native.

Para usarlo, primero instalamos el plugin:

```
cordova plugin add cordova-plugin-snackbar
```

Y podemos usarlo (ejecutando en móvil) con:

```
cordova.plugins.snackbar('texto', 'INDEFINITE', 'acción', function () {});
```

Para que Typescript no se queje tenemos que indicarle que *cordova.plugins.snackbar* en una variable que rellenaremos en base a un script local, fuera de su control y que no tendrá tipos.

Para decirle que ignore el chequeo de tipos sobre esa variable habrá que escribir algo similar a:

```
declare let cordova: any;
```

Para publicar en los markets tenemos que tener en cuenta varias cosas:

- *config.xml*, el fichero de configuración de apache cordova que especifica los plugins que estamos usando, versiones…
- *resources/*, la carpeta en la que dejaremos los iconos de nuestra aplicación y las imágenes promocionales.

Comandos

- *ionic serve —lab*, un comando para servir la aplicación en html pero con un pequeño simulador de cómo se vería en diferentes plataformas.
- *ionic cordova run (—livereload)* para lanzar en dispositivo físico. Podemos lanzarlo con el atributo *livereload* para que haga recarga dinámica (más rápida que reconstruir el binario) en el dispositivo móvil.
- *ionic cordova build android —release*, para construir el binario que subiremos al market
- keytool, jarsigner, zipalign, los comandos que tendremos que lanzar a mano para firmar el binario para producción.
- chrome://inspect/#devices, la url de chrome para hacer remote debugging

Publicación en los
markets
Antes de la Release
● config.xml
● resources/
● ionic serve --lab
● ionic run android (--livereload)

Release
● ionic build android --release
● Keytool, jarsigner, zipalign
● ionic build android --release --
--keystore="../android.keystore"
--storePassword=android --alias=mykey
--password=myKeyPassword
● chrome://inspect/#devices -> remote debugging
Ionic y
herramientas en la
nube
Ionic y herramientas en la nube
● https://apps.ionic.io/app/03d40782/overview
Augury y language
service
Extensiones para angular
● Usa angular language service!
● Usa Augury
Webpack
Webpack
Angular CLI no permite un custom webpack.config.js
Futuro sistema de addons
Webpack
Opciones
● Usar una semilla (y perder angular cli)
● Usar el CLI y hacer ng eject (y perder angular cli)
○ Cuidado con los flags
Nativescript
Nativescript
● npm install -g nativescript
● tns create
● tns run android
Nativescript
● Angular
○ Código nativo (single source & native)
○ Tags móviles
● Código nativo
○ Wrap librerías
○ ...
Qué cambia?
Lo mismo:
● Angular (componentes, servicios, pipes…)
Differente:
● UI

● Librerías

View Components

http://docs.nativescript.org/angular/ui/components

● Button
● Label
● TextView
● Image
● ...

http://docs.nativescript.org/angular/ui/layout-containers

● StackLayout
● DockLayout
● ...

Integración nativa

Añadir una librería en el fichero gradle

● Generar un fichero de types (con esto)
● declare let com: any;
● Instanciar la librería (no se puede desde layouts)