Tutorial de Ionic 2

Construye Apps móviles multiplataforma con ionic 2 desde cero

1 - Introducción

Hola a todos, hoy voy a comenzar con la serie de entradas destinadas a aprender a crear aplicaciones móviles multiplataforma utilizando Ionic 2.

Aunque este tutorial está destinado a aprender ionic 2 desde cero si que es recomendable tener un conocimiento básico de javascript, html y css, por lo que si eres desarrollador web podrás reciclar tus conocimientos.

Aunque los puristas tal vez piensen que es mejor aprender una base teórica solida antes de empezar a programar nada, creo que esto puede hacer que muchos desistan por el camino. En cambio es mucho más motivador ver que nada más empezar se puede hacer cosas que funcionan aunque sean sencillas y ir sobre la marcha aprendiendo conceptos a medida que los vamos necesitando.

Es probable que se queden cosas en el tintero pero aprenderás lo suficiente para empezar a desarrollar tus propias apps y podrás investigar por tu cuenta cuando te encuentres con alguna necesidad que no este explicada en este tutorial.

Iremos aprendiendo las cosas sobre la marcha según las vayamos necesitando en los ejemplos de aplicaciones que vamos a realizar.

Voy a intentar separar el tutorial en entradas relativamente cortas para que se haga más ameno el aprendizaje.

Vamos a ver una pequeña introducción sobre que es ionic y que nos aporta en el desarrollo de aplicaciones móviles.

¿Ques es ionic?

Ionic es un framework que nos permite crear de una manera rápida y sencilla aplicaciones móviles multiplataforma (Android, IOS, Windows) utilizando tecnologías web (HTML, JAVASCRIPT, CSS).
Para poder utilizar elementos web en la app utiliza lo que se conoce como una Webview.

A este tipo de aplicaciones se las conoce como aplicaciones híbridas. El resultado final es una app “nativa” que puedes subir a las tiendas de apps.

Ionic 2 esta basado en Apache Cordova y Angular 2, por lo que serán necesarios unos conocimientos básicos de estas tecnologías para sacar mayor provecho al desarrollo.

Además ionic 2 utiliza TypeScript como lenguaje de programación, si no dominas las anteriores tecnologías mencionadas no te preocupes, tratare de ir explicando las cosas básicas necesarias según las vayamos necesitando en los ejemplos que realizaremos.

No profundizare en cada tecnología, es decir no voy a hacer un tutorial completa de Angular 2, seguro que puedes encontrar con facilidad muchos en la web, solo aprenderemos una pequeña base que nos permita saber lo suficiente para realizar las cosas más comunes de una app, sin embargo podrás investigar por tu cuenta si en algún momento necesitas saber algo más sobre Angular.

Al terminar sabrás lo suficiente para defenderte y crear tus propias aplicaciones.

Ventajas de utilizar ionic para desarrollar apps

La principal ventaja de utilizar Ionic es que es multiplataforma, es decir que con un mismo código podemos generar apps para Android, IOS y Windows, por lo que tiempo y coste de desarrollo y mantenimiento de una app se reduce sensiblemente.

Otra ventaja es que si dispones de conocimientos previos en desarrollo web frontend ya tienes medio camino andado ya que la curva de aprendizaje será mucho menor.

Además Ionic dispone de muchos componentes ya creados para que sin apenas esfuerzos puedas desarrollar una app de apariencia profesional sin necesidad de ser un gran diseñador.

Desventajas de utilizar ionic para desarrollar apps

La principal diferencia con las apps puramente nativas es que estas utilizan los elementos de la interfaz nativa de la plataforma en lugar de correr en una webview, lo que supone una mayor fluidez en el funcionamiento de la app a la hora de cambiar de pantalla, hacer scroll, etc, sin embargo con los dispositivos cada vez más potentes que existen en el mercado y la mejora en el

rendimiento de las webview que incorporan las versiones modernas de los sistemas operativos móviles, esta diferencia en el rendimiento es cada vez menos notoria y en la mayoría de los casos la experiencia de usuario de una aplicación híbrida desarrollada con ionic bien diseñada será muy similar a la de una aplicación nativa.

Diferencia entre Ionic 2 e Ionic 1

En ionic 1 la apariencia de la App era igual para cada plataforma salvo que modificases elementos en función de la plataforma lo que hace más engorroso el diseño si queremos diferenciar el diseño según la plataforma.
Con ionic 2 sin tener que modificar nada tendremos un diseño con el estilo propio de cada plataforma (con material design en caso de Android) dando una apariencia de app nativa.

La estructura del proyecto y la organización del código esta mejor estructurada y es más modular, lo que nos permite un desarrollo mas organizado y fácil de mantener.

Ionic 2 dispone del comando Ionic generator que nos permite desde consola crear diferentes elementos como páginas, tabs, providers etc, ahorrandonos tiempo de desarrollo. Veremos Ionic generator y más adelante.

Ionic 2 se basa en Angular 2 por lo que incorpora las mejoras en cuanto a rendimiento que ha incorporado AngularJs en su nueva versión.

Como ya he mencionado Ionic 2 utiliza typescript, lo que nos permite utilizar toda la potencia de la programación orientada a objetos, tipado estático, además nos permite utilizar todos los elementos de EcmaScript 6 y muchos del futuro EcmacScript 7. Veremos esto con más profundidad en próximos capítulos.

Un saludo y hasta el próximo post.

P.D: si no quieres perderte los próximos capítulos del tutorial ¡suscribete a mi blog! 😉

2 - Instalar ionic y las herramientas necesarias para el desarrollo

En el post anterior hicimos una introducción sobre ionic 2 y sus características.

Hoy vamos a ver como instalar ionic 2 y todas la herramientas necesarias para empezar a desarrollar tus aplicaciones con Ionic 2.

Para instalar ionic debemos instalar primero nodejs para ello descargamos el paquete mas reciente de http://nodejs.org/ y lo instalamos.

Una vez instalado nodejs abrimos un terminal (consola del sistema) e instalamos ionic y cordova con el siguiente comando:

```
npm install -g cordova ionic
```

Si lo estás ejecutando en linux o en mac debes utilizar sudo por delante, es decir:

```
sudo npm install -g cordova ionic
```

Al utilizar sudo ejecutamos el comando con privilegios de administrador (root) por lo que nos pedirá la contraseña de usuario.

Una vez instalado ya podríamos crear aplicaciones con ionic y ejecutarlas en el navegador, sin embargo para poder ejecutarlas en un dispositivo o emulador Android debemos instalar las herramientas de desarrollo de Android, así mismo para poder ejecutar la app para IOS necesitaremos instalar Xcode.

Cabe mencionar que aunque Android Studio lo podemos instalar en cualquier plataforma, es decir podemos desarrollar para Android desde un pc con Windows, Linux o MAC, Para poder compilar las Apps para IOS necesitamos un MAC con Xcode instalado.

Para instalar Xcode en Mac solo tenemos que buscarlo en la App Store e instalarlo, es gratuito.

Existen soluciones en la nube como Phonegap Build para poder compilar una App para IOS si no disponemos de un MAC.

Ahora vamos a ver como instalar android studio en Ubuntu (Linux), Mac OS y Windows:

Como instalar Android Studio en Ubuntu

Primero debemos instalar Java SE Development Kit 8:

Desde consola ejecutamos los siguientes comandos:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

Para verificar la versión de java instalada escribimos:

```
java -version
```

Ahora para configurar las variables de entorno escribimos lo siguiente:

```
sudo apt-get install oracle-java8-set-default
```

Para versiones de 64 bits tenemos que instalar los siguientes paquetes:

```
sudo dpkg --add-architecture amd64
sudo apt-get update
sudo apt-get install libncurses5:amd64 libstdc++6:amd64 zlib1g:amd64
```

Por ultimo Instalamos Android studio via ppa con los siguientes comandos:

```
sudo apt-add-repository ppa:paolorotolo/android-studio
```

```
sudo apt-get update
sudo apt-get install android-studio lib32stdc++6 mesa-utils
cd /opt/android-studio/bin
./studio.sh
```

Como Instalar Android Studio en Mac

Instalamos java SE

Descargamos la última versión de aqui:

http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

La instalamos y después descargamos e instalamos android studio

https://developer.android.com/studio/index.html

Después de descargar el paquete seguimos los siguientes pasos:

1. Ejecuta el archivo DMG de Android Studio.
2. Arrastra Android Studio y suéltalo en la carpeta Applications. Luego inicia Android Studio.
3. Elige si deseas importar configuraciones previas de Android Studio y luego haz clic en **OK**.
4. El asistente de configuración de Android Studio te guiará en el resto de la configuración. Esto incluye la descarga de componentes del Android SDK que se necesiten para el desarrollo.

Como instalar android studio en windows

Instalamos java SE

Descargamos la última versión de aqui:

http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

Buscamos variables de entorno en el panel de control, le damos añadir nueva y le damos el nombre de JAVA_HOME.

Despues le damos a examinar directorio y buscamos donde esta la carpeta del jdk, en mi caso en C:\Program Files\Java\jdk1.8.0_101

Le damos a aceptar y listo.

Ahora descargamos android studio de aquí:

https://developer.android.com/studio/install.html

Una vez descargado lo ejecutamos el instalador y ya tendríamos Android studio funcionando.

Instalar un editor de código compatible con TypeScript

Para finalizar de instalar las herramientas necesarias, como último paso necesitaremos un editor de código que nos coloree typescript para facilitarnos el trabajo. En realidad podríamos editar el código con cualquier editor de texto plano, pero personalmente recomiendo utilizar Visual Studio Code de Microsoft que es multiplataforma y podéis descargarlo desde el siguiente enlace: https://code.visualstudio.com/

Este editor nos marcara los errores de sintaxis mientras introducimos el código lo que no ayuda bastante. Si prefieres otros editores eres libres de utilizar el que más te guste.

Puedes dejar en los comentarios que editor con soporte para typescript utilizas y así ampliamos el abanico de posibilidades.

En el próximo post veremos como crear por fin nuestro primer hola mundo con Ionic 2.

Un saludo.

P.D: si no quieres perderte los próximos capítulos del tutorial ¡suscribete a mi blog! 😉

3 - Hola Mundo con Ionic 2

Hola a todos:

En el capitulo anterior vimos como instalar ionic y las herramientas de desarrollo que necesitamos para crear aplicaciones multiplataforma con ionic 2.

Ahora vamos a crear nuestra primera aplicación con Ionic 2, el famoso “hola mundo” que siempre es el punto inicial en el aprendizaje de cualquier lenguaje de programación o framework.

Para crear nuestro proyecto “hola mundo” iremos a la consola de comandos ó terminal y escribimos el siguiente comando:

```
ionic start hola-mundo blank --v2
```

Tras un rato descargando los paquetes necesarios para crear nuestra aplicación ya estará listo nuestro proyecto.

El comando **start** de ionic cli (command–line interface) se utiliza para crear (iniciar) un nuevo proyecto, el siguiente parámetro es el nombre del proyecto, en este caso **hola-mundo**, el siguiente parámetro es el tipo de plantilla que vamos a utilizar, en este caso **blank** que indica que utilizaremos una plantilla vacía. Por ultimo usamos –**v2** para indicar que vamos a crear un proyecto de ionic 2, si no lo indicamos nos creará un proyecto de ionic 1.

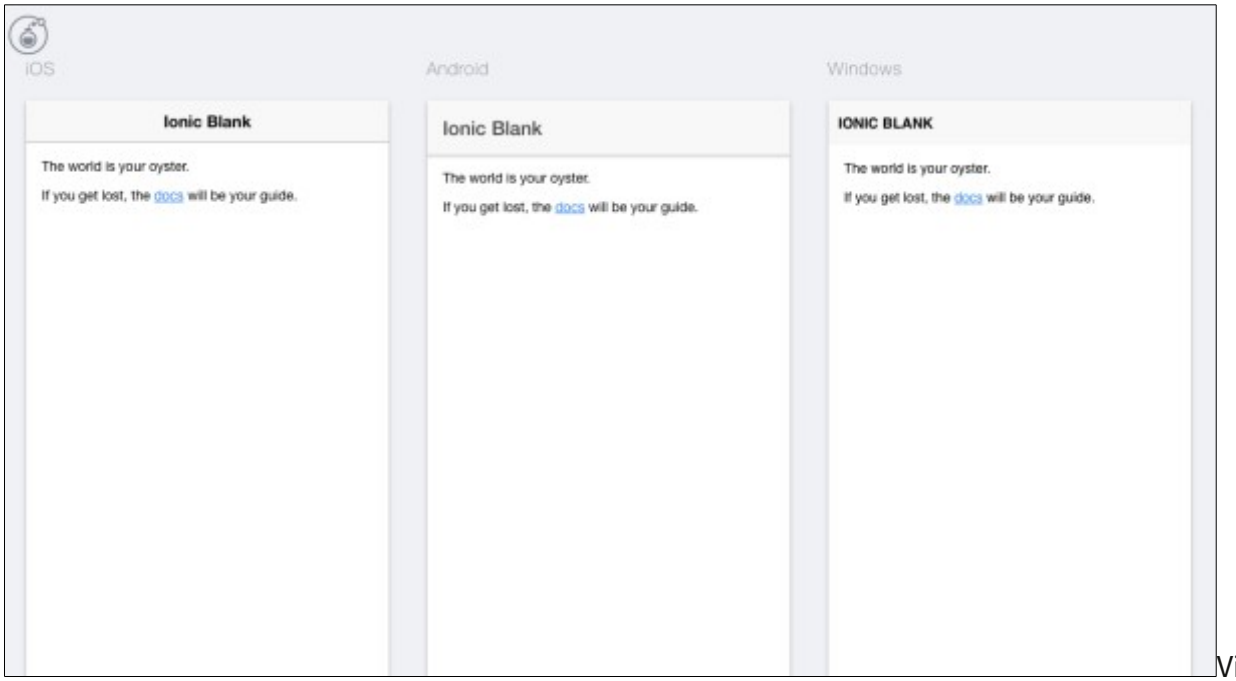
Existen tres tipos de plantillas:

- **blank** : Crea una plantilla vacía.
- **sidemenu**: Crea una plantilla con menú lateral.
- **tabs**: Crea una plantilla con Tabs (Pestañas).

Para ver el resultado en el navegador debemos entrar dentro de la carpeta del proyecto ‘hola-mundo’ que se acabamos de crear con **ionic start** y escribimos el siguiente comando:

```
ionic serve -l
```

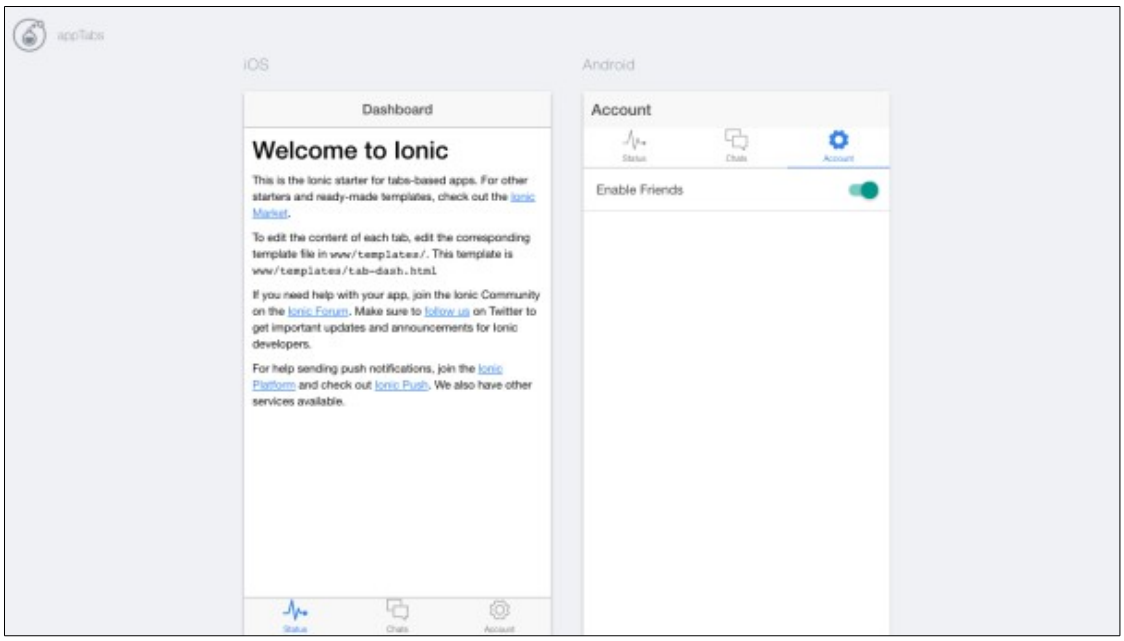
Con el parámetro **-l** nos muestra como queda nuestra app en las tres plataformas soportadas, es decir, IOS, Android y Windows.



Vista de como quedaría nuestra primera App con plantilla Blank en el navegador.
Si no utilizamos -l mostrará solo una plataforma.

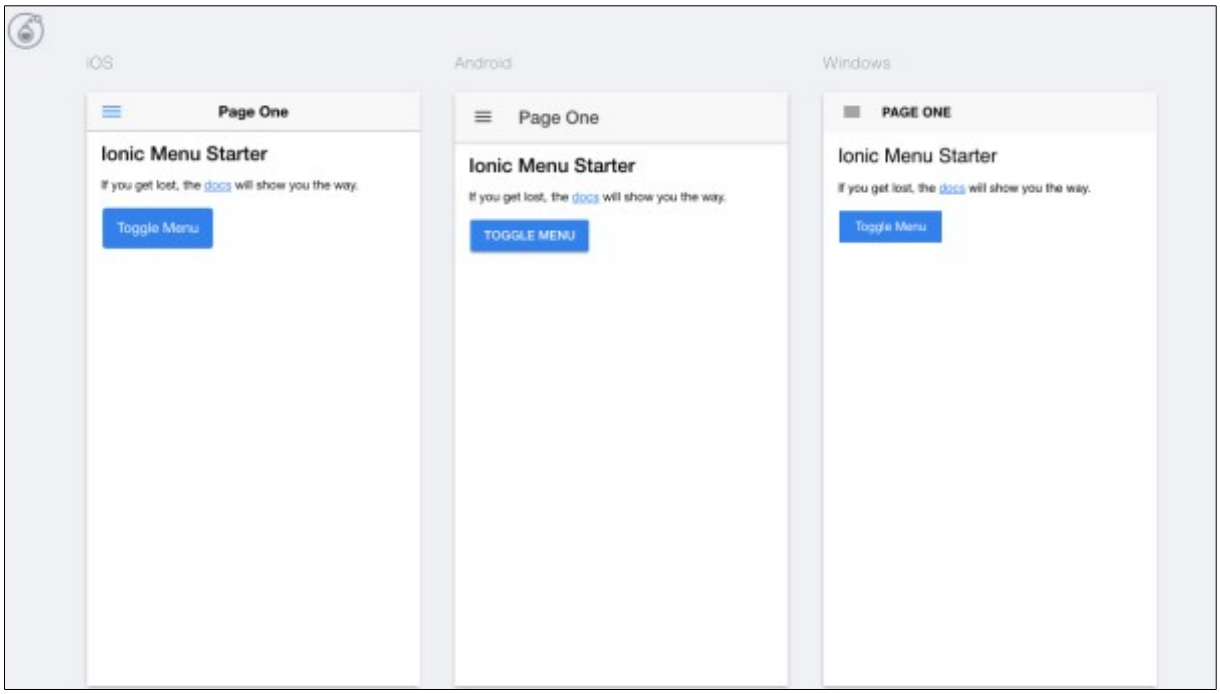
Para ver las posibilidades de ionic podemos crear un proyecto de prueba con plantilla tabs y otro con sidemenu:

```
ionic start appTabs tabs --v2
```



Aplicación con plantilla tabs

```
ionic start appSide sidemenu --v2
```



Aplicación con plantilla sidemenu

Como podemos ver sin nosotros hacer nada tenemos creada la estructura de una app que podremos modificar para añadir lo que necesitemos.

En el próximo capitulo profundizaremos en la estructura de carpetas de una aplicación con ionic 2 y modificaremos nuestro hola mundo.

P.D: si no quieres perderte los próximos capítulos del tutorial ¡suscribete a mi blog! 

4 - Estructura de un proyecto Ionic 2

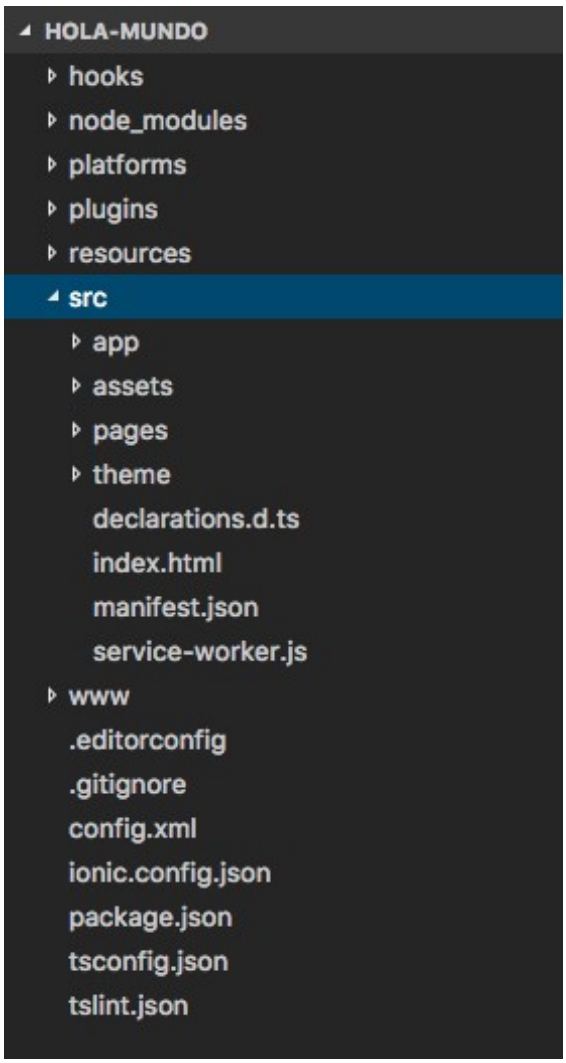
Hola a todos:

En el post anterior vimos como crear un proyecto en ionic 2 creando el famoso “Hola Mundo”.

Hoy vamos a ver la estructura de carpetas y archivos que se general al crear un proyecto ionic.

Al crear un proyecto con ionic 2 se crea una carpeta con el nombre del proyecto y dentro de ella una estructura de archivos y directorios que contiene todos los elementos del proyecto.

Vamos a echar un vistazo a la estructura de archivos y carpetas que se ha generado al crear nuestro proyecto hola-mundo:



Estructura de carpetas de un proyecto Ionic 2

Veamos que contiene cada carpeta:

hooks: Contiene scripts que se ejecutan en el proceso de construcción de la app y que pueden ser creados por el propio sistema de compilación para personalizar comandos de cordova, automatizar procesos, etc. Normalmente no tendremos que modificar nada aquí.

node_modules: La carpeta node_modules se genera automáticamente al instalar las dependencias npm con “npm install”. Este comando explora el archivo package.json para todos los paquetes que necesitan ser instalados. No necesitamos tocar nada en esta carpeta.

platforms: En esta carpeta se generará los proyectos nativos para cada plataforma que hayas añadido previamente. Si hemos añadido la plataforma IOS y Android se creará una carpeta llamada ios y otra llamada android y dentro tendrán los archivos y carpetas con la estructura de un proyecto nativo.
Estas carpetas se generan al añadir la plataforma y se actualizan cada vez que compilas o

ejecutas el proyecto en un emulador o en un dispositivo. Salvo excepciones no tendremos que modificar nada en estas carpetas.

plugins: Contiene los plugins de Cordova que hayamos instalado. Se crean automáticamente en esta carpeta al instalar un plugin así que tampoco tendremos que modificar nada en esta carpeta a mano.

resources: Contiene el icono y la "splash screen" (pantalla de presentación) de la aplicación con la que después podremos crear automáticamente todas las imágenes en todos los tamaños necesarios para cada plataforma, lo que nos ahorrará mucho tiempo al no tener que generar a mano todos los tamaños de imagen necesarios del icono y la pantalla de presentación. Veremos más adelante como se generan automáticamente.

src: Esta es la carpeta más importante y donde realizaremos la mayor parte de nuestro trabajo. Aquí es donde están los archivos con el contenido de nuestra aplicación, donde definimos las pantallas, el estilo y el comportamiento que tendrá nuestra aplicación.

www: Esta carpeta se genera automáticamente y contiene la versión actual del código cada vez que efectuamos un cambio. Si habías trabajado anteriormente con ionic 1 puede que tengas la tentación de editar el contenido de esta carpeta pero **NO debemos cambiar nada aquí** ya que todo lo que cambiemos en esta carpeta se machacará con cada cambio que realicemos en la carpeta **src** que es donde realmente debemos editar nuestra aplicación.

Veamos ahora varios archivos que se generan al crear un proyecto com ionic 2.

.editorconfig y **.gitignore** son dos archivos ocultos, así que dependiendo de tu sistema puede que no los veas, están relacionados con la configuración del editor de código y Git, en principio no tenemos que preocuparnos por ellos.

config.xml: El archivo config.xml contiene parámetros que se utilizan cuando se construye un proyecto nativo a partir de un proyecto ionic. Aquí deberemos indicar los permisos especiales que necesite la aplicación y otras configuraciones que puedan ser necesarias.

Ionic.config.json: Contiene información básica sobre la configuración nuestro proyecto, se utiliza si vas a subir tu aplicación a la plataforma Ionic.io.

package.json: Contiene paquetes y dependencias de nodeJS.

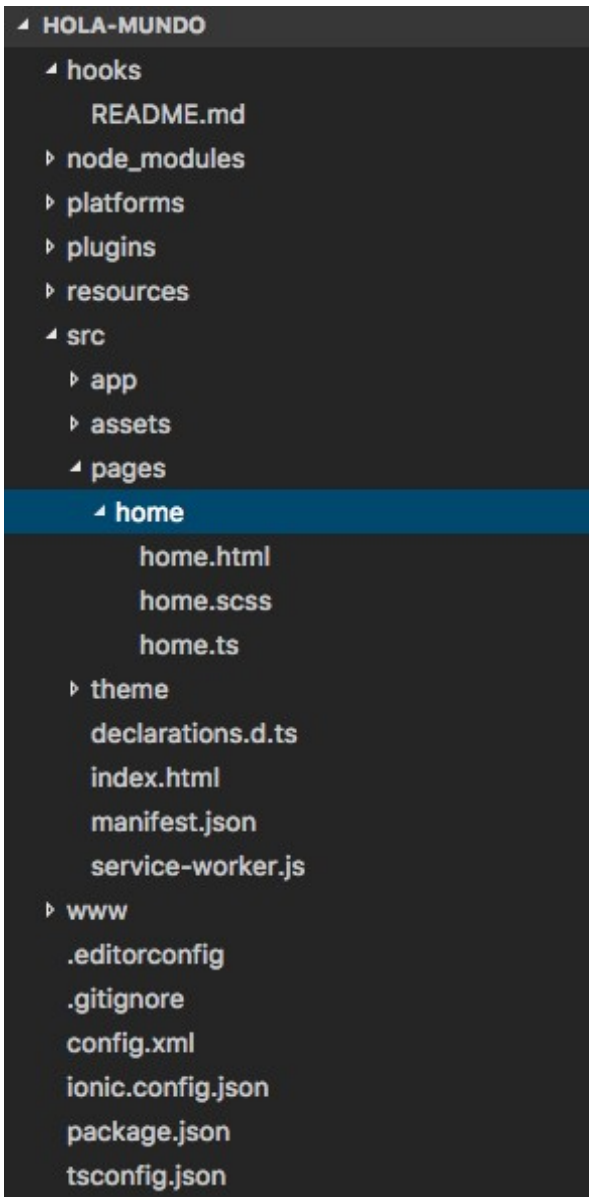
tsconfig.json y **tslint.json:** Son archivos que contienen información necesaria a la hora de compilar TypeScript, no necesitamos editar estos archivos.

Aunque pueda parecer complicado en realidad la mayoría de los elementos los gestiona automáticamente Ionic y nosotros solo tenemos que preocuparnos de la carpeta **src** que es donde se va a situar nuestro código. Ocasionalmente puede que tengamos que editar algún archivo fuera del directorio **src** como el archivo config.xml.

Ahora que ya hemos visto la estructura de un proyecto en ionic 2 vamos a modificar nuestro hola-mundo para que realmente diga “hola mundo”.

Si desplegamos el directorio **src** podemos ver la carpeta **pages**, en está carpeta es donde se van a alojar todas la páginas que contenga nuestra aplicación. Para que nos entendamos una página será como una vista o un pantalla de nuestra aplicación.

Al crear un proyecto con la plantilla blank ionic genera por defecto un página llamada **home,** que como su propio nombre indica es la pagina inicial que se va a mostrar al iniciar nuestra aplicación. Esta página la podemos mantener como página principal y modificarla, o podemos eliminarla y crear otra con el nombre que nosotros queramos. De momento vamos a mantener la que nos ha creado por defecto y vamos a modificar su contenido.



Carpeta pages dentro de src.

Como podemos ver dentro de la carpeta de la página home que nos ha creado hay tres archivos:

El archivo **home.html** que contiene la plantilla html de la página.
El archivo **home.scss** que contiene el archivo sass donde podremos modificar el estilo de los componentes de la página.
El archivo **home.ts** que es el archivos typescript que contiene el controlador de la página, donde definiremos el comportamiento de la misma, como por ejemplo la función con la lógica a ejecutarse cuando se pulse sobre un botón de la página etc. Veremos mas adelante en profundidad cada una de las partes de una página.

Modificando nuestro hola mundo

Si abrimos el archivo home.html veremos que contiene algo como esto:

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Ionic Blank
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  The world is your oyster.
  <p>
    If you get lost, the <a
href="http://ionicframework.com/docs/v2">docs</a> will be your guide.
  </p>
</ion-content>
```

Contenido home.html.

Las paginas se pueden crear utilizando html puro, sin embargo aquí podemos ver algunas etiquetas que no corresponden con las etiquetas html "estándar". Lo que vemos aquí son componentes de ionic.

Ionic nos ofrece una amplia gama de componentes listos para utilizar y que nos facilitarán la labor de crear nuestra interfaz de usuario con un estilo atractivo y profesional.
Iremos viendo diferentes componentes de ionic según los vayamos necesitando a lo largo de este tutorial. En este enlace podéis consultar en la documentación oficial de ionic los componentes disponibles con pequeños ejemplos de como implementarlos: https://ionicframework.com/docs/v2/components/

Todos los componentes de ionic comienzan con el prefijo "**ion-".**

Como ionic está basado en Angular 2 si en algún caso no nos es suficiente con los componentes que nos ofrece ionic podríamos crear nuestros propios componentes personalizados de la misma manera que en angular 2, aunque en la mayoría de los casos no será necesario ya que ionic nos ofrece una amplia gama de componentes para poder desarrollar nuestras

aplicaciones.
Veremos mas sobre componentes en posteriores capítulos.

En la página principal (y única de momento) de nuestro proyecto hola-mundo vemos que tenemos los siguientes componentes:

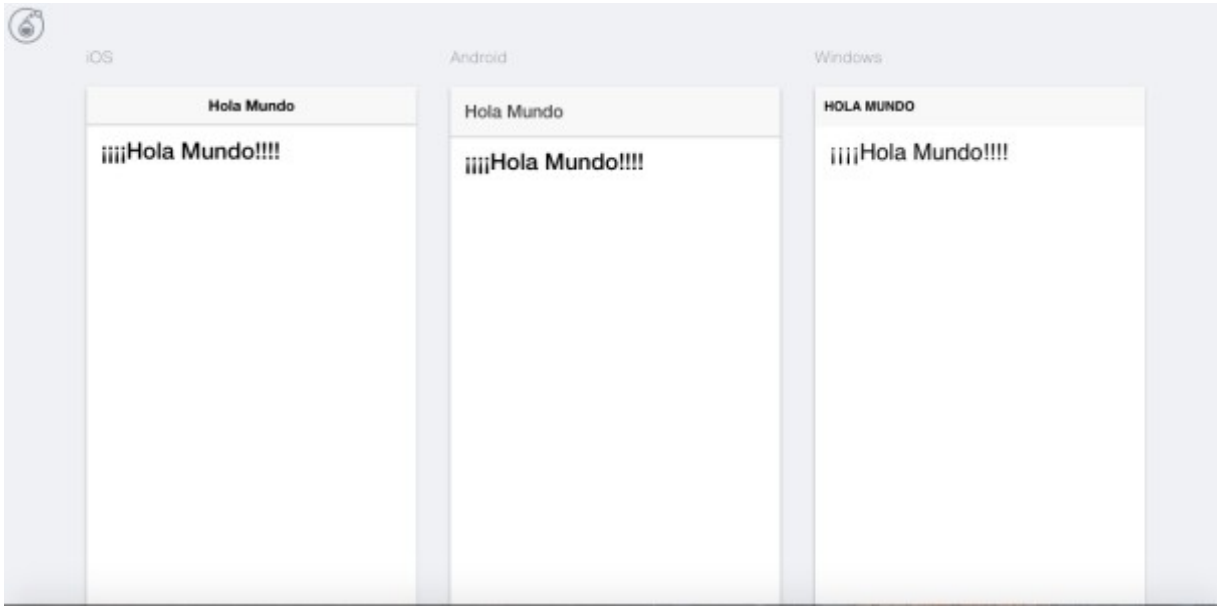
- **ion-header**: Cabecera.
- **ion-navbar**: Barra de navegación.
- **ion-title**: Título.
- **ion-content**: Contenido de la página.

Bien, vamos a cambiar el contenido de ion-title por “Hola Mundo”, también vamos a borrar todo lo que hay dentro de la etiqueta **ion-content** y vamos a poner orgullosos “**<h1>¡¡¡¡Hola mundo!!!!<h1>”**, así el código de home.html debería quedar de la siguiente manera:

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Hola Mundo
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
 <h1>¡¡¡¡Hola Mundo!!!!</h1>

</ion-content>
```

Contenido modificado de home.html.

Si desde el terminal,escribimos **ionic serve -l** para ver desde el navegador como queda en las tres plataformas nuestra aplicación veremos algo como esto:



Nuestro Hola Mundo en acción.

Ahora ya si podemos ver nuestro Autentico Hola Mundo en acción.

En el siguiente capitulo veremos como implementar la lógica de una página en el controlador, para ello construiremos un minijuego de acertar números con ionic.

P.D: si no quieres perderte los próximos capítulos del tutorial ¡suscribete a mi blog! 😉

5 - Mini Juego de acertar números en ionic 2, el controlador de la página y Data Binding y *ngIF

Hola a todos:

Después de ver en el capitulo anterior como modificar la plantilla de una página para simplemente mostrar el texto “Hola Mundo”, hoy vamos a avanzar un poco más en nuestro aprendizaje y vamos ha programar nuestra primera aplicación interactiva.

Se trata de un simple juego de acertar el número secreto. La aplicación creará un número aleatorio que nosotros debemos adivinar.

Por primera vez vamos a programar con ionic 2. Por el camino aprenderemos conceptos como **Data Binding**, que nos permite actualizar valores entre el html y el controlador y recoger las acciones del usuario como pulsar un botón.

Bien, sin mas preámbulos vamos a empezar:

Vamos a crear un nuevo proyecto ionic 2 que en este caso le vamos a llamar adivina: abrimos el terminal o consola de comandos y escribimos:

```
ionic start adivina blank --v2
```

Nos situamos dentro de la carpeta del proyecto de acabamos de crear:

```
cd adivina
```

Y una vez dentro ejecutamos ionic serve para ver en el navegador los cambios que vayamos haciendo en nuestra aplicación.

```
ionic serve
```

Recuerda que si utilizas la opción **-l** con ionic serve puede visualizar como queda la app en todas las plataformas.

Ahora abrimos la carpeta del proyecto que acabamos de crear en el editor, entramos en **src/pages/home** y abrimos el archivo **home.html.**

Editamos su contenido para que quede de la siguiente manera:

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Adivina el número secreto
    </ion-title>
  </ion-navbar>
</ion-header>
<ion-content padding>
    <ion-input type="number" min="1" max="100" [(ngModel)]="num"
placeholder="Introduce un número del 1 al 100"></ion-input>
    <p>El número secreto es {{ mayorMenor }}</p>
   <button ion-button block (click)="compruebaNumero()">Adivina</button>
</ion-content>
```

Si nunca antes has programado en angular ni en ionic es posible que el código te resulte extraño, no te preocupes, ahora vamos a ver lo que es cada cosa, si por el contrario ya has programado en angular o ionic 1 es probable este código te resulte familiar.

Tenemos una plantilla básica de una página con **ion-header**, **ion-navbar** y **ion-title** donde hemos puesto el título de nuestra aplicación “Adivina el número secreto”. Hasta aquí ninguna novedad con lo que ya habíamos visto en el capitulo anterior.

Dentro de **ion-content** que como sabemos es donde va el contenido de la página es donde esta lo interesante:

```
 <ion-input type="number" min="1" max="100" [(ngModel)]="num"
placeholder="Introduce un número del 1 al 100"></ion-input>
```

Primero tenemos un **ion-input** que es muy parecido a un input de html, le decimos que sea de tipo number, le ponemos un rango mínimo a 1 y máximo a 100.

El siguiente parámetro **[(ngModel)]** es el encargado de hacer el Data Binding entre el valor del input y una variable llamada **num** que estará definida como luego veremos en el controlador de la página. Este es un Data Binding bidireccional ya que si se introduce un valor en el input automáticamente este será reflejado en la variable del controlador, del mismo modo si algún proceso cambia el valor de esta variable en el controlador este automáticamente aparecerá reflejado como valor del input. Por último el input tiene un placeholder indicando que se introduzca un valor entre 1 y 100.

La siguiente linea es un párrafo que contiene lo siguiente:

```
<p>El número secreto es {{ mayorMenor }}</p>
```

Si ya conoces ionic o angular sabes de que se trata. En este caso las con las dobles llaves “{{ }}” hacemos un Data Binding unidireccional, mayorMenor es una variable que estará definida en el controlador y con las dobles llaves muestra su valor. En este caso la variable contendrá el texto “mayor” en caso de que el número secreto sea mayor al introducido, “menor” en caso de ser menor e “igual” en caso de ser igual.

Por último tenemos un botón para comprobar si el número introducido coincide con el número secreto.

```
<button ion-button block (click)="compruebaNumero()">Adivina</button>
```

En la documentación oficial de ionic podéis ver más sobre como crear distintos tipos de botones con ionic 2: https://ionicframework.com/docs/v2/components/#buttons

De momento esto es todo lo que necesitamos en la plantilla de nuestro juego.

Ahora necesitamos programar el comportamiento en el controlador de la página.

Si abrimos el archivo **home.ts** que se genera automáticamente al crear un proyecto en blanco veremos el siguiente código:

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

import { NavController } from 'ionic-angular';

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

  constructor(public navCtrl: NavController) {

  }

}
```

Este código esta es typeScript. Como ya hemos comentado TypeScript permite utilizar elementos de las últimas versiones del estándar ECMAscript aunque todavía no estén implementadas en el navegador que estemos utilizando. También nos permite utilizar variables tipadas.

Ionic 2 se basa Angular 2 que a su ver utiliza TypeScript, así que es muy beneficioso conocer angular 2 y typescript.
Aquí vamos a explicar a groso modo las cuestiones imprescindibles para poder desenvolvernos con ionic 2 pero para saber más sobre angular y sacarle el máximo partido podéis consultar en http://learnangular2.com/ así como otros muchos recursos que se pueden encontrar en la red para aprender Angular 2.

Analicemos el código de **home.ts:**

Lo primero que vemos son dos lineas **import:**

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

import { NavController } from 'ionic-angular';
```

Import se utiliza para importar módulos que contienen librerías y clases para poder utilizarlas en nuestro proyecto. Podemos importar módulos propios de Ionic que ya se incorporan al crear un proyecto, librerías de AngularJS, librerías de terceros que podemos instalar o nuestras propias librerías.

En este caso se importa el elemento **Component** de Angular y el elemento **NavController**de ionic-angular.

NavController lo utilizaremos más adelante para poder navegar entre distintas páginas, de momento en este ejemplo no lo necesitamos.

Las páginas son componentes de Angular, por eso necesitamos importar **Component.**

Después vemos el siguiente código:

```
@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
```

@Component es un decorador de Angular. Angular 2 usa los decoradores para registrar un componente.

Los decoradores empiezan con **@.**

Los decoradores se colocan antes de la clase y definen cómo se va a comportar esta.

Existen los siguientes decoradores:

- @Component
- @Injectable
- @Pipe

- @Directive

  De momento nos interesa **@Component**.

  En el código vemos que **@Component** contiene un objeto con dos atributos, **selector:'page-home',** que es el selector css que se va a aplicar a la página, y **templateUrl:'home.html'** que es la plantilla html que va a renderizar la página.

  Por último se exporta una clase llamada HomePage:

```
export class HomePage {

  constructor(public navCtrl: NavController) {

  }

}
```

Tenemos que exportar la clase para luego poderla importar si queremos llamar a la página desde cualquier otro sitio de la aplicación, por ejemplo cuando navegamos entre páginas. Vemos que la clase tiene un constructor, en TypeScript la creación de clases es muy similar a como sería en otros lenguajes de programación orientado a objetos como Java.

Todos los datos que se muestren en la plantilla y todas las funciones a las que se llame por ejemplo al hacer click en un botón deben formar parte de la clase **HomePage** que es el controlador de la página.

Como ya hemos visto en la plantilla **home.html** utilizamos **(ngModel)]="num"** para recoger en la variable **num** el valor que se introduzca en el campo input, también mostrábamos otra variable llamada **mayorMenor** que indicará si el número introducido es mayor o menor que el número que hay que adivinar:

```
 <ion-input type="number" min="1" max="100"[(ngModel)]="num"
placeholder="Introduce un número del 1 al 100"></ion-input>
    <p>El número secreto es {{ mayorMenor }}</p>
```

Por lo tanto estas variables deben de estar definidas en el controlador dentro de la clase **HomePage** del archivo **home.ts**:

```
export class HomePage {

  num:number;
  mayorMenor: string = '...';

  constructor(public navCtrl: NavController) {

  }

}
```

Como podemos observar para definir una variable ponemos el nombre de la variable y seguido de dos puntos "**:**" el tipo de valor que va a contener. Si lo deseamos podemos inicializar la variable con un valor en el momento de declararla. En este caso **num** es de tipo number y **mayorMenor** de tipo string ya que va a contener un texto con la palabra "Mayor", "Menor" o "Igual" según sea el caso. Inicializamos la variable **mayorMenor** con tres puntos suspensivos "…".

Los tipos primitivos de variable que podemos definir son:

- number (Numérico).
- string (cadenas de texto).
- boolean (Booleano: true o false).
- any (cualquier tipo).
- Array.

También vamos a necesitar otra variable que contenga el número secreto que debemos adivinar, le vamos a llamar **numSecret** y como valor le vamos a asignar la respuesta a la llamada a una función llamada **numAleatorio** que crearemos a continuación.

Definimos la variable **numSecret** de tipo number:

```
numSecret: number = this.numAleatorio(0,100);
```

Hacemos referencia a la función **numAleatorio** con this porque va a ser un método de la propia clase.

Ahora vamos a crear la función **numAleatorio** que nos devolverá un número aleatorio.

```
numAleatorio(a,b)
{
      return Math.round(Math.random()*(b-a)+parseInt(a));

}
```

Esta función recibe como parámetros dos valores que serán el rango mínimo y el máximo para el número aleatorio, es este caso le estamos pasando 0 y 100 por lo que obtendremos un número aleatorio entre 0 y 100.

Por último vamos a crear la función **compruebaNumero** que se llama al pulsar el botón **Adivina** que en la parte html lo hemos definido así:

```
<button ion-button block (click)="compruebaNumero()">Adivina</button>
```

Y ahora en **home.ts** definimos la función dentro de la clase HomePage:

```
compruebaNumero(){
    if(this.num)
    {
      if(this.numSecret < this.num)
      {
        this.mayorMenor = 'menor';
      }
      else if(this.numSecret > this.num)
```

```
      {
        this.mayorMenor = 'mayor';
      }
      else{
        this.mayorMenor = 'igual';
      }
    }
  }
```

Esta función compara el número secreto que esta contenido en la variable **numSecret**con el número introducido por el usuario que se aloja en la variable **num** y le asigna a la variable mayorMenor el texto “menor”, “mayor” o “igual” en función de si es menor, mayor o igual que esta.
Observa que debemos utilizar **this** para las variables ya que son atributos de la propia clase.

El código completo de **home.ts** quedará de esta manera:

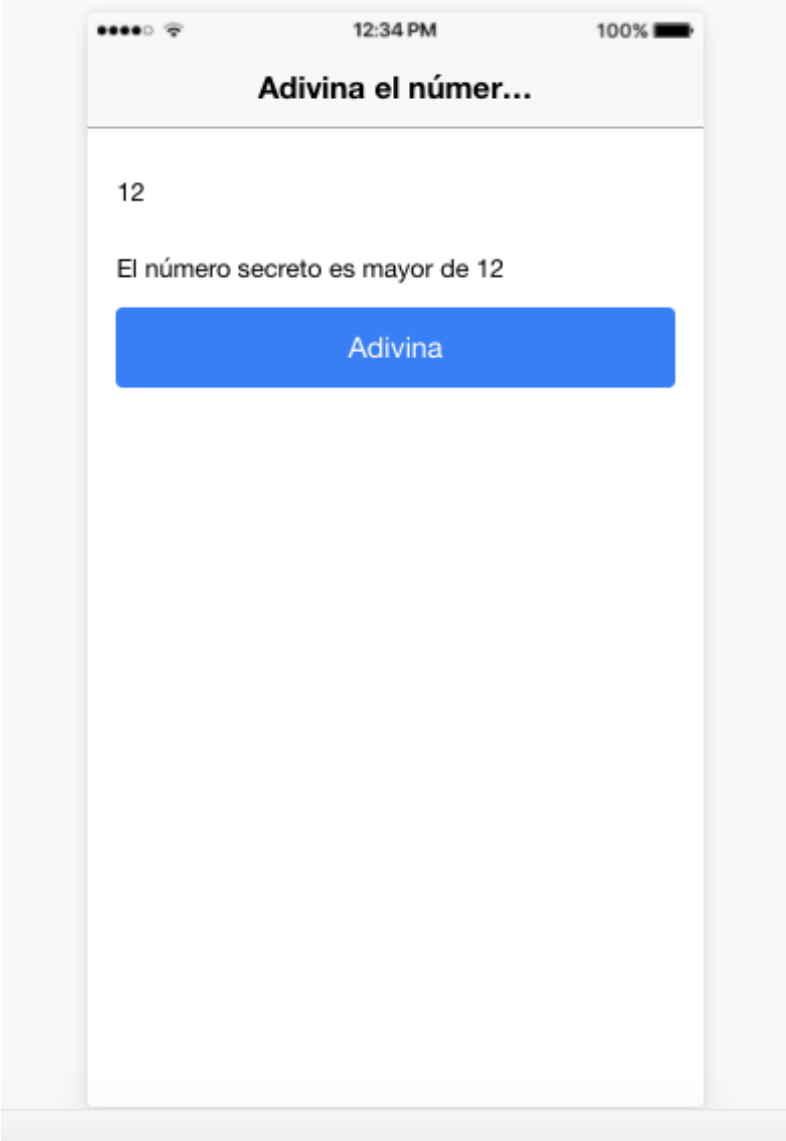
```
import { Component } from '@angular/core';

import { NavController } from 'ionic-angular';

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

  num:number;
  mayorMenor: string = '...';
  numSecret: number = this.numAleatorio(0,100);

  constructor(public navCtrl: NavController) {

  }

  compruebaNumero(){
```

```
    if(this.num)
    {
      if(this.numSecret < this.num)
      {
        this.mayorMenor = 'menor';
      }
      else if(this.numSecret > this.num)
      {
        this.mayorMenor = 'mayor';
      }
      else{
        this.mayorMenor = 'igual';
      }
    }
  }

  numAleatorio(a,b)
  {
        return Math.round(Math.random()*(b-a)+parseInt(a));
  }

}
```

Ahora si lo ejecutamos en el navegador con **ionic serve** (si no lo tenias ya en marcha), podemos ver un campo para introducir un número, y debajo un texto que dice “el número secreto es … de”, en estos momentos no hay introducido ningún número por lo que no aparece ningún número en el texto.

Si introducimos un número vemos que automáticamente en el texto aparece ese número el mismo tiempo que lo escribimos. Eso es porque estamos realizando un **Data Binding**con la variable **num**. Al utilizar **[(ngModel)]=”num”** en el campo input le estamos indicando que la variable coja el valor que introducimos en el input y lo refresque automáticamente en todos los sitios donde se utilice, por ejemplo en el texto de debajo.Lo mismo ocurre si cambiamos el valor de la variable desde el código.

Por últimos si pulsamos el botón adivina obtendremos algo como esto:



Vista de nuestro Juego en acción

Al pulsar el botón se ha ejecutado la función **compruebaNumero(),** dentro de la función se ha comprobado que el número secreto es mayor a 12 por lo que a la variable **mayorMenor**se le asigna el texto “mayor”.

Cuando acertemos el número secreto el texto dirá que el número secreto es igual al número introducido. Esto es un poco soso, vamos a hacer algunos cambios para que cuando acertemos el número nos muestre un mensaje felicitándonos por haber acertado.

Este mensaje lo ponemos en la plantilla html pero solo se debe mostrar cuando se cumpla una condición y es que la variable **mayorMenor** contenga el texto ‘**igual’.**

Vamos a añadir lo siguiente en **home.html** justo antes del botón “Adivina”:

```
<ion-card *ngIf="mayorMenor=='igual'">
    <ion-card-header>
       ¡¡¡Enhorabuena!!!
     </ion-card-header>
     <ion-card-content>
         Has acertado, el número secreto es el {{ num }}
     </ion-card-content>
</ion-card>
```

Aprovecho la ocasión para introducir otro componente de ionic: **ion-card**, las cards o “tarjetas” son componentes que muestran la información en un recuadro. Como vemos dentro contiene otros dos componentes **<ion-card-header>** y **<ion-card-content>,** Como habrás adivinado el primero permite mostrar una cabecera dentro de la tarjeta y el segundo el contenido que deseemos.
En la documentación oficial de ionic podéis ver todas las posibilidades que tiene **ion-card**: https://ionicframework.com/docs/v2/components/#cards

Queremos que está card se muestre solo cuando la variable de nuestro controlador **mayorMenor** contenga el texto ‘igual’ después de hacer la comprobación.

Aquí entra en juego la directiva condicional ***ngIf:**

```
<ion-card *ngIf="mayorMenor=='igual'">
```

La directiva ***ngIf** es una directiva estructural, lo que significa que nos permite alterar el DOM (Document Object Model), estas directivas llevan un asterisco “*” delante.
Con ***ngIf** le estamos indicando que el elemento donde se encuentra, ion-card en este caso, solo se muestre si se cumple la condición que tiene definida entre las dobles comillas. En este caso le estamos diciendo que solo se muestre el elemento ion-card cuando mayorMenor==’igual’.

Sabiendo esto vamos a darle otra vuelta de tuerca más y vamos a añadir un botón para volver a jugar una vez que hemos acertado, este botón llamará a una función para volver a generar un

nuevo número aleatorio y reiniciará las variables para que se oculte el ion-card. Este botón debe permanecer oculto hasta que hayamos acertado, y cuando se muestra debe ocultarse el botón **Adivina** para que no podamos pulsarlo después de haber adivinado el número hasta que empecemos una nueva partida.

Editamos **home.html** y añadimos ***ngIf** al botón Adivina para que solo se muestre cuando no hemos acertado el número:

```
<button *ngIf="mayorMenor!='igual'" ion-button block
(click)="compruebaNumero()">Adivina</button>
```

Ahora añadimos el botón "Volver a Jugar" a **home.html**:

```
<button *ngIf="mayorMenor=='igual'" ion-button block
(click)="reinicia()">Volver a Jugar</button>
```

Con ***ngIf** indicamos que solo se muestre cuando hayamos acertado el número, y en el evento **(click)** le indicamos que ejecute la función **reinicia()**.

Por lo tanto ahora vamos a editar nuestro controlador (**home.ts**) y añadimos la función **reinicia()**, que como hemos dicho debe reiniciar las variables para poder comenzar el juego:

```
reinicia(){
    // reiniciamos las variables
    this.num = null;
    this.mayorMenor = '...';
    this.numSecret = this.numAleatorio(0,100);
  }
```

Ahora ya podemos volver a jugar una vez de que hemos adivinado el número.

Este juego se podría mejorar añadiendo un contador de intentos. Podríamos limitar el número de intentos y mostrar un mensaje si se han consumido todos los intentos sin acertar el número secreto. Esto lo dejo como deberes para el que quiera practicar más, si queréis ver como sería la

solución completa con contador de intentos **Luis Jordán** me ha hecho llegar su solución, en el siguiente enlace podéis encontrar el código de la página home que ha mejorado realizado Luis: https://drive.google.com/file/d/0B7M15FxlLmQJZHBSTUlfMmFkSmM/view?usp=sharing

De momento este simple ejemplo nos ha servido para aprender algunos conceptos básicos sobre Ionic 2, en el siguiente capitulo veremos como navegar entre distintas páginas de una aplicación.

P.D: si no quieres perderte los próximos posts ¡suscribete a mi blog! 😉

6 - Crear una aplicación para guardar nuestros sitios geolocalizados

– Parte 1 –

Navegación por Tabs

Hola a todos.

En el capitulo anterior de este tutorial sobre Ionic 2 creamos una app simple, un minijuego de adivinar números que nos sirvió para prender como se programa la lógica de una página en Ionic 2.

Vimos como pasar variables entre el controlador de la página y la vista y como llamar a funciones desde un evento de la vista como pulsar un botón.

La aplicación era extremadamente sencilla y toda la lógica se desarrollaba en la misma página, sin embargo lo normal en cualquier aplicación que sea mínimamente completa es que tenga varias vistas o páginas y haya que navegar entre ellas.

Hoy vamos a aprender a hacer una aplicación con varias páginas y veremos como podemos navegar entre ellas, para ello vamos a realizar una aplicación de ejemplo.

Descripción de la aplicación:

La aplicación consistirá en una herramienta que nos permita guardar el lugar donde nos encontramos actualmente recogiendo las coordenadas gracias al gps del móvil.

Está aplicación puede sernos util por ejemplo para recordar donde hemos aparcado el coche o para guardar un restaurante que nos ha gustado y queremos volver más tarde etc.

La aplicación constará de una pantalla inicial con un mapa donde se mostrará la posición actual y un botón para añadir la posición actual. Al pulsar el botón se abrirá una ventana modal con un pequeño formulario donde añadir una descripción y un fotografía.

Los lugares que guardemos se mostrarán en otra página que contendrá un listado de tus sitios guardados.

Al pinchar sobre uno de nuestros sitios guardados se abrirá otra ventana modal donde mostraremos un mapa con el sitio, la foto, la dirección y la descripción del mismo.

Bien, vamos a comenzar creando nuestra aplicación. Desde la consola de comandos o terminal escribimos:

```
ionic start misSitios blank --v2
```

Una vez creada nuestro proyecto vemos que por defecto nos ha generado la página home dentro de la carpeta pages.

Nosotros a priori vamos a utilizar tres páginas, una con el mapa donde se muestra nuestra posición actual, otra con el listado de nuestros sitios guardados y otra que llamaremos info donde mostraremos información sobre la aplicación y que simplemente utilizaremos de relleno para tener una tercera página y poder mostrar mejor como funciona la navegación entre páginas.

Las paginas normalmente suelen estar asociadas a “tabs” o pestañas y se navega entre ellas cambiando de tab.

Ionic Generator

Nosotros podemos crear las páginas a mano creando una carpeta dentro de la carpeta pages con su vista html y su controlador .ts, y su css, también podemos crear y configurar los tabs a mano, pero el cli (command line interface o interfaz de linea de comandos) de ionic nos facilita muchísimo el trabajo. Ionic dispone de una herramienta llamada **ionic generator**.

Ionic generator nos permite generar plantillas con los componentes que queramos.

Con el siguiente comando obtenemos la lista de elementos disponible que podemos generar con ionic generator:

```
ionic g --list
```

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

- **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.

- **Directive**:
  Una directiva sirve para modificar atributos de un elemento.

- **Page**: Páginas.

- **Pipe**: Los pipes son lo que angular 1 se consideraban filtros.

- **Provider**:Los providers son proveedores que se encargan del manejo de datos, bien extraídos de la base de datos, desde una api, etc.

- **Tabs**: Nos genera la página maestra de Tabs y una página para cada tab.

  Veremos con más detalle cada elemento según lo vayamos necesitando, para este ejemplo de momento nos interesan los **Tabs**.

  Generamos los tabs con el siguiente comando:

```
ionic g tabs misTabs
```

Al ejecutar el comando primero nos pregunta el número de tabs que queremos crear, para este ejemplo vamos a utilizar 3.Utilizamos la flecha hacia abajo del cursor para seleccionar el tres y pulsamos enter.

Acto seguido debemos introducir el nombre del primer tab: le damos el nombre de **Inicio** y pulsamos enter.

Después nos pide que introduzcamos el nombre para el segundo Tab: A este tab le vamos a llamar **Listado**.

Pulsamos enter una vez mas y por ultimo introducimos el nombre del tercer tab. Le llamamos **Info** y pulsamos enter.

```
? How many tabs would you like? 3
? Enter the first tab name: Inicio
? Enter the second tab name: Listado
? Enter the third tab name: Info
```

Hay que tener cuidado al utilizar el comando **ionic g tabs** porque si alguna de las páginas que creamos con ionic g tabs ya existe la “machaca” y la crea de nuevo vacía.

Si echamos un vistazo a las carpetas de nuestro proyecto vemos que en la carpeta **/src/pages** se han creado cuatro páginas nuevas, una por cada uno de los tabs que acabamos de crear y una cuarta llamada **mis-tabs** que sera la página maestra de los tabs.

Si editamos el archivo **mis-tabs.ts** dentro de carpeta mis tabs, vemos que se ha generado el siguiente código:

```
import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';
import { Inicio } from '../inicio/inicio'
import { Listado } from '../listado/listado'
import { Info } from '../info/info'

/*
  Generated class for the MisTabs tabs.

  See https://angular.io/docs/ts/latest/guide/dependency-injection.html
  for more info on providers and Angular 2 DI.
*/
```

```
@Component({
  selector: 'page-mis-tabs',
  templateUrl: 'mis-tabs.html'
})
export class MisTabsPage {

  tab1Root: any = Inicio;
  tab2Root: any = Listado;
  tab3Root: any = Info;

  constructor(public navCtrl: NavController) {

  }

}
```

Esta es la pagina maestra donde vamos a controlar la navegación por tabs.

Si ejecutamos desde consola **ionic serve -lab** (siempre dentro de la carpeta de nuestro proyecto) veremos que salen una serie de errores. Eso es porque hay un pequeño bug y es que el comando **ionic g** cuando genera paginas les agrega la palabra Page por detrás, si vemos el código ts de cualquiera de las paginas que hemos creado, por ejemplo **inicio.ts**vemos que la clase controladora de la página que ha generado se llama **InicioPage,** sin embargo cuando las trata de importar en **mis-tabs.ts** le llama **Inicio** a secas, por lo tanto debemos añadirle **Page** por detrás al nombre de las páginas en los imports.

Después vemos que ha creado tres variables miembro, una para cada página que contiene una instancia de cada página, aquí también debemos añadirle Page por detrás, por lo tanto el código del archivo **mis-tabs.ts** debería quedar de la siguiente manera:

```
import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';
import { InicioPage } from '../inicio/inicio'
import { ListadoPage } from '../listado/listado'
import { InfoPage } from '../info/info'

/*
  Generated class for the MisTabs tabs.
```

```
  See https://angular.io/docs/ts/latest/guide/dependency-injection.html
  for more info on providers and Angular 2 DI.
*/
@Component({
  selector: 'page-mis-tabs',
  templateUrl: 'mis-tabs.html'
})
export class MisTabsPage {

  tab1Root: any = InicioPage;
  tab2Root: any = ListadoPage;
  tab3Root: any = InfoPage;

  constructor(public navCtrl: NavController) {

  }

}
```

He marcado de amarillo las variables que tenéis que cambiar añadiendo Page por detrás.

Bien, ahora ya no muestra esos errores sin embargo vemos que todavía nos muestra la página home que se crea por defecto al crear un proyecto ionic vacío, tenemos que indicarle a la aplicación que la página inicial debe ser la página maestra con los tabs que acabamos de crear.

Cambiar el root de nuestra aplicación (Página principal)

Para indicarle a la aplicación que la página principal sea la página maestra de los tabs que acabamos de crear debemos editar el archivo **src/app/app.component.ts** e importar la pagina **MisTabsPage** con import y luego a la variable **rootPage** asignarle el nombre de la clase de la página que queremos asignar como página principal, en este caso **MisTabsPage**, es decir debemos sustituir:

```
rootPage = HomePage;
```

Por:

```
rootPage = MisTabsPage;
```

El código del archivo **src/app/app.component.ts** completo quedaría así:

```
import { Component } from '@angular/core';
import { Platform } from 'ionic-angular';
import { StatusBar, Splashscreen } from 'ionic-native';

import { MisTabsPage } from '../pages/mis-tabs/mis-tabs';



@Component({
  templateUrl: 'app.html'
})
export class MyApp {
  rootPage = MisTabsPage;

  constructor(platform: Platform) {
    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();
    });
  }
}
```

El import de la página home ya no lo necesitamos así que podemos eliminarlo. de hecho podemos eliminar la carpeta de la pagina home ya que no la vamos a utilizar en esta aplicación.

El archivo **app.component.ts** es el punto de entrada de nuestra aplicación, vemos que se define como un componente que utiliza como plantilla el archivo **app.html.**

```
@Component({
```

```
  templateUrl: 'app.html'
})
```

Si vemos el contenido de **app.html** podemos observar que únicamente contiene la siguiente linea de código:

```
<ion-nav [root]="rootPage"></ion-nav>
```

Le asigna como **root** (es decir como página inicial o raíz) la variable **rootPage**, y como a la variable **rootPage** le hemos asignado la página **MisTabsPage**, la aplicación cargará dentro de **<ion-nav>** el contenido de la página **MisTabsPage**.

De momento no necesitamos modificar nada mas en **app.component.ts**, solo comentar que por defecto se importa el elemento **platform** que tiene el método **ready** que podemos utilizar para ejecutar acciones que queremos que se produzcan en el momento que la aplicación esté completamente cargada:

```
platform.ready().then(() => { ... }
```

De momento lo dejamos como está.

Por último todas las páginas y servicios que vayamos a utilizar los tenemos que declarar en el archivo **app.module.ts.**

Todas las páginas que creemos tienen que ser añadidas tanto al array **declarations** como al array **entryComponents.**

Todos los providers que creemos tienen que ser añadidos al array **providers**. Los providers los veremos más adelante

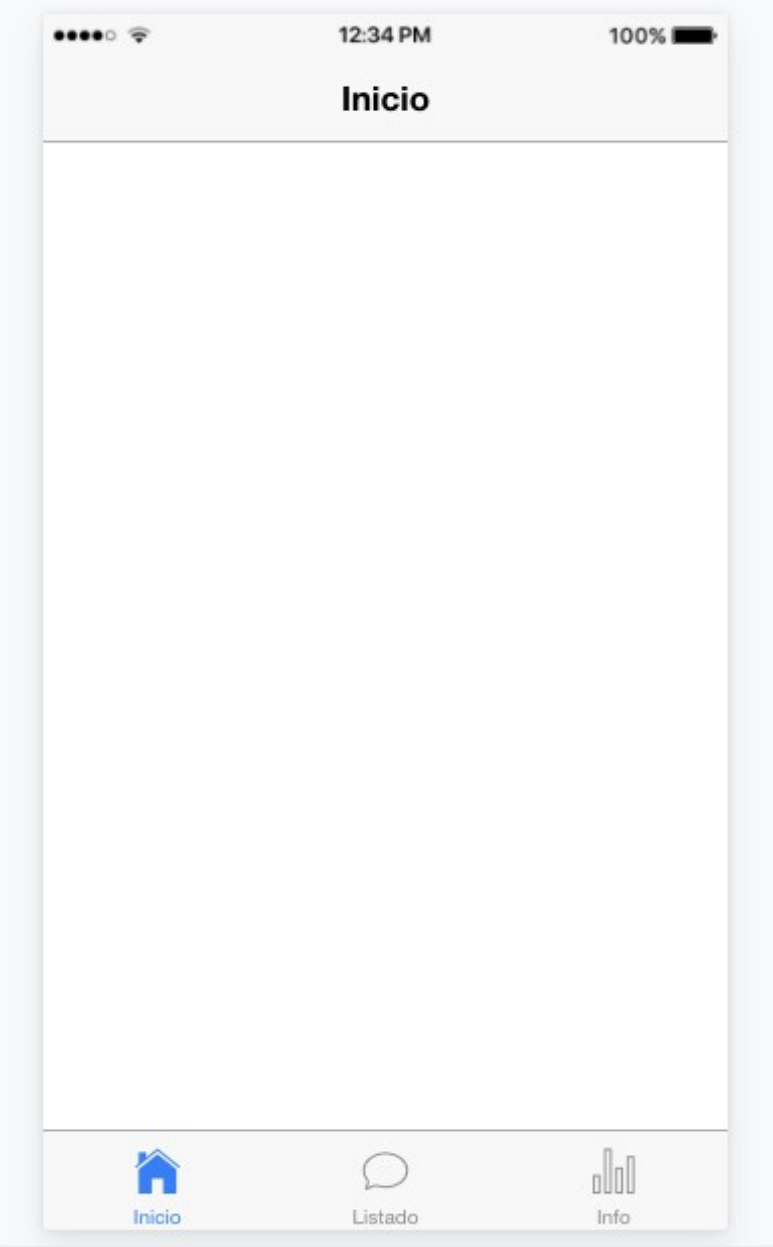
Por último todos los componentes personalizados o pipes deben ser añadidos al array **declarations**.

Siempre que vayamos a declarar o utilizar alguna página, modulo o componente en cualquier lugar debemos importarlo primero, así que en **app.module.ts** debemos importar tanto la página maestra de los tabs como las tres páginas. Un vez importadas tal y como hemos mencionado, debemos declararlas en **declarations** y en **entryComponents.**

Vemos como quedaría el código del archivo **app.module.ts**. Destaco con fondo amarillo los cambios que debéis realizar.

```
import { NgModule, ErrorHandler } from '@angular/core';
import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';
import { MyApp } from './app.component';
import { MisTabsPage } from '../pages/mis-tabs/mis-tabs';
import { InicioPage } from '../pages/inicio/inicio';
import { ListadoPage } from '../pages/listado/listado';
import { InfoPage } from '../pages/info/info';

@NgModule({
  declarations: [
    MyApp,
    MisTabsPage,
    InicioPage,
    ListadoPage,
    InfoPage
  ],
  imports: [
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    MisTabsPage,
    InicioPage,
    ListadoPage,
    InfoPage
  ],
  providers: [{provide: ErrorHandler, useClass: IonicErrorHandler}]
})
export class AppModule {}
```

Ahora ya podemos ejecutar nuestra aplicación y ver las tres pestañas que nos ha creado:



Nuestra app con tabs

Como vemos ha creado por defecto un icono para cada pestaña.

Vamos a ver como podemos cambiar los iconos de los tabs para ello vamos ahora a ver el contenido de la plantilla **mis-tabs.html**:

```
<ion-tabs>
  <ion-tab [root]="tab1Root" tabTitle="Inicio" tabIcon="home"></ion-tab>
  <ion-tab [root]="tab2Root" tabTitle="Listado" tabIcon="text"></ion-tab>
  <ion-tab [root]="tab3Root" tabTitle="Info" tabIcon="stats"></ion-tab>
</ion-tabs>
```

Como podemos observar la página maestra de los tabs (**mis-tabs.html**) solamente contiene el contenedor de tabs que se define con la etiqueta **<ion-tabs>** y luego una etiqueta **<ion-tab>** por cada tab. También podemos observar que cada tab tiene el atributo **[root]** donde se le asigna una variable que si volvemos al controlador en **mis-tabs.ts** vemos que a cada una de esas variables (tab1Root, tab3Root y tab3Root) les hemos asignado una página que hemos importado. Por lo tanto al pulsar sobre un tab se carga el contenido de la página a la que apunta la variable que tiene asignada.

Luego tenemos el atributo **tabTitle** al que se le asigna el texto que se muestra en el tab, si solo queremos que se muestre el icono podemos eliminar o dejar vacío este atributo.

Y por último tenemos el atributo **tabIcon** y es aquí donde se le asigna el icono que se muestra en cada tab, si solo queremos que se muestre texto podemos eliminar o dejar vacío este atributo. Si corres la aplicación en el navegador con **ionic serve –lab** puedes observar que en iOS los iconos son diferentes a los iconos de Android. Los iconos cambian dependiendo de la plataforma móvil, adaptandose al estilo propio de la plataforma. Esta es una de las ventajas de ionic 2, que no te tienes que preocupar por adaptar el estilo que tu app para cada plataforma ya que la mayoría los elementos se adaptan automáticamente al estilo propio de cada sistema operativo.

Probablemente te estarás preguntando, pero…¿de donde salen estos iconos?.La respuesta es simple, ionic viene de serie con una colección de los iconos mas comunes lista para ser usada solamente con asignar como hemos visto el nombre del icono al atributo tabIcon.

El listado de iconos disponibles lo podéis consultar el la documentación oficial de ionic 2 desde el siguiente enlace: https://ionicframework.com/docs/v2/ionicons/.

El tab **Inicio** lo vamos a dejar con el icono **home**, al tab **Listado** le vamos a asignar el icono **list-box** y al tab **Info** le vamos a asignar el icono **information-circle**:

```
<ion-tabs>
  <ion-tab [root]="tab1Root" tabTitle="Inicio" tabIcon="home"></ion-tab>
  <ion-tab [root]="tab2Root" tabTitle="Listado" tabIcon="list-box"></ion-
tab>
  <ion-tab [root]="tab3Root" tabTitle="Info" tabIcon="information-
circle"></ion-tab>
</ion-tabs>
```

Para observar mejor como cambiamos entre las distintas páginas de momento vamos a poner un comentario en cada una.

Editamos **inicio.html** y dentro de **ion-content** añadimos lo siguiente:

```
<!--
  Generated template for the Inicio page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Inicio</ion-title>
  </ion-navbar>

</ion-header>



<ion-content padding>
  <h1>Esta es lá página de inicio, aquí situaremos un mapa con un botón de
añadir lugar.</h1>
</ion-content>
```

Editamos **listado.html** y añadimos los siguiente:

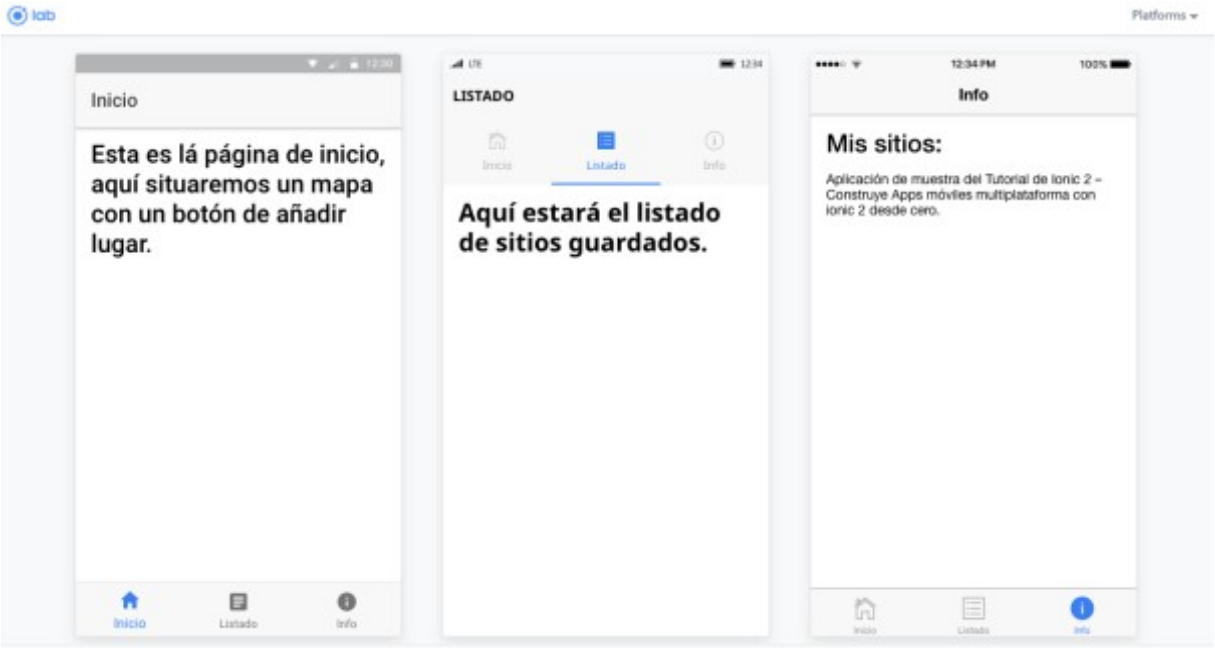
```
<!--
  Generated template for the Listado page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Listado</ion-title>
```

```
  </ion-navbar>

</ion-header>



<ion-content padding>
  <h1>Aquí estará el listado de sitios guardados.</h1>
</ion-content>
```

Y por último editamos **info.html** dejándolo de la siguiente manera:

```
<!--
  Generated template for the Info page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Info</ion-title>
  </ion-navbar>

</ion-header>



<ion-content padding>
<h1>Mis sitios:</h1>
<p>Aplicación de muestra del Tutorial de Ionic 2 - Construye Apps móviles
multiplataforma con ionic 2 desde cero.</p>
</ion-content>
```

Ahora al navegar por las pestañas podemos ver algo parecido a esto:



Nuestras tres pestañas en funcionamiento.

Como no quiero hacer posts demasiado largos de momento lo dejamos aquí. Hoy hemos aprendido a crear una aplicación con tabs para navegar entre diferentes paginas usando el comando **ionic g tabs**.

En el próximo post seguiremos creando nuestra aplicación para guardar sitios. Insertaremos un mapa en la página de inicio y capturaremos las coordenadas actuales.

Un saludo y hasta el próximo post.

P.D: si no quieres perderte los próximos posts **¡suscribete a mi blog!** 😉

7 - Crear una aplicación para guardar nuestros sitios geolocalizados

Parte 2: Mostrando el mapa.

Hola a todos:

En el post anterior creamos una aplicación con tres tabs y vimos como navegar entre ellas. Hoy vamos a seguir construyendo la aplicación.Lo siguiente que vamos a hacer es mostrar un mapa en la página de inicio centrado en las coordenadas actuales donde nos encontremos.

Lo primero que vamos a necesitar es cagar la librería de Google maps , si quieres saber más sobre la api de google maps puedes consultar este tutorial.

Editamos el archivo **index.html** que es la plantilla principal que se carga al iniciar la aplicación y añadimos la siguiente linea antes de cargar el script **build/main.js:**

```
<script  src="https://maps.google.com/maps/api/js"></script>
```

El código completo quedaría así:

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
  <meta charset="UTF-8">
  <title>Ionic App</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0,
minimum-scale=1.0, maximum-scale=1.0, user-scalable=no">
  <meta name="format-detection" content="telephone=no">
  <meta name="msapplication-tap-highlight" content="no">

  <link rel="icon" type="image/x-icon" href="assets/icon/favicon.ico">
```

```
  <link rel="manifest" href="manifest.json">
  <meta name="theme-color" content="#4e8ef7">

  <!-- cordova.js required for cordova apps -->
  <script src="cordova.js"></script>

  <!-- un-comment this code to enable service worker

    if ('serviceWorker' in navigator) {
      navigator.serviceWorker.register('service-worker.js')
        .then(() => console.log('serviceker installed'))
        .catch(err => console.log('Errorrr'));
    }
  -->

  <link href="build/main.css" rel="stylesheet">

</head>
<body>

  <!-- Ionic's root component and where the app will load -->
  <ion-app></ion-app>

  <!-- The polyfills js is generated during the build process -->
  <script src="build/polyfills.js"></script>

   <script  src="https://maps.google.com/maps/api/js"></script>

  <!-- The bundle js is generated during the build process -->
  <script src="build/main.js"></script>

</body>
</html>
```

Como vemos esta el la página principal donde se va a cargar todo el contenido de la app dentro de **<ion-app></ion-app>** que es componente raid de la aplicación.

Ahora para poder acceder a las coordenadas del móvil entra en juego **ionic native.**

Con **ionic native** lo que hacemos es instalar un plugin que nos da acceso a alguna característica nativa del móvil como la cámara, el gps, la brújula etc a través de typescript sin tener que programar nosotros en código nativo (Java para Android, Swift o objetive c para iOS).

Simplemente instalamos el plugin y luego importamos el plugin de **ionic-native**.

En la documentación de ionc 2 podemos ver todos los plugins disponibles en ionic native:

http://ionicframework.com/docs/v2/native/

En esta ocasión necesitamos el plugin **cordova-plugin-geolocation.**
Este plugin proporciona información sobre la ubicación del dispositivo, tales como la latitud y la longitud.

Para instalarlo desde el terminar dentro de la carpeta de nuestro proyecto tenemos que ejecutar el siguiente comando:

```
ionic plugin add cordova-plugin-geolocation
```

Si estás utilizando Linux o Mac y te sale un error al intentar instalar el plugin prueba a ejecutar el comando con **sudo** por delante.

Bien ahora vamos a importar el plugin en el controlador de la página inicio, para ello dentro de la carpeta **pages/inicio** editamos el archivo **inicio.ts** e importamos el plugin **Geolocation** de **ionic-native**.
También vamos a importar **Platform** desde **ionic-angular** para poder acceder al evento ready de Platform que se lanza cuando la aplicación se ha cargado completamente y esta lista. Esto nos evita errores al intentar llamar a un plugin antes de que se haya cargado.

Para importar estos dos módulos incluimos las siguientes lineas en los imports:

```
import { Geolocation } from 'ionic-native';
import { Platform } from 'ionic-angular';
```

Ahora cuando la aplicación esté completamente cargada y lista (con el evento ready de Platform) vamos a obtener las coordenadas donde nos encontramos y mostrar un mapa centrado en las coordenadas actuales.

Creamos una variable miembro llamada **map** del typo **any** (admite cualquier valor) que contendrá el manejador del mapa de google.
Después para poder utilizar Platform tenemos que inyectarlo como dependencia en el constructor de la clase. En el evento ready de Platform llamaremos a una función que vamos a llamar obtenerPosición. En estos momentos el código de **inicio.ts** debería quedarnos así:

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

import { Geolocation } from 'ionic-native';
import { Platform } from 'ionic-angular';



/*
  Generated class for the Inicio page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
*/
@Component({
  selector: 'page-inicio',
  templateUrl: 'inicio.html'
})
export class InicioPage {

  map: any; // Manejador del mapa.

  constructor(
    public navCtrl: NavController,
    public navParams: NavParams,
   public  platform: Platform){

     platform.ready().then(() => {
       // La plataforma esta lista y ya tenemos acceso a los plugins.
         this.obtenerPosicion();
     });
```

```
  }

  ionViewDidLoad() {
    console.log('ionViewDidLoad InicioPage');
  }

}
```

Hagamos un pequeño paréntesis en nuestra app para hablar de un par de conceptos que vamos a utilizar con frecuencia en nuestros desarrollos con ionic 2:

Promesas

Vemos que dentro del constructor lamamos al método **ready()** de **platform** y después de un punto utilizamos la palabra **then(...)**.

Esto es lo que llamamos una promesa.

Normalmente los plugins de **ionic-native** nos devuelven una promesa

Las promesas las incluía angular 1 pero ahora las promesas ya son nativas en EMACScript 6.

Las promesas vienen a sustituir a los **callbacks**. Se ejecuta la función y cuando obtenemos un resultado en el **then** se ejecuta el código que necesitemos.

Si falla la ejecución de alguno podemos utilizar la función **catch** para tratar el error.

Funciones de flecha

Otra cosa que te puede resultar rara es que dentro del **then** en lugar de ejecutar una función al uso estamos utilizando lo que se denomina funciones de flecha o en inglés (Fat Arrow functions).

La funciones con flecha permiten crear funciones anónimas más fácilmente y además permiten utilizar **this** en el contexto actual.
Vamos a ver un poco más en detalle esto con el siguiente ejemplo:

```
class MiClase {
  constructor() {
    this.nombre = 'Eduardo';
    setTimeout(() => {
      // Esto imprimé en la consola "Eduardo" ya que en las funciones con flecha this hace referencia al contexto actual.
      console.log(this.nombre);
    });
  }
}
```

Esto sería lo mismo que escribir lo siguiente:

```
class MiClase {
  constructor() {
    this.nombre = 'Eduardo';
    var _this = this;
    setTimeout(function() {
     console.log(_this.nombre);
    });
  }
}
```

Ahora que ya conocemos lo que son las promesas y las funciones de flecha podemos seguir con nuestra aplicación.

Antes de nada vamos a añadir una variable en en controlador que vamos a vamos a llamar **coords** y en ella guardaremos un objeto con la latitud y longitud donde nos encontramos.

Por lo tanto encima del constructor de la clase después de la variable map definimos la variable **coords**:

```
...

export class InicioPage {

  map: any; // Manejador del mapa.
  coords : any = { lat: 0, lng: 0 }

  constructor(

...
```

Si el editor os marca los errores observareis que os subraya **this.obtenerPosicion()**, esto es evidentemente porque estamos llamando a una función que todavía no hemos definido.

Vamos a definir la función **obtenerPosición()**:

```
obtenerPosicion():any{
    Geolocation.getCurrentPosition().then(res => {
      this.coords.lat = res.coords.latitude;
      this.coords.lng = res.coords.longitude;

      this.loadMap();
    })
    .catch(
      (error)=>{
        console.log(error);
      }
    );
  }
```

La función **getCurrentPosition()** del plugin Geolocation nos devuelve una promesa.

Como podemos observar resolvemos la promesa con una función de flecha. En la función de flecha recibimos como parámetro el objeto **res**. Lo que nos interesa obtener es la longitud y latitud donde nos encontramos, estos valores están
en **res.coords.longitude** y **res.coords.latitude**.

Asignamos esos valores al objeto de la variable **this.cords** que acabamos de definir y después llamamos a la función **this.loadMap()** que aún no hemos creado.

Recordad que debemos de utilizar **this** para hacer referencia a las variables miembro y métodos que definamos en la clase del controlador.

Ahora deberemos crear la función **loadMap** que se encargará de mostrar un mapa en la página centrado en las coordenadas que hemos recogido.

Antes de definir la función necesitamos crear el contenedor donde se va a mostrar el mapa en la vista. La librería javascript de Google maps lo que hace es insertar un mapa en un div, por lo que necesitamos crear ese div en la vista y luego pasárselo como referencia para crear el mapa.

Editamos la vista de la página inicio, es decir el archivo **inicio.html** y añadimos un div dentro de **ion-content** al que le asignamos como id “map”:

```
<!--
  Generated template for the Inicio page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Inicio</ion-title>
  </ion-navbar>

</ion-header>


<ion-content padding>

  <div id="map"></div>

</ion-content>
```

Ahora tenemos que asignarle un tamaño al mapa así que por primera vez vamos a editar el archivo de estilos de una página, en este caso vamos a editar el archivo **inicio.scss.**

La extensión del archivo **.scss**, hace referencia a que es una archivo Sass, sus siglas hacen referencia a (*Syntactically Awesome Stylesheets)* algo así como "Hoja de estilo sintácticamente impresionante".

Sass es un lenguaje de hoja de estilos que extiende el css tradicional proveyendo de varios mecanismos que están presentes en los lenguaje de programación. Con Sass puedes utilizar variables, código anidado, mixins, etc.

Ionic ya viene con Sass instalado lo que hace realmente fácil su utilización. Sass es un lenguaje de script que es traducido a css. Podemos añadir reglas de estilo de igual manera que lo hacemos con css por lo que de momento para definir el tamaño del mapa no necesitamos saber mas. Si quieres más información sobre Sass puede consultar la documentación oficial: http://sass-lang.com/documentation/

Veamos como tiene que quedar el archivo **inicio.scss:**

```
page-inicio {
ion-content{
    #map {
      width: 100%;
      height: 100%;
    }
  }
}
```

Como podemos ver dentro del elemento **page-inicio** definimos el estilo para **ion-content** y a su vez dentro definimos el estilo para **#map** al que le estamos diciendo que ocupe todo al ancho y el alto de la página, como vemos Sass nos permite anidar los elementos.

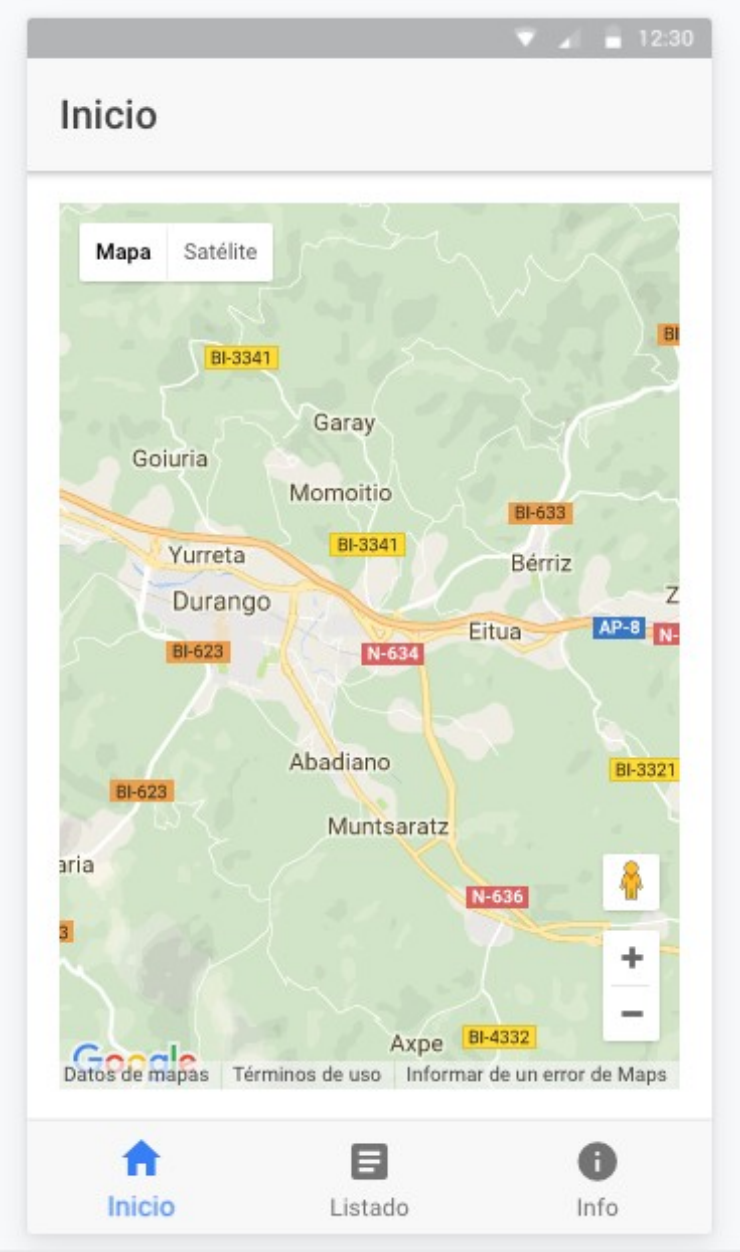
Bien, una vez hechos estos preparativos ya podemos definir la función **loadMap** que se encargará de mostrar un mapa en la pagina centrado en las coordenadas que hemos recogido. Para que Typescript no de error por no reconocer la clase google cuando la llamemos desde la función que vamos a crear, vamos a declarar la variable **google** justo debajo de los imports con declare var google: any; . Veamos como tiene que quedar el código de**inicio.ts** con la variable google definida y nuestra función **loadMap**:

```
import { Component } from '@angular/core';
import { NavController, NavParams } from 'ionic-angular';
import { Geolocation } from 'ionic-native';
import { Platform } from 'ionic-angular';

declare var google: any;

/*
  Generated class for the Inicio page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
*/
@Component({
  selector: 'page-inicio',
  templateUrl: 'inicio.html'
})
export class InicioPage {

  map: any; // Manejador del mapa.
  coords : any = { lat: 0, lng: 0 }
  address: string;

  constructor(
    public navCtrl: NavController,
    public navParams: NavParams,
    platform: Platform) {

      platform.ready().then(() => {
      // La plataforma esta lista y ya tenemos acceso a los plugins.
        this.obtenerPosicion();
      });

  }

  obtenerPosicion():any{
    Geolocation.getCurrentPosition().then(res => {
      this.coords.lat = res.coords.latitude;
      this.coords.lng = res.coords.longitude;
```

```
      this.loadMap();
    })
    .catch(
      (error)=>{
        console.log(error);
      }
    );
  }


loadMap(){
    let mapContainer = document.getElementById('map');
    this.map = new google.maps.Map(mapContainer, {
      center: this.coords,
      zoom: 12
    });
}


  ionViewDidLoad() {
    console.log('ionViewDidLoad InicioPage');
  }

}
```

En la función **loadMap()** asignamos a la variable **mapContainer** el elemento div con id=”map” que habíamos creado en la vista y creamos un mapa con **google.maps.Map** donde le pasamos dos parámetros, el primero es el elemento contenedor del mapa que lo hemos recogido en la variable **mapContainer** y el segundo parámetro es un objeto donde le pasamos la configuración del mapa, en este caso le pasamos las coordenadas al parámetro **center** y le decimos que muestre el mapa con un **zoom** de 12.

En este punto si probamos nuestra aplicación en el navegador con **ionic serve -l**deberíamos ver algo como esto:



Pantalla de Inicio con nuestro mapa

Para no hacer demasiado largo este post lo vamos a dejar aquí.

En el siguiente post seguiremos desarrollando la app, veremos como añadir un botón FAB, como añadir un marcador al mapa y como crear una ventana modal y pasarle datos desde el controlador de la página que hace la llamada.

P.D: si no quieres perderte los próximos posts **¡suscribete a mi blog!** 😉

8 - Crear una aplicación para guardar nuestros sitios geolocalizados

Parte 3: Añadiendo FAB, marcador y ventana modal.

Hola a todos:

En el post anterior vimos como insertar en nuestra app un mapa de google maps centrado en las coordenadas actuales.
Vamos a continuar desarrollando nuestra app.

Para poder apreciar mejor donde estamos situados vamos a mostrar un marcador personalizado en el mapa que nos indique nuestra posición.

Para el marcado vamos a utilizar una imagen personalizada. La forma correcta de utilizar imágenes locales en nuestra app es alojarlas en la carpeta **src/assets**, así que vamos a crear dentro de src/assets una carpeta la la que llamaremos **img** donde alojaremos nuestras imágenes.

Podemos poner la imagen que queramos como marcador para el mapa. Si no os queréis complicar podéis dar botón derecho sobre la siguiente imagen y descargarla para utilizarla en este ejemplo:

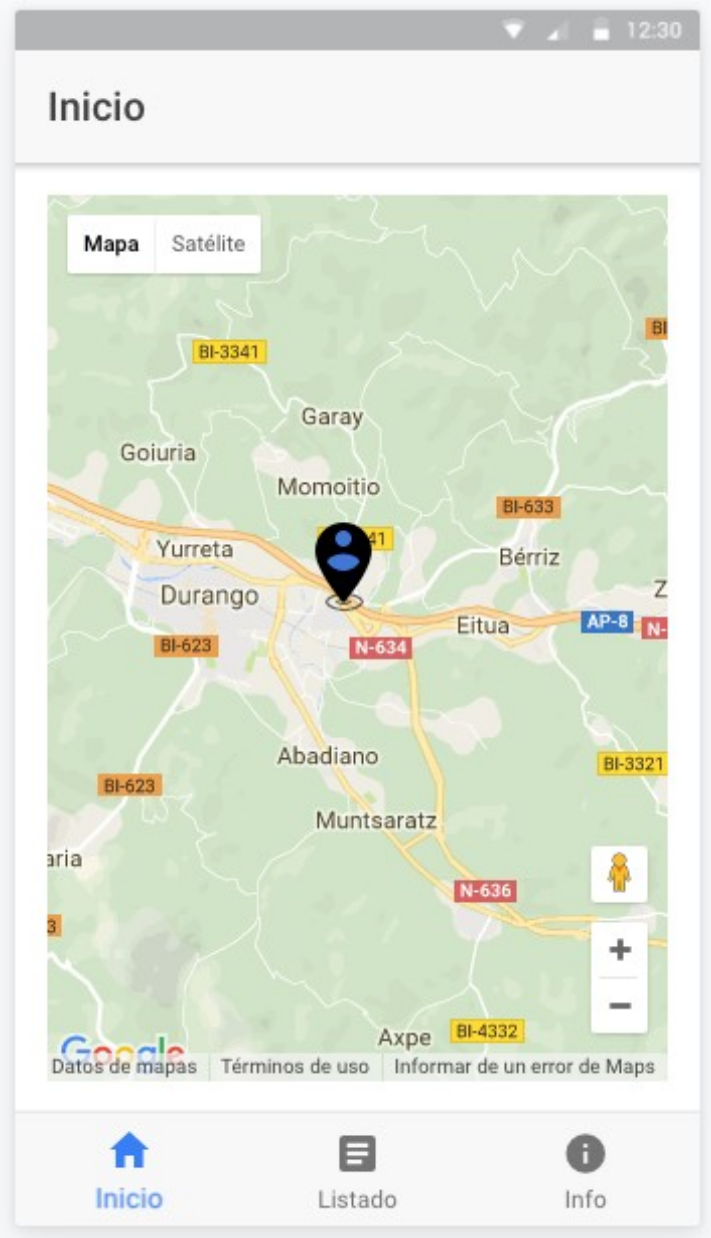
Bien, una vez descargada la imagen la copiamos en la carpeta **img** que acabamos de crear.

Ahora para situar el marcador en el mapa editamos el archivo **inicio.ts** y en la función **loadMap()** que muestra el mapa añadimos el siguiente código:

```
loadMap(){

   let mapContainer = document.getElementById('map');
    this.map = new google.maps.Map(mapContainer, {
      center: this.coords,
      zoom: 12
    });

    // Colocamos el marcador
    let miMarker = new google.maps.Marker({
                icon : 'assets/img/ico_estoy_aqui.png',
                map: this.map,
                position: this.coords
           });

}
```

No hay mucho que comentar, como veis para crear un marcador creamos un objeto **google.maps.Marker** y le pasamos un objeto como parámetro donde en **icon** le indicamos donde se aloja la imagen del icono, en este caso la ruta a la imagen que hemos guardado destro de assets en la carpeta img es '**assets/img/ico_estoy_aqui.png'**, si vuestra imagen se llama de otra forma tenéis lógicamente que poner el nombre de la imagen que vayáis a utilizar.
En **map** le idicamos la variable que contiene el mapa donde se tiene que situar, es este caso **this.map**, por ultimo en **position** le asignamos las coordenadas donde se tiene que situar, en este caso le pasamos la variable **this.coords** que contiene las coordenadas actuales.

Ahora si ejecutamos **ionic serve -l** veremos en el navegador algo como esto:



Mostrando un marcador en el mapa

Ahora vamos a dar un paso más y vamos a añadir un **FAB** (Floating Action Button) es decir botón de acción flotante al mapa.Los FAB son componentes estándar de material design, tienen la forma de un círculo y flotan sobre el contenido en una posición fija.

Este FAB lo utilizaremos para añadir la posición actual a nuestros sitios, para ello haremos que cuando se pulse en el fav se habra una ventana modal donde mostraremos un pequeño formulario donde aparecerán las coordenadas y la dirección de la posición actual y nos permitirá añadir una descripción y una fotografía desde la cámara de nuestro móvil.

Vayamos por partes.

Primero vamos a colocar el FAB en la vista de muestra página de inicio, editamos el archivo **inicio.html** y añadimos lo que esta marcado con fondo amarillo.

```
<!--
  Generated template for the Inicio page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Inicio</ion-title>
  </ion-navbar>

</ion-header>

<ion-content padding>
  <div id="map"></div>
   <ion-fab right top>
     <button ion-fab (tap)="nuevoSitio()">
       <ion-icon name="pin"></ion-icon>
       <ion-icon name="add"></ion-icon>
     </button>
  </ion-fab>
  </ion-content>
```

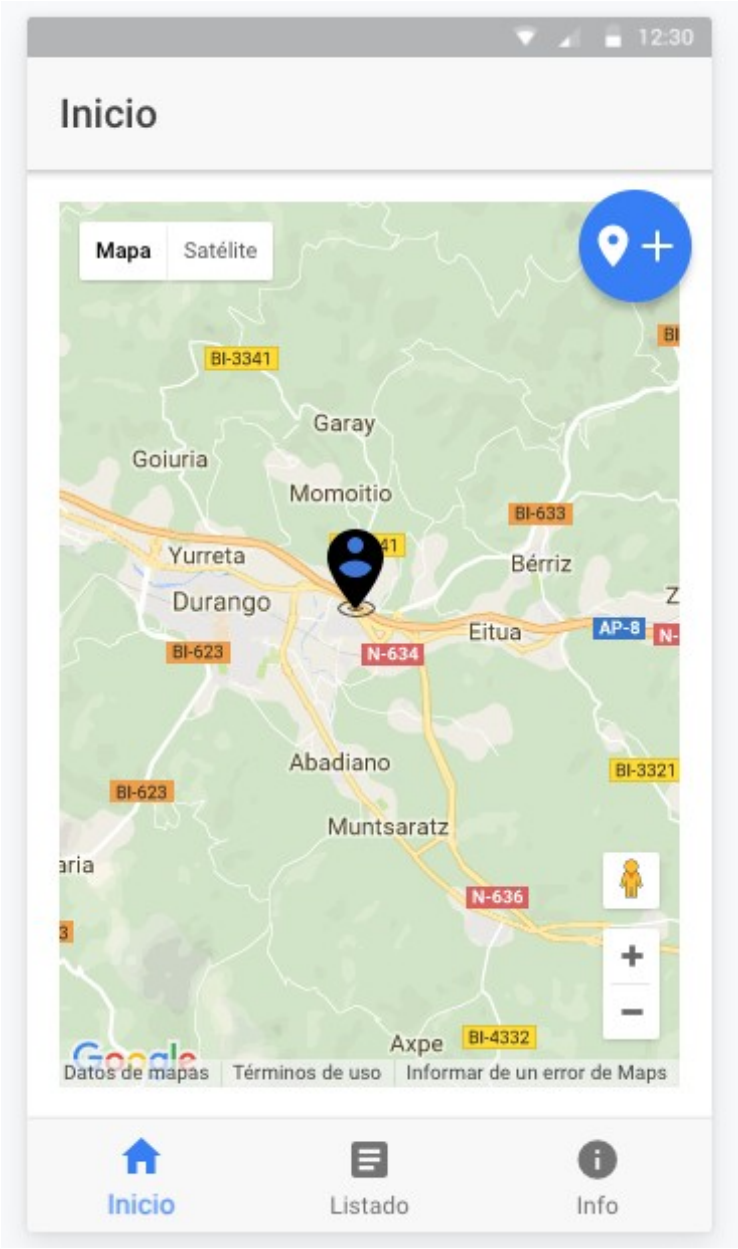
Bien, como vemos tenemos el componente ion-fab al que le indicamos que se sitúe arriba a la derecha con **right** y **top**.

En su interior contiene un botón al que le tenemos que indicar que es del tipo **ion-fab**.

Con **(tap)=”nuevoSitio()”** le indicamos que cuando se pulse o tape el botón se llame a la función **nuevoSitio()** que definiremos luego en el controlador de la página.

Después tenemos dos componentes **ion-icon** para mostrar los iconos en el FAB, uno con el icono pin y otro con el icono add, lo habitual es mostrar un solo icono, pero he querido poner dos para que quede más claro que queremos añadir una localización.

Si probamos ahora nuestra app tendrá un aspecto similar a este:



Bien, ahora vamos a definir la función **nuevoSitio()** en el controlador, editamos el archivo **inicio.ts** y añadimos la siguiente función debajo de la función **loadMap**:

```
nuevoSitio(){
 // aquí vamos a abrir el modal para añadir nuestro sitio.
}
```

La idea es que al llamar a esta función desde el FAB se abra un modal para poder añadir una descripción y una foto a muestro sitio si lo deseamos, vamos a explicar un poco que son los modales y como se utilizan:

Modales

Los modales son como ventanas que se abren dentro de nuestra aplicación sin que afecten a la pila de navegación.

Para crear un modal debemos de crear una página con el ionic generator.
Desde consola vamos a escribir el siguiente comando para crear el modal donde irá el formulario para introducir el nuevo sitio:

```
ionic g page modalNuevoSitio
```

Al igual que el resto de páginas que hemos creado para esta aplicación debemos importarla y declararla en **app.module.ts** que quedará de la siguiente manera:

```
import { NgModule, ErrorHandler } from '@angular/core';
import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';
import { MyApp } from './app.component';
import { MisTabsPage } from '../pages/mis-tabs/mis-tabs';
import { InicioPage } from '../pages/inicio/inicio';
import { ListadoPage } from '../pages/listado/listado';
import { InfoPage } from '../pages/info/info';
import { ModalNuevoSitioPage } from '../pages/modal-nuevo-sitio/modal-nuevo-sitio';



@NgModule({
  declarations: [
    MyApp,
    MisTabsPage,
    InicioPage,
    ListadoPage,
    InfoPage,
    ModalNuevoSitioPage
```

```
  ],
  imports: [
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    MisTabsPage,
    InicioPage,
    ListadoPage,
    InfoPage,
    ModalNuevoSitioPage
  ],
  providers: [{provide: ErrorHandler, useClass: IonicErrorHandler}]
})
export class AppModule {}
```

Ahora en **inicio.ts** debemos importar la página que acabamos de crear:

```
import { ModalNuevoSitioPage } from '../modal-nuevo-sitio/modal-nuevo-
sitio';
```

Recordad que no hay que añadir la extensión **.ts** después del nombre del archivo.

También debemos importar el componente **ModalController** de la librería **ionic-angular**, por lo tanto después de **NavController** y **NavParams** importamos también ModalController.

```
import { NavController, NavParams, ModalController } from 'ionic-angular';
```

Debemos inyectar también en el constructor el componente **ModalController** al que hemos llamado **modalCtrl:**

```
constructor(
    public navCtrl: NavController,
    public navParams: NavParams,
    public modalCtrl : ModalController,
    platform: Platform) {
```

```
    ......
```

Ahora ya estamos listos para crear el modal en nuestra función **nuevoSitio**.

Para crear un modal utilizamos el método **create** del componente **ModalController** y le pasamos como primer parámetro el controlador de la página que hemos creado para el modal y que hemos llamado **ModalNuevoSitioPage**.
El segundo parámetro es opcional y se utiliza para pasarle datos a nuestro modal, en este caso lo vamos a utilizar para pasarle el objeto **this.coords** que contiene las coordenadas que hemos obtenido.

Una vez creado el modal para que se muestre en pantalla invocamos al método **present()**;

```
nuevoSitio(){
  // aquí vamos a abrir el modal para añadir nuestro sitio.
   let mimodal = this.modalCtrl.create( ModalNuevoSitioPage,this.coords );
   mimodal.present();
}
```

Ahora que sabemos como mostrar un modal y como pasarle datos desde la página que lo llama, vamos a ver como recibimos esos datos y los mostramos en el modal.

Vamos a crear una variable en el controlador del modal (**modal-nuevo-sitio.html)** que al igual que en la página inicio llamaremos coords y sera de tipo any, esta variable al igual que en la página de inicio va a contener un objeto con la latitud y longitud que recibimos en la llamada.

Para recibir los datos desde la página que llama al modal solo tenemos que utilizar el método **get** de **NavParams** que ya se importa por defecto al crear una página. Con navParams controlamos los parámetros que recibimos.
Dentro de la función **ionViewDidLoad** que se ejecuta al cargar la vista vamos a asignar a cada atributo del objeto **coords** que acabamos de crear el valor que corresponde de la latitud y longitud que recibimos.
Veamos como tiene que quedar el código de **modal-nuevo-sitio.ts** en estos momentos:

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

```
/*
  Generated class for the ModalNuevoSitio page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
*/
@Component({
  selector: 'page-modal-nuevo-sitio',
  templateUrl: 'modal-nuevo-sitio.html'
})
export class ModalNuevoSitioPage {

  coords : any = { lat: 0, lng: 0 }

  constructor(public navCtrl: NavController, public navParams: NavParams,
private viewCtrl : ViewController) {}

  ionViewDidLoad() {
    console.log('ionViewDidLoad ModalNuevoSitioPage');
    this.coords.lat = this.navParams.get('lat');
    this.coords.lng = this.navParams.get('lng');
  }
}
```

Ahora que hemos recibido las coordenadas desde la página inicio vamos a mostrarlas en el modal para comprobar que las recibimos correctamente, para ello vamos a editar el archivo **modal-nuevo-sitio.html** y simplemente añadimos las coordenadas dentro de **ion-content**:

```
<ion-content padding>
  <p>Hemos recibido: {{ coords.lat }}, {{ coords.lng }}<p>
</ion-content>
```

Si probamos ahora muestra aplicación observamos que al pulsar el botón (FAB) que hemos creado para añadir sitios nos abre una ventana modal donde se muestran las coordenadas que acabamos de recibir, pero tenemos un pequeño problema, y es que no tenemos forma de

cerrar el modal, así que antes de editar cualquier otra cosa en la página vamos a crear un botón de cerrar.

Para cerrar el modal tenemos que importar en el controlador de la pagina del modal (**modal-nuevo-sitio.ts**) el controlador **ViewController** e inyectarlo en el constructor.

Luego creamos una función que vamos a llamar cerrarModal donde utilizaremos el método **dismiss** de **ViewController** para cerrarlo:

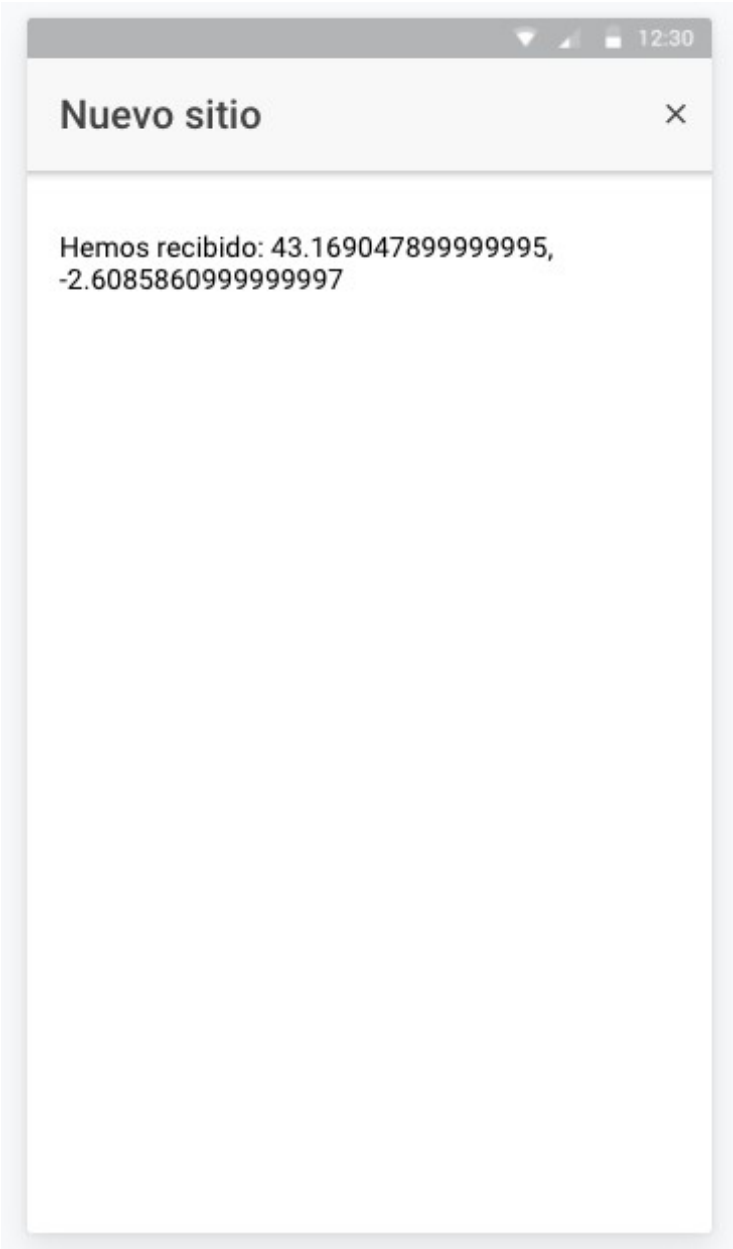
```
import { Component } from '@angular/core';
import { NavController, NavParams, ViewController } from 'ionic-angular';



/*
  Generated class for the ModalNuevoSitio page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
*/
@Component({
  selector: 'page-modal-nuevo-sitio',
  templateUrl: 'modal-nuevo-sitio.html'
})
export class ModalNuevoSitioPage {

  coords : any = { lat: 0, lng: 0 }
  constructor(public navCtrl: NavController, public navParams: NavParams,
private viewCtrl : ViewController ) {}

  ionViewDidLoad() {
    console.log('ionViewDidLoad ModalNuevoSitioPage');
    this.coords.lat = this.navParams.get('lat');
    this.coords.lng = this.navParams.get('lng');
  }

  cerrarModal(){
    this.viewCtrl.dismiss();
```

```
  }

}
```

Ahora vamos a añadir el botón de cerrar que llamará a la función **cerrarModal** que acabamos de crear en la cabecera de la vista de la página del modal (**modal-nuevo-sitio.html**), aprovechamos tambien para cambiar el título del modal a “Nuevo sitio”:

```
<!--
  Generated template for the ModalNuevoSitio page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Nuevo sitio</ion-title>
   <ion-buttons start>
      <button ion-button (click)="cerrarModal()">
        <ion-icon name="md-close"></ion-icon>
      </button>
    </ion-buttons>
  </ion-navbar>

</ion-header>

<ion-content padding>
  <p>Hemos recibido: {{ coords.lat }}, {{ coords.lng }}<p>
</ion-content>
```

Ahora ya tenemos un botón de cerrar para nuestro modal, si habéis seguido correctamente todos los pasos al pulsar sobre el FAV se abrirá nuestro modal como en la siguiente imagen:



Por ahora lo dejamos aquí, hoy hemos aprendido como añadir un marcador al mapa de Google maps, como añadir a nuestra app botones tipo FAB, y como crear, abrir, pasar datos y cerrar un modal.
En el próximo post veremos como añadir un nuevo con descripción y foto, haciendo uso de la cámara del móvil.

Si aún no lo has hecho no olvides **suscribirte a mi blog** para no perderte los próximos posts ;)

9 – Crear una aplicación para guardar nuestros sitios geolocalizados

Parte 4 – Mostrando la dirección a partir de las coordenadas y sacando foto con la cámara.

Hola a todos:

En el post anterior vimos como poner un marcador al mapa, aprendimos a utilizar los botones FAB y aprendimos a utilizar ventanas modales.

Hoy vamos a continuar desarrollando nuestra app, vamos a crear un pequeño formulario en el modal donde mediante una llamada a la api de google maps obtendremos y mostraremos la dirección a partir de las coordenadas y permitiremos introducir una descripción y también tomar una fotografía del lugar.

Bien, vayamos por partes:

Ahora vamos a añadir al controlador del modal tres variables nuevas que vamos a necesitar:

- Una variable de tipo **string** que vamos a llamar **address** donde guardaremos la dirección que luego mostraremos en la vista.

- Una variable de tipo **string** a la que vamos a llamar **description** y que contendrá la descripción del lugar que introduzcamos desde el formulario.

- Por último una variable de tipo any que vamos a llamar foto, donde guardaremos una foto del lugar codificada en base 64.

  Por lo tanto editamos el controlador de nuestro modal (**modal-nuevo-sitio.ts**) encima del constructor de la clase después de la variable **coords** definimos las siguientes variables:

```
...

export class ModalNuevoSitioPage {

  coords : any = { lat: 0, lng: 0 }
  address: string;
  description: string = '';
  foto: any = '';

  constructor(public navCtrl: NavController, public navParams: NavParams,
private viewCtrl : ViewController) {}

...
```

Para obtener la dirección correspondiente a unas coordenadas Google maps cuenta con el elemento **Geodecoder.**

Vamos a ver brevemente como funciona Google Maps Geodecoder:

Para realizar una petición debemos crear un objeto dela clase **Geodecoder** y llamar al método **geodecode** pasandole las coordenadas y una función callback donde recibimos los datos en caso de tener éxito y el estado de la petición:

```
var geocoder = new google.maps.Geocoder();

geocoder.geocode({'location': coords} , function (results, status) {
    if (status == google.maps.GeocoderStatus.OK) {
       // en results tenemos lo que nos devuelve la llamada;
    } else {
      // Ha habido algún error
    }
});
```

Para obtener los resultados tenemos pasarle una función callback, sin embargo lo que nos interesa en que nos devuelva una promesa que es la mejor manera de gestionar las peticiones asíncronas desde el controlador.

Creando nuestras propias promesas

Ya vimos en el capitulo anterior como se tratan las funciones que nos devuelve una promesa utilizando **then**, pero si queremos que una función que creemos nosotros devuelva una promesa tenemos que devolver un objeto **Promise** donde la promesa se crea a partir de una función **callback** en el que ejecutaremos la sentencia que nos devolverá el resultado asíncrono y llamaremos a las funciones pasadas como argumento **resolve** y **reject**. Si la operación se a ejecutado correctamente se llama a **resolve**, y si a ocurrido un error llamamos a **reject.**

Sabiendo esto para conseguir que geodecode nos devuelva la dirección en una promesa tenemos que utilizar un pequeño truco que consiste en crear una función que vamos a llamar **getAddress** y que haremos que nos devuelva una promesa con el resultado de la llamada a geodecode, para ello editamos el archivo **modal-nuevo-sitio.ts** añadimos la siguiente función en el controlador:

```
getAddress(coords):any {
    var geocoder = new google.maps.Geocoder();

    return new Promise(function(resolve, reject) {
        geocoder.geocode({'location': coords} , function (results, status) {
// llamado asincronamente
            if (status == google.maps.GeocoderStatus.OK) {
                resolve(results);
            } else {
                reject(status);
            }
        });
    });
}
```

Como podemos ver la función retorna una promesa donde se le pasa como parámetro una función con **resolve** y **reject**, luego dentro de la función se ejecuta la llamada a **geodecoder.geodecode** pasándole las coordenadas y la función callback donde si se ha recibido como status **google.maps.GeocoderStatus.OK** significa que hemos recibido correctamente los datos y entonces ejecutamos **resolve**, de lo contrario ejecutamos **reject**.

De esta manera podemos hacer que geodecode nos devuelva una promesa.

Ahora en el método **ionViewDidLoad** que se ejecuta cuando la página se ha cargado vamos ha hacer una llamada a **getAddress** pasándole las coordenadas que hemos recibido para obtener la dirección y asignársela a **this.address**:

```
ionViewDidLoad() {
    console.log('ionViewDidLoad ModalNuevoSitioPage');
    this.coords.lat = this.navParams.get('lat');
    this.coords.lng = this.navParams.get('lng');
   this.getAddress(this.coords).then(results=> {
        this.address = results[0]['formatted_address'];
      }, errStatus => {
          // Aquí iría el código para manejar el error
      });
  }
```

Un ejemplo de la estructura de datos que recibimos en **results** sería:

```
{
 "results": [ {
 "types": street_address,
 "formatted_address": "Etorbidea Abandoibarra, 2, 48001 Bilbo, Bizkaia,
España",
 "address_components": [ {
 "long_name": "2",
 "short_name": "2",
 "types": street_number
 }, {
 "long_name": "Etorbidea Abandoibarra",
 "short_name": "Etorbidea Abandoibarra",
 "types": route
 }, {
 "long_name": "Bilbo",
 "short_name": "Bilbo",
 "types": [ "locality", "political" ]
 }, {
 "long_name": "Bizkaia",
 "short_name": "BI",
 "types": [ "administrative_area_level_2", "political" ]
 }, {
```

```
"long_name": "Euskadi",
"short_name": "PV",
"types": [ "administrative_area_level_1", "political" ]
}, {
"long_name": "España",
"short_name": "ES",
"types": [ "country", "political" ]
}, {
"long_name": "48001",
"short_name": "48001",
"types": postal_code
} ],
"geometry": {
"location": {
"lat": 43.26861,
"lng": -2.934380000000033
},
"location_type": "ROOFTOP",
"viewport": {
"southwest": {
"lat": 43.26726101970851,
"lng": -2.9357289802915147
},
"northeast": {
"lat": 43.26995898029151,
"lng": -2.933031019708551
}
}
}
} ]
}
```


Lo que nos interesa obtener que es la dirección completa se encuentra en **results[0] [‘formatted_address’]**, por lo tanto le asignamos este dato a **this.address.**

Mostrando las coordenadas y la dirección.

Ahora vamos a añadir en la vista un componente **ion-card** donde mostraremos las coordenadas y la dirección donde nos encontramos, editamos el archivo **modal-nuevo-sitio.html** para que quede de la siguiente manera:

```
<!--
  Generated template for the ModalNuevoSitio page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Nuevo sitio</ion-title>
    <ion-buttons start>
      <button ion-button (click)="cerrarModal()">
        <ion-icon name="md-close"></ion-icon>
      </button>
    </ion-buttons>
  </ion-navbar>

</ion-header>

<ion-content padding>
  <ion-card>
    <ion-card-header>
      Localización actual
    </ion-card-header>
    <ion-card-content>
      <p><strong>lat:</strong>{{ coords.lat}}<br/>
      <strong>lng:</strong>{{coords.lng }}</p>
      <hr>
      <p>{{ address }}</p>
    </ion-card-content>
  </ion-card>
</ion-content>
```

Como podéis observar hemos creado un componente **ion-card** que consta a su vez de un elemento **ion-card-header** donde ponemos como título “Localización actual”, y en **ion-car-content** pondremos el contenido que queremos mostrar, en este caso mostramos un elemento **<p>** donde mostramos el valor de las variables **coords.lat** y **coords.lng** que hemos definido en el controlador y que contendrán las coordenadas actuales.

Por otro lado mostramos otro elemento **<p>** con el contenido de la variable **address** que de momento no contiene nada pero que contendrá la dirección que corresponda con las coordenadas que hemos recogido.

Al pulsar en el FAB se abrirá el modal mostrando algo similar a esto:

Modal con las coordenadas y a dirección

Creando el formulario.

Vamos a seguir añadiendo elementos a la vista del modal. Además de las coordenadas y la dirección queremos dar la posibilidad de tomar una foto del lugar y escribir anotaciones. Para ello vamos a crear dentro del card, debajo de la dirección un pequeño formulario donde habrá un botón para sacar una foto y un campo text-area para escribir una descripción del lugar.

Editamos de nuevo **modal-nuevo-sitio.html** y añadimos lo que está marcado de amarillo:

```
<!--
  Generated template for the ModalNuevoSitio page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Nuevo sitio</ion-title>
    <ion-buttons start>
      <button ion-button (click)="cerrarModal()">
        <ion-icon name="md-close"></ion-icon>
      </button>
    </ion-buttons>
  </ion-navbar>

</ion-header>

<ion-content padding>
  <ion-card>
    <ion-card-header>
     <strong>Localización actual</strong>
    </ion-card-header>
    <ion-card-content>
      <p><strong>lat:</strong>{{ coords.lat}}<br/>
      <strong>lng:</strong>{{coords.lng }}</p>
      <hr>
```

```
      <p>{{ address }}</p>
      <hr/>
      <form (ngSubmit)="guardarSitio()">
      <ion-item>
        <img [src]="foto" *ngIf="foto" />
        <button ion-button icon-left full type="button" (tap)="sacarFoto()">
          Foto  
          <ion-icon name="camera"></ion-icon>
        </button>
      </ion-item>
      <hr/>
      <ion-item>
        <ion-label>Descripción</ion-label>
        <ion-textarea [(ngModel)]="description" name="description"></ion-
textarea>
      </ion-item>
      <button ion-button type="submit" block>Guardar Sitio</button>
    </form>
    </ion-card-content>
    </ion-card>
</ion-content>
```

Como vemos al formulario le hemos añadido (**ngSubmit)**, con esto le indicamos que cuando se ejecute el evento submit del formulario se ejecute la función **guardarSitio**definida en el controlador.

Después hemos añadido un elemento **ion-item**, donde vamos a mostrar la imagen con la fotografía que tomemos:

```
<img [src]="foto" *ngIf="foto" />
```

Por un lado con **[src]** le indicamos que la imagen va a mostrar lo que contenga la variable **foto**, que hemos definido el el controlador y que al principio estará vacía, y por otro lado con ***ngIf** le indicamos que solo se muestre la imagen si dicha variable **foto** tiene un valor, es decir que solo se mostrará una vez hayamos tomado la fotografía.

Después tenemos un botón que al pulsar llama a la función **sacarFoto** que después definiremos en el controlador:

```
<button ion-button icon-left full type="button" (tap)="sacarFoto()">
          Foto  
          <ion-icon name="camera"></ion-icon>
 </button>
```

Con **icon-left** le indicamos al botón que el icono se situará a la izquierda, para mostrar el icono después del texto "Foto" y un par de espacios en blanco "** **" insertamos el elemento **ion-icon** con el icono camera.

Después tenemos un campo de tipo **ion-textarea** donde podremos introducir la descripción del sitio:

```
<ion-item>
    <ion-label>Descripción</ion-label>
    <ion-textarea [(ngModel)]="description" name="description"></ion-
textarea>
</ion-item>
```

Por último tenemos un botón de tipo submit para enviar el formulario. Recordad que cuando pulsemos en este botón se disparará el evento **ngSubmit** y por lo tanto se ejecutará la función **guardarSitio** tal y como hemos definido en la etiqueta **form**.

Utilizando la cámara del teléfono móvil

Ahora vamos a ver como sacar una foto y para que se muestre en la etiqueta img que hemos creado:

Para poder utilizar la cámara del móvil tenemos que instalar el plugin **cordova-plugin-camera** con el siguiente comando:

```
ionic plugin add cordova-plugin-camera
```

Ahora el controlador del modal (**modal-nuevo-sitio.ts**) tenemos que importar **Camera**desde **ionic-native:**

```
import {Camera} from 'ionic-native';
```

Ahora vamos a crear el método **sacarFoto** en el controlador que se ejecutará al pulsar sobre el botón **Foto**.

En el archivo **modal-nuevo-sitio.ts** añadimos el siguiente código dentro del controlador:

```
sacarFoto(){

    let cameraOptions = {
        quality: 50,
        encodingType: Camera.EncodingType.JPEG,
        targetWidth: 800,
        targetHeight: 600,
        destinationType: Camera.DestinationType.DATA_URL,
        sourceType: Camera.PictureSourceType.CAMERA,
        correctOrientation: true
    }


    Camera.getPicture(cameraOptions).then((imageData) => {
      // imageData is a base64 encoded string
        this.foto = "" + imageData;
    }, (err) => {
        console.log(err);
    });
  }
```

Para sacar una foto utilizamos el método **getPicture** pasándole un array de opciones. En las opciones definimos las características que va a tener la imagen:

- **encodingType**: Selecciona la codificación del archivo de imagen devuelto, puede ser JPEG o PNG.

- **targetWidth**: Anchura de la foto.

- **targetHeight**: Altura de la foto.

- **destinationType**: Define el formato del valor devuelto, puede ser :

- DATA_URL devuelve la imagen como una cadena codificada en base64.
- FILE_URI: Crea un archivo con la imagen y devuelve la ruta al archivo.
- NATIVE_URI: devuelve la ruta nativa al archivo (assets-library:// en iOS o content:// en Android).

- **sourceType**: Indica el origen de la foto, puede ser:

- CAMERA (por defecto).
- PHOTOLIBRARY
- SAVEDPHOTOALBUM

- **correctOrientation**: Gira la imagen para corregir la orientación del dispositivo durante la captura.

  Puedes visitar el siguiente enlace para conocer todas las opciones posibles y saber más sobre el plugin Camera:

  https://ionicframework.com/docs/v2/native/camera/

  En **this.foto** guardamos la imagen codificada en formato base64 que recibimos de la cámara, para poder mostrarla como parte de la url en el parámetro src de la imagen tenemos que añadirle "**"** por delante.

  Por si alguno se ha perdido muestro el contenido completo de como tiene que quedar en estos momentos el archivo **modal-nuevo-sitio.ts:**

```
import { Component } from '@angular/core';
import { NavController, NavParams,  ViewController } from 'ionic-angular';
import { Camera } from 'ionic-native';

declare var google: any;

/*
  Generated class for the ModalNuevoSitio page.
```

```
  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
*/
@Component({
  selector: 'page-modal-nuevo-sitio',
  templateUrl: 'modal-nuevo-sitio.html'
})
export class ModalNuevoSitioPage {

  coords : any = { lat: 0, lng: 0 }
  address: string;
  description: string = '';
  foto: any = '';

  constructor(public navCtrl: NavController, public navParams: NavParams,
private viewCtrl : ViewController ) {}

  ionViewDidLoad() {
    console.log('ionViewDidLoad ModalNuevoSitioPage');
    this.coords.lat = this.navParams.get('lat');
    this.coords.lng = this.navParams.get('lng');
    this.getAddress(this.coords).then(results=> {
        this.address = results[0]['formatted_address'];
      }, errStatus => {
          // Aquí iría el código para manejar el error
      });
  }

  cerrarModal(){
    this.viewCtrl.dismiss();
  }

  getAddress(coords):any {
    var geocoder = new google.maps.Geocoder();

    return new Promise(function(resolve, reject) {
        geocoder.geocode({'location': coords} , function (results, status) {
// llamado asincronamente
            if (status == google.maps.GeocoderStatus.OK) {
                resolve(results);
```

```
            } else {
                reject(status);
            }
        });
    });
  }

  sacarFoto(){

    let cameraOptions = {
        encodingType: Camera.EncodingType.JPEG,
        targetWidth: 800,
        targetHeight: 600,
        destinationType: Camera.DestinationType.DATA_URL,
        sourceType: Camera.PictureSourceType.CAMERA,
        correctOrientation: true
    }


    Camera.getPicture(cameraOptions).then((imageData) => {
      // imageData is a base64 encoded string
        this.foto = "" + imageData;
    }, (err) => {
        console.log(err);
    });
  }

}
```

Añadiendo plataformas

Bien, hasta ahora hemos estado probando nuestra aplicación en el navegador, sin embargo no podemos probar el plugin camera desde el navegador. Ha llegado la hora de probar nuestra aplicación en un dispositivo móvil.

Lo primero que tenemos que hacer es añadir la plataforma en la que queremos probar nuestra aplicación.

Para añadir una plataforma utilizamos el siguiente comando desde consola, recordad que debemos esta siempre dentro de la ios
ionic platform add wp

Por lo tanto si queremos probar nuestra app en android escribiremos:

```
ionic platform add android
```

Esto nos creará una carpeta llamada platforms si no estaba creada, y añadirá una carpeta android con todo el código necesario para poder generar un archivo apk instalable.

Ejecutando nuestra app en el dispositivo móvil

Una vez tenemos añadida la plataforma si enchufamos nuestro móvil con un cable usb a nuestro pc podemos ejecutar la app directamente en el dispositivo con el siguiente comando:

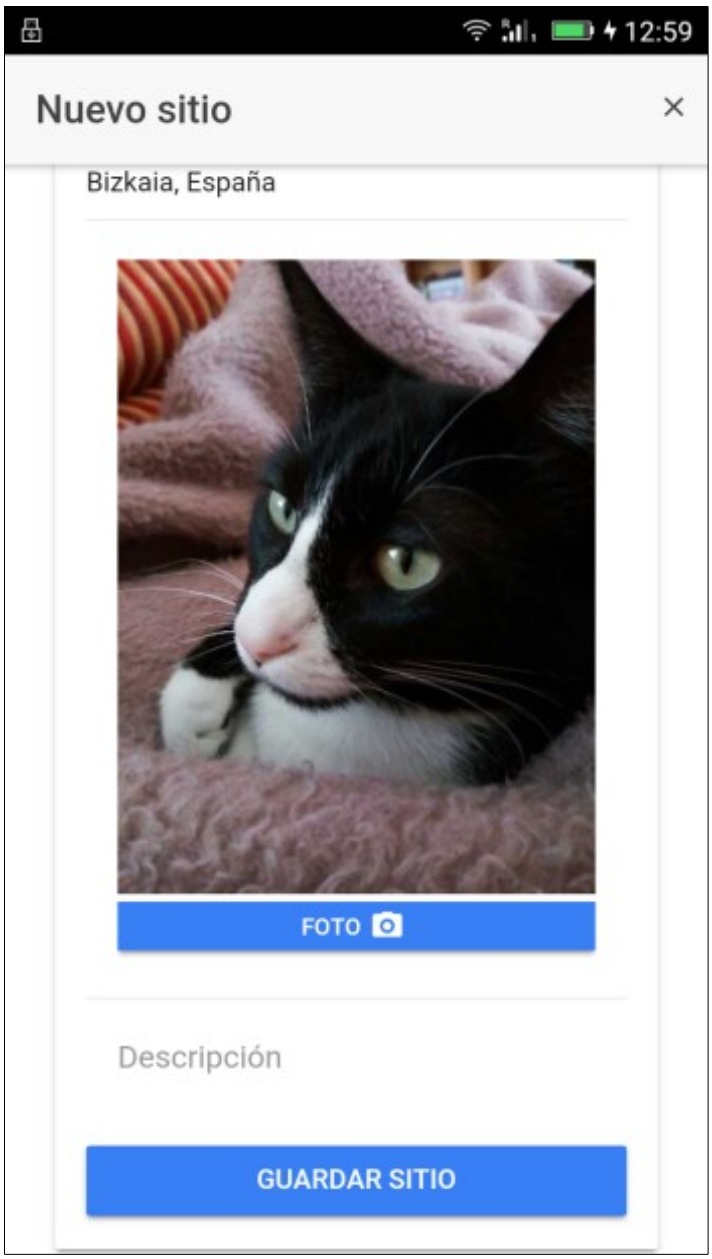
```
ionic run android
```

Si no disponéis de un dispositivo también podéis emular la aplicación utilizando ionic emulate:

```
ionic emulate android
```

En ios también puedes entrar en la carpeta platforms/ios y abrir el archivo con extensión .xcodeproj desde Xcode y ejecutarlo o emularlo desde Xcode.

El el móvil la app con el modal abierto se vería algo así:



Nuestra App funcionando en un dispositivo móvil.

Como podéis observar mi gato se ha prestado voluntariamente para posar en la foto ;-P

Dependiendo del sistema operativo que utilicéis en vuestro pc y del dispositivo puede que tengáis algún problema para que os reconozca el móvil, yo no tengo tiempo de investigar cada caso pero googleando seguro que dais con la solución, os animo a que dejéis en los comentarios si tenéis algún problema y que quién encuentre la solución lo ponga para ayudarnos unos a otros.

Por hoy lo dejamos aquí, en el siguiente capitulo veremos como guardar nuestros sitios en una base de datos local en nuestro dispositivo.

Un saludo y si aun no lo has hecho **subscribete a mi blog para no perderte los siguientes posts** 🙂

10 – Crear una aplicación para guardar nuestros sitios geolocalizados

Parte 5 – Guardando nuestros sitios en una base de datos local

Hola a todos:

En el post anterior obtuvimos la dirección a partir de las coordenadas, creamos el formulario para introducir la descripción del lugar y aprendimos a sacar fotografías con nuestro móvil, bien, todo esto nos vale de muy poco si cuando cerramos la aplicación perdemos toda esta información.

Para poder guardar nuestros sitios y que sigan allí para poder consultarlos siempre que queramos tenemos que almacenarlos en una base de datos local en el dispositivo.

Provider

Antes de nada vamos a introducir un concepto nuevo, para manejar los datos que introduciremos y extraeremos de la base de datos vamos a utilizar un **provider.**

Los **providers** son proveedores que se encargan del manejo de datos, bien extraídos de la base de datos, desde una API REST, etc.

Para crear un provider acudimos una vez más a ionic generator con el comando **ionic g provider.**

Para crear un proveedor para gestionar nuestra base de datos de sitios escribimos desde consola:

```
ionic g provider db
```

Esto nos creará un archivo typescript con el nombre del proveedor que hemos creado, en este caso **db.ts** dentro de la carpeta **providers**.

Por supuesto podéis dar el nombre que deseéis al provider que acabamos que crear, no tiene por que ser "db".

Vamos a echar un vistazo al código que se ha generado por defecto en **db.ts**:

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';
import 'rxjs/add/operator/map';

/*
  Generated class for the Db provider.

  See https://angular.io/docs/ts/latest/guide/dependency-injection.html
  for more info on providers and Angular 2 DI.
*/
@Injectable()
export class Db {

  constructor(public http: Http) {
    console.log('Hello Db Provider');
  }

}
```

Por defecto al crear un provider importa **Http** ya que los providers pueden ser utilizados para obtener datos des una petición a un servicio mediante Http, sin embargo como lo vamos a utilizar para gestionar nuestra base de datos no lo necesitamos y podemos eliminarlo, también debemos eliminarlo del constructor. Tampoco vamos a necesitar el "import 'rxjs/add/operator/map'", por lo que podemos eliminarlo también.

Para crear una base de datos vamos a utilizar **SQlite**, así que lo primero que necesitamos es instalar el plugin escribiendo desde consola:

```
ionic plugin add cordova-sqlite-storage
```

Recuerda que si está utilizando Linux o Mac necesitarás utilizar **sudo** por delante.

Una vez instalado el plugin ya podemos importarlo en nuestro provider desde **ionic-native**. Para poder utilizarlo inyectamos la dependencia **SQLite** en el constructor :

```
import { Injectable } from '@angular/core';
import { SQLite } from 'ionic-native';


/*
  Generated class for the Db provider.

  See https://angular.io/docs/ts/latest/guide/dependency-injection.html
  for more info on providers and Angular 2 DI.
*/
@Injectable()
export class Db {

  constructor(public db: SQLite) {

  }
}
```

Ahora vamos a crear en nuestro provider un método para abrir la base de datos:

```
public openDb(){
      return this.db.openDatabase({
          name: 'data.db',
          location: 'default' // el campo location es obligatorio
      });
  }
```

Como vemos utilizamos el método **openDatabase** al cual le pasamos como parámetro un objeto especificando el nombre y la localización.El campo location lo dejamos en '**default**'.

Ahora vamos a crear una tabla con los campos que vamos a necesitar para guardar nuestros sitios con los campos que necesitamos:

```
public createTableSitios(){
    return this.db.executeSql("create table if not exists sitios( id INTEGER PRIMARY KEY AUTOINCREMENT, lat FLOAT, lng FLOAT, address TEXT, description TEXT, foto TEXT )",{})
  }
```

Si ya has trabajado antes con otras bases de datos como por ejemplo **MySQL** no te costará trabajo entender como funciona **SQLite** ya que la sintaxis es muy similar aunque con algunas limitaciones.

En este caso nuestra tabla va a tener un campo **id** de tipo integer que sera la clave primaria, un campo **lat** de tipo float donde guardaremos la latitud de las coordenadas, un campo **lng** de tipo float donde guardaremos la longitud, un campo **address** de tipo textpara guardar la dirección, un campo llamado description de tipo text donde guardaremos la descripción del sitio y por último un campo **foto** también de tipo text donde guardaremos la foto en formato base 64.

Para continuar vamos a crear un método para guardar nuestros sitios:

```
public addSitio(sitio){
    let sql = "INSERT INTO sitios (lat, lng, address, description, foto) values (?,?,?,?,?)";
    return this.db.executeSql(sql, [sitio.lat,sitio.lng,sitio.address,sitio.description,sitio.foto]);
  }
```

En la sql definimos el insert con los campos de la tabla que vamos a introducir y en los valores ponemos interrogaciones '**?**', luego en el método execute pasamos como primer parámetro la sql y como segundo un array con los valores que corresponden a los campos donde van las interrogaciones, es decir hay que poner el valor de los campos en el mismo orden. El valor de los campos lo recibimos como parámetro en el objeto sitio. Cuando posteriormente llamemos a la función **addSitio** le tendremos que pasar un objeto con todos los campos.

Para poder utilizar el **provider** y **SQLite** debemos importar ambos en **app.module.ts** y declararlos en la sección **providers:**

```
import { NgModule, ErrorHandler } from '@angular/core';
import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';
import { MyApp } from './app.component';
import { MisTabsPage } from '../pages/mis-tabs/mis-tabs';
import { InicioPage } from '../pages/inicio/inicio';
import { ListadoPage } from '../pages/listado/listado';
import { InfoPage } from '../pages/info/info';
import { ModalNuevoSitioPage } from '../pages/modal-nuevo-sitio/modal-nuevo-
sitio';
import { Db } from '../providers/Db';
import { SQLite } from 'ionic-native';
@NgModule({
  declarations: [
    MyApp,
    MisTabsPage,
    InicioPage,
    ListadoPage,
    InfoPage,
    ModalNuevoSitioPage
  ],
  imports: [
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    MisTabsPage,
    InicioPage,
    ListadoPage,
    InfoPage,
    ModalNuevoSitioPage
  ],
```

```
  providers: [{provide: ErrorHandler, useClass:
IonicErrorHandler},Db,SQLite]
})
export class AppModule {}
```

Vamos a importar nuestro provider también en **app.component.ts** y vamos a abrir la base de datos y crear la tabla en **platform.ready** para asegurarnos de que el plugin **SQlite** ya se ha cargado antes de utilizarlo:

```
import { Component } from '@angular/core';
import { Platform } from 'ionic-angular';
import { StatusBar, Splashscreen } from 'ionic-native';
import { MisTabsPage } from '../pages/mis-tabs/mis-tabs';
import { Db } from '../providers/Db';



@Component({
  templateUrl: 'app.html'
})
export class MyApp {
  rootPage = MisTabsPage;

  constructor(platform: Platform, public db: Db) {

    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();
      this.db.openDb()
      .then(() => this.db.createTableSitios())
    });
  }
}
```

Por último importamos el provider en la página donde vamos a utilizarlo, en este caso en **modal-nuevo-sitio.ts**, también debemos inyectarlo en el constructor :

```
import { Component } from '@angular/core';
import { NavController, NavParams,  ViewController } from 'ionic-angular';
import { Camera } from 'ionic-native';
import { Db } from '../../providers/Db';



declare var google: any;

/*
  Generated class for the ModalNuevoSitio page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
*/
@Component({
  selector: 'page-modal-nuevo-sitio',
  templateUrl: 'modal-nuevo-sitio.html'
})
export class ModalNuevoSitioPage {

  coords : any = { lat: 0, lng: 0 }
  address: string;
  description: string = '';
  foto: any = '';

  constructor(public navCtrl: NavController, public navParams: NavParams,
private viewCtrl : ViewController, public db: Db) {


  }


...
```

Si recordamos el post anterior, en la vista **modal-nuevo-sitio.html**, en el evento **ngSubmit** del formulario llamábamos a la función **guardarSitio()**.
Vamos a definir esta función en el controlador del modal en el archivo **modal-nuevo-sitio.ts:**

```
guardarSitio(){
    let sitio = {
```

```
      lat: this.coords.lat,
      lng: this.coords.lng ,
      address: this.address,
      description: this.description,
      foto: this.foto
    }
    this.db.addSitio(sitio).then((res)=>{
      this.cerrarModal();
     /*  alert('se ha introducido correctamente en la bd'); */
    },(err)=>{ /* alert('error al meter en la bd'+err) */ })
}
```

Bien, como podemos observar en el código lo que hacemos es crear un objeto llamado **sitio** al que le asignamos los valores que tenemos recogidos.

Recuerda que el valor de la descripción se refleja automáticamente el la variable **this.description** que tenemos definida en el controlador al utilizar **[(ngModel)]**, las coordenadas y la dirección ya las teníamos recogidas y la foto se le asigna a **this.foto** en el momento de sacarla.

Después llamamos al método **addSitio** que hemos definido en nuestro provider pasándole como parámetro el objeto **sitio.**

Si todo ha ido bien cerramos el modal.

Bien, en este punto ya podemos guardar nuestros sitios en la base de datos, ahora nos toca poder sacar y mostrar un listado de los sitios que hemos guardado.

Para ello lo primero que vamos a hacer es crear un nuevo método en nuestro provider para obtener los sitios que tenemos guardados en la base de datos, por lo tanto editamos el archivo **db.ts** y añadimos el método **getSitios**:

```
import { Injectable } from '@angular/core';
import { SQLite } from 'ionic-native';


/*
  Generated class for the Db provider.

  See https://angular.io/docs/ts/latest/guide/dependency-injection.html
```

```
  for more info on providers and Angular 2 DI.
*/
@Injectable()
export class Db {

  constructor(public db: SQLite) {

  }

  openDb(){
      return this.db.openDatabase({
          name: 'data.db',
          location: 'default' // el campo location es obligatorio
      });
  }

  createTableSitios(){
    return this.db.executeSql('create table if not exists sitios( id INTEGER
PRIMARY KEY AUTOINCREMENT, lat FLOAT, lng FLOAT, address TEXT, description
TEXT, foto TEXT )',{})
  }

  addSitio(sitio){
    let sql = "INSERT INTO sitios (lat, lng, address, description, foto)
values (?,?,?,?,?)";
    return this.db.executeSql(sql,
[sitio.lat,sitio.lng,sitio.address,sitio.description,sitio.foto]);
  }

  getSitios(){
    let sql = "SELECT * FROM sitios";
    return this.db.executeSql(sql,{});
  }


}
```

Ha llegado el momento de mostrar nuestros sitios en la página listado, así que editamos el archivo **listado.ts** y vamos a importar el provider **Db** en el controlador de la página listado, debemos una vez más inyectar **Db** en el constructor, también vamos a crear una variable

miembro llamada **sitios** de tipo any que contendrá un array con todos los sitios que tenemos guardados:

```
import { Component } from '@angular/core';
import { NavController, NavParams } from 'ionic-angular';
import { Db } from '../../providers/Db';



/*
  Generated class for the Listado page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
*/
@Component({
  selector: 'page-listado',
  templateUrl: 'listado.html'
})
export class ListadoPage {

  sitios: any;

  constructor(public navCtrl: NavController, public navParams:
NavParams,public db : Db) {}


...
```

Ahora necesitamos extraer nuestros sitios de la base de datos cuando accedamos a la página del listado, podríamos hacerlo en el evento **ionViewDidLoad** que se ejecuta al cargar la página, el problema es que al contrario del modal, que se carga cada vez que lo llamamos, las paginas que se muestran al cambiar de pestaña se cargan una única vez al inicio, por lo que si guardamos nuevos sitios estos no se refrescaran a no ser que cierres la aplicación y la vuelvas a abrir.

Este es por lo tanto un buen momento para incorporar un nuevo concepto:

Ciclo de vida de una página

Cuando generamos una página con ionic generator vemos que nos crea por defecto el método **ionViewDidLoad**().

Como ya hemos comentado este método se ejecuta cuando la página se ha cargado:

```
ionViewDidLoad() {
    console.log('ionViewDidLoad ListadoPage');
 }
```

Vamos a ver los métodos con los que cuenta ionic en función de los eventos del ciclo de vida de una página:

- **ionViewDidLoad**:
  Se ejecuta cuando la página se ha terminado de cargar.
  Este evento sólo ocurre **una vez por página** cuando se está creando.

- **ionViewWillEnter**:
  Se ejecuta cuando la página está a punto de entrar y convertirse en la página activa.

- **ionViewDidEnter**:
  Se ejecuta cuando la página ha sido cargada y ahora es la página activa.

- **ionViewWillLeave**:
  Se ejecuta cuando se está a punto de salir de la página y ya no será la página activa.

- **ionViewDidLeave**:
  Se ejecuta cuando se ha salido de forma completa de la página
  y ya no es la página activa.

- **ionViewWillUnload**:
  Se ejecuta cuando la página está a punto de ser destruida,
  ella y todos sus elementos.

Bien, una vez sabido esto, vamos a cargar nuestros sitios en el evento **ionViewDidEnter**, es decir cada vez que entremos en la página, una vez se ha cargado y está activa.

Añadimos el método **ionViewDidEnter** al controlador de la página listado en el archivo **listado.ts** y hacemos la llamada para extraer nuestros sitios de la base de datos:

```
import { Component } from '@angular/core';
import { NavController, NavParams } from 'ionic-angular';
import { Db } from '../../providers/Db';



/*
  Generated class for the Listado page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
*/
@Component({
  selector: 'page-listado',
  templateUrl: 'listado.html'
})
export class ListadoPage {

  sitios: any;

  constructor(public navCtrl: NavController, public navParams: NavParams,
public db : Db) {}

  ionViewDidLoad() {
    console.log('ionViewDidLoad ListadoPage');
  }

  ionViewDidEnter(){
     this.db.getSitios().then((res)=>{
     this.sitios = [];
```

```
    for(var i = 0; i < res.rows.length; i++){
        this.sitios.push({
          lat: res.rows.item(i).lat,
          lng: res.rows.item(i).lng,
          address: res.rows.item(i).address,
          description: res.rows.item(i).description,
          foto: res.rows.item(i).foto
        });
    }

    },(err)=>{ /* alert('error al sacar de la bd'+err) */ })
  }

}
```

Al llamar a **this.db.getSitios** obtenemos una promesa donde en **res** obtenemos el recurso devuelto por la base de datos. No podemos acceder directamente a los registros extraidos de la base de datos, debemos llamar a **res.rows.item**, y pasarle entre paréntesis el indice del elemento que queremos obtener, para ello recorremos **res.rows** mediante un bucle **for**y añadimos cada item al array **this.sitios**.

Al finalizar el bucle **this.sitios** contendrá un array con todos los sitios que hemos guardados en la base de datos. Ahora solo nos queda mostrarlos en el listado.

Vamos a editar la vista de la página listado, para ello abrimos el archivo **listado.html**, borramos el texto provisional que teníamos y lo dejamos de la siguiente manera:

```
<!--
  Generated template for the Listado page.

  See http://ionicframework.com/docs/v2/components/#navigation for more info
on
  Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Listado</ion-title>
  </ion-navbar>
```

```
</ion-header>

<ion-content padding>
<ion-list>
  <ion-item *ngFor="let sitio of sitios">
    <ion-thumbnail item-left>
      <img [src]="sitio.foto">
    </ion-thumbnail>
    <h2>{{ sitio.address }}</h2>
    <p>{{ sitio.description }}</p>
  </ion-item>
</ion-list>
</ion-content>
```

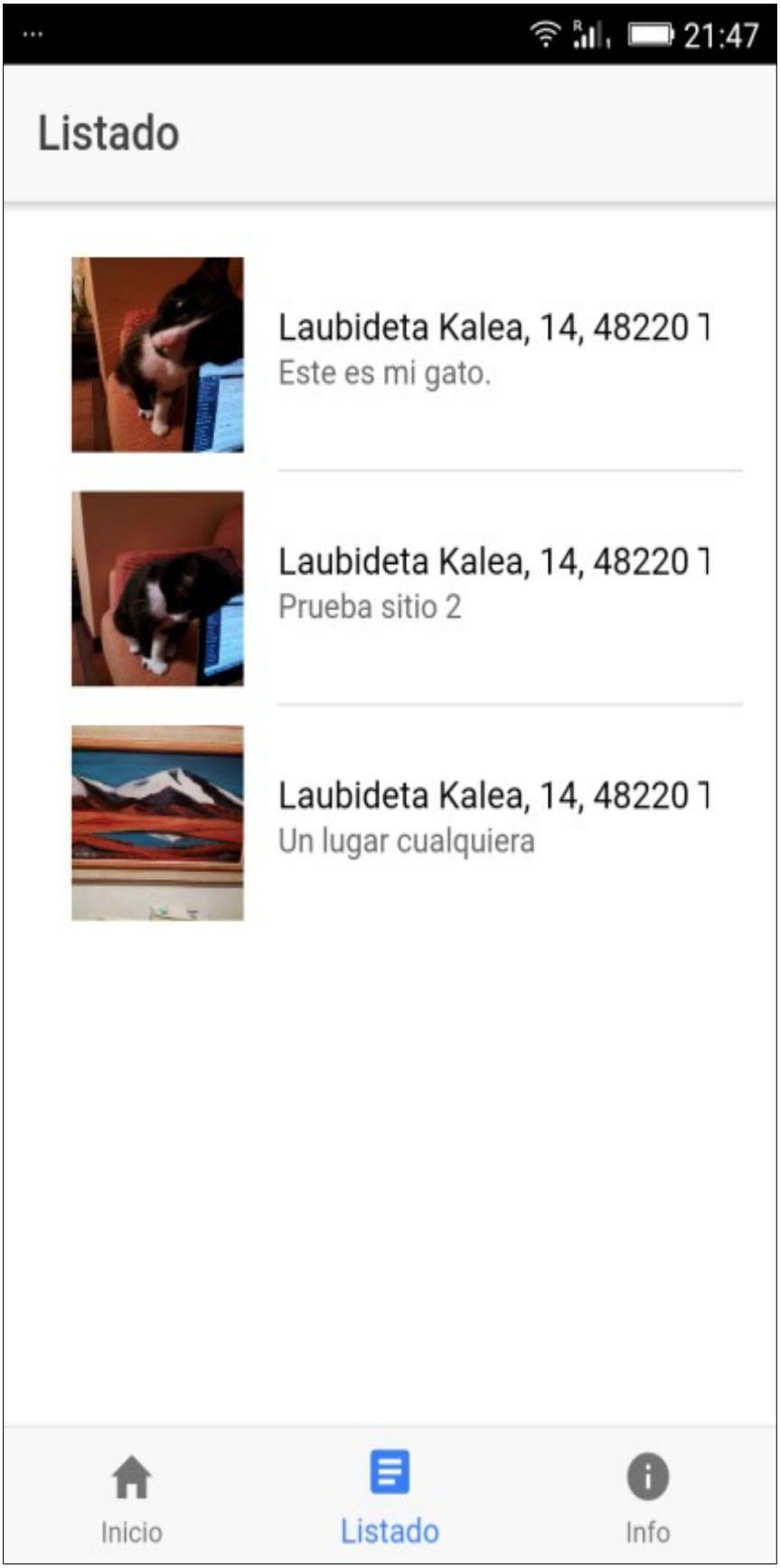
Como vemos dentro de **ion-content** hemos creado un elemento **ion-list**, después con ***ngFor** recorremos nuestro array de sitios creando un elemento **ion-item** por cada iteración.

Con **ion-thumbnail** mostramos una miniatura de la foto del sitio, despues mostramos la dirección y la descripción.

Como veis no tiene mayor dificultad. Si queréis saber más sobre el componente **ion-list** y sus posibilidades podçeis consultar la documentación oficial siguiendo este enlace:

https://ionicframework.com/docs/components/#lists

Si todo ha ido bien la pantalla listado debería tener un aspecto similar a este, bueno… tal vez en tus fotos no salga un gato ;-P


21:47
Listado
Laubideta Kalea, 14, 48220 1
Este es mi gato.
Laubideta Kalea, 14, 48220 1
Prueba sitio 2
Laubideta Kalea, 14, 48220 1
Un lugar cualquiera
Inicio
Listado
Info

Esto es todo de momento, hoy hemos aprendido como guardar datos en una base de datos local con SQLite, hemos aprendido a crear un provider para utilizarlo como servicio para gestionar los accesos a la base de datos y hemos aprendido como funcionan los ciclos de vida de un página.

En el siguiente post seguiremos avanzando con nuestra app. Hay muchas cosas que se pueden mejorar, aquí solo pretendo explicar los conceptos básicos para que podáis crear vuestras propias aplicaciones. Os animo a que experimentéis y tratéis de mejorar la app, es sin duda la mejor forma de aprender. Podéis compartir vuestra experiencias en los comentarios y así ayudar a los que tengan algún problema.

Un saludo y si aun no lo has hecho **subscribete a mi blog para no perderte los siguientes posts** 🙂