### Qué es Ionic

La primera versión de Ionic se publica en 2013. Y ¿Qué es Ionic? Ionic es un framework que une AngularJS como herramienta de desarrollo frontend con Apache Cordova (PhoneGab) para el uso de funcionalidades nativas en dispositivos móviles, creando una revolución en el mundo de las aplicaciones híbridas: aplicaciones que emplean tecnología web y en empaquetan en un webviewer con una capa que permite acceso a determinadas funcionalidades nativas.



La historia de Ionic ha ido ligada a la de Angular. Al igual que éste, Ionic sufrió una revolución con la salida de Angular 2 y también sufrió su comunidad, incluso más, estos drásticos cambios. Con Angular2 llegó Typescript y, a partir de aquí, toda esta carrera que intenta este documento sintetizar.

Con Ionic3 se intenta dar solidez al cambio, mejorando Ionic Cli e Ionic Native.

Pero sin duda, con Ionic 4 se ha establecido el rumbo que, por ahora, parece más representativo del framework: la posibilidad de desarrollo de PWA, la posibilidad de emplear como framework de desarrollo VUE, Angular o React y la incorporación de Capacitor como wrapper nativo en lugar de Cordova, aunque aún puede seguir empleándose.

Da la sensación de que el equipo de Ionic intenta comenzar a hacer más fácil el camino a los desarrolladores que quieran apostar por él y enmendar todos esos vaivenes del pasado que tantos quebraderos de cabeza supusieron.

Por otra parte, existen en el desarrollo multiplataforma de aplicaciones nativas existen muchas más opciones actualmente: React Native o Flutter, entre otros.



No obstante, la filosofía es diferente. Por ejemplo, React Nativa busca emplear como base React para desarrollar aplicaciones nativas, es decir, algunas modificaciones serán necesarias entre Android e IOS. Mientras, Ionic tiene el objetivo de compartir el mismo código basado en desarrollo web en PWA y aplicaciones híbridas.

En cuanto a rendimiento, hoy en día se puede afirmar que para el 90% de las aplicaciones no habrá diferencia entre una aplicación nativa y una híbrida, dadas las especificaciones de los nuevos terminales, no obstante, cabe decir que las aplicaciones nativas siempre sacan el mejor partido del hardware. Es más, el acceso a los recursos será mucho más completo y óptimo en el desarrollo nativo.

Ionic aporta una capa de personalización de componentes que, salvando las diferencias, no permiten aportar una apariencia nativa. En React no es tan completa, pero Flutter o los propios entornos nativos de desarrollo mejoran radicalmente este aspecto respecto a Ionic.

En cuanto al número de aplicaciones móviles desarrolladas con los diferentes entornos, claramente el desarrollo nativo gana por goleada, aunque está ganando terreno React Native paulatinamente. Ionic se mantiene en una zona donde, evidentemente no es el rey de las aplicaciones nativas, pero abre las puertas al desarrollo híbrido y al web con las PWA.

Sea cual sea la elección, tener conocimientos de Ionic nunca podría ser una mala idea. Veamos su funcionamiento.

### Instalación y configuración

Desde su web podemos seguir los pasos según nuestra plataforma para completar la instalación y configuración del entorno. En este ejemplo vamos a emplear Angular como framework de desarrollo, aunque tal como se ha comentado, actualmente se soportan tanto VUE como ReactJS.

Antes de seguir con los siguientes pasos se debe tener configurado NodeJs, Angular (en nuestro caso) y el SDK nativo que optemos, tanto Android como IOS. En este ejemplo, optamos tan solo por Android y es necesario para que se puedan realizar el wrapper con las funcionalidades nativas y la compilación de la app.

* Para Android podemos seguir los siguientes pasos descritos en la web de Cordova: <https://cordova.apache.org/docs/en/8.x/guide/platforms/android/> que incluyen:
* Instalación de Java Development Kit (JDK) y configuración de las variables de entorno: <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
* Instalación de Gradle: <https://gradle.org/install/>
* Instalación de Android SDK y configuración de las variables de entorno: <https://developer.android.com/studio/index.html>

A partir de aquí podemos comenzar a instalar el cliente de Ionic:

npm install -g ionic

Y desde este momento emplear la línea de comandos para crear nuestro primer proyecto:

|  |  |  |
| --- | --- | --- |
| ionic start myApp blank | ionic start myApp tabs | ionic start myApp sidemenu |
|  |  |  |

Podemos emplear en la creación comandos que nos permiten crear un proyecto basado en plantillas de pestañas o menú lateral. No obstante, en cualquier momento se pueden añadir estas funcionalidades, incluso mezclar. En nuestro caso comenzaremos empleando la plantilla de pestañas, por lo que ejecutamos:

ionic start PicReporter tabs

Importante señalar que con el comando anterior se emplear Angular por defecto como framework de desarrollo y Typescript como lenguaje de programación.

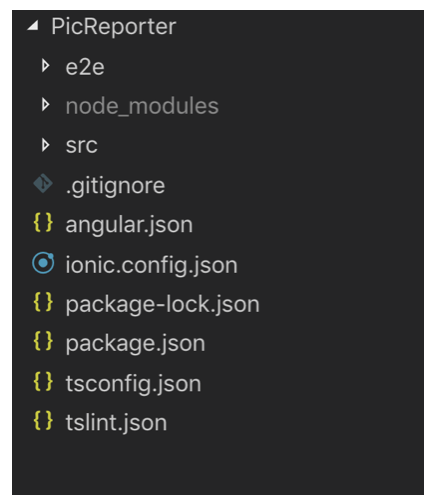
Para ver el resultado, ejecutamos

ionic serve

Nuestra aplicación estará disponible en <http://localhost:8100>.

### Principios de Ionic

Analizamos la estructura de archivos generada en el proyecto:



La estructura base es similar a un proyecto de Angular com la diferencia del archivo ionic.config.json:

{

"name": "PicReporter",

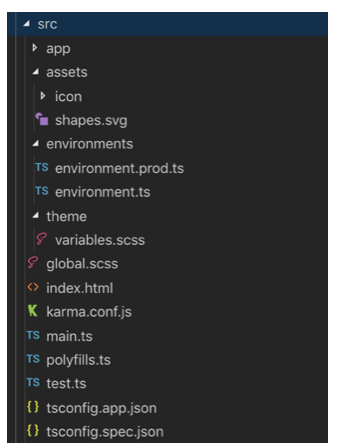
"integrations": {},

"type": "angular"

}

Que, por ahora, indica el nombre del proyecto el tipo de framework empleado.

La carpeta src es ahora algo más compleja, podemos destacar los siguientes archivos:



* index.html: es la página principal de la SPA (Single-Page-Application). Es similar al producido por Angular:

<body>

<app-root></app-root>

</body>

Como se observa, existe un componente de entrada denominado app-root que es el que contiene a los demás componentes.

* global.scss

// http://ionicframework.com/docs/theming/

@import '~@ionic/angular/css/core.css';

@import '~@ionic/angular/css/normalize.css';

@import '~@ionic/angular/css/structure.css';

@import '~@ionic/angular/css/typography.css';

@import '~@ionic/angular/css/display.css';

@import '~@ionic/angular/css/padding.css';

@import '~@ionic/angular/css/float-elements.css';

@import '~@ionic/angular/css/text-alignment.css';

@import '~@ionic/angular/css/text-transformation.css';

@import ‘~@ionic/angular/css/flex-utils.css';

Importa a todas las hojas de estilo que incluye ionic para que sus componentes tengan la apariencia nativa. Como se observa, su extensión no es css, sino scss. Esto es debido a que Ionic emplea por defecto Sass: es un framework para preprocesas hojas de estilo, es decir, nos permite incluir variables y funciones de forma intuitiva que luego al ser transitadas generan complejas hojas de estilo. Existen algunas diferencias que iremos viendo a lo largo del ejemplo sin entrar en profundidad. Para más información: <https://sass-lang.com/guide>



* main.ts: es el archivo Typescript punto de entrada para la aplicación.

platformBrowserDynamic().bootstrapModule(AppModule)

.catch(err => console.log(err));

Emplea como módulo de entrada el denominado, por defecto, AppModule que analizaremos a continuación.

Además, contiene las carpetas:

* enviroment y assets, con el mismo objetivo que en Angular.
* themes que incluye el archivo variables.scss que define las variables que emplean las hojas de estilo por defecto y que puede ser empleado para personalizar nuestro look&feel incluso en tiempo real. Muy útil para aquellas aplicaciones que tienen modo día y modo noche.

A continuación, debemos analizar la carpeta app que será donde principalmente operemos. En la raíz se encuentran:

* app.scss: para definir hojas de estilos a nivel del módulo principal. Es decir, para toda la aplicación. Aquí debemos incluir aquellas reglas que son globales.
* app.module.ts: es el módulo que define al componente app y que, tal como vimos en main.ts es el módulo que se carga al inicio.

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { RouteReuseStrategy } from '@angular/router';

import { IonicModule, IonicRouteStrategy } from '@ionic/angular';

import { SplashScreen } from '@ionic-native/splash-screen/ngx';

import { StatusBar } from '@ionic-native/status-bar/ngx';

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

@NgModule({

declarations: [AppComponent],

entryComponents: [],

imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule],

providers: [

StatusBar,

SplashScreen,

{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy }

],

bootstrap: [AppComponent]

})

export class AppModule {}

Carga al componente AppComponent que veremos a continuación y algunos servicios como son la barra de estado y la pantalla de inicio de carga (splashScreen). Además, otros módulos como son el de enrutamiento definido por la variable AppRoutingModule que lo importa del archivo app-routing.module que analizamos posteriormente.

* app.component.ts: es el código del módulo principal, antes descrito.

@Component({

selector: 'app-root',

templateUrl: 'app.component.html'

})

export class AppComponent {

constructor(

private platform: Platform,

private splashScreen: SplashScreen,

private statusBar: StatusBar

) {

this.initializeApp();

}

initializeApp() {

this.platform.ready().then(() => {

this.statusBar.styleDefault();

this.splashScreen.hide();

});

}

}

Como era de prever define su selector como app-root (ver index.html) y su archivo HTML que veremos a continuación. Su código básicamente comprueba cuando está la aplicación cargada para mostrar la barra de estado y ocultar la pantalla de carga.

* app.component.html: es el archivo HTML del módulo principal.

<ion-app>

<ion-router-outlet></ion-router-outlet>

</ion-app>

Emplea la etiqueta de maquetación ion-app e inserta ion-router-outlet, similar a la etiqueta de enrutamiento que empleamos en el ejemplo de Angular. Esto quiere decir que en función de la ruta el módulo de enrutamiento inyectará uno u otro componente en <ion-router-outlet>. Por otra parte, durante el ejemplo iremos viendo las diferentes etiquetas de maquetación de Ionic.

* app-routing.module.ts: define el enrutamiento para las pestañas (tabs) por defecto y es el importado en el módulo principal app.module.ts.

import { NgModule } from '@angular/core';

import { PreloadAllModules, RouterModule, Routes } from '@angular/router';

const routes: Routes = [

{ path: '', loadChildren: './tabs/tabs.module#TabsPageModule' }

];

@NgModule({

imports: [

RouterModule.forRoot(routes, { preloadingStrategy: PreloadAllModules })

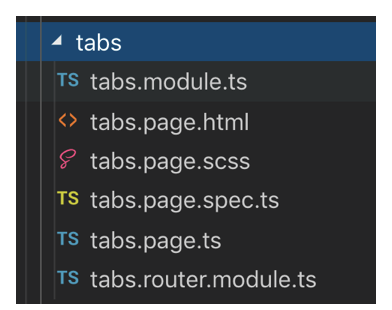
],

exports: [RouterModule]

})

export class AppRoutingModule {}

En este archivo, en realidad se realiza una delegación de la ruta por defecto ‘’ al archivo ./tabs/tabs.module, concretamente a la clase TabsPageModule, por tanto debemos analizar la carpeta tabs.



Tabs es un componente que ha creado el cliente de Ionic automáticamente cuando le indicamos que queríamos una aplicación por pestañas. Por tanto, comencemos por el archivo de módulo:

* tabs.module.ts: incluyen a los módulos básicos, más el de formulario y el de enrutamiento que está definido en el archivo tabs.router.module.ts

import { IonicModule } from '@ionic/angular';

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { TabsPageRoutingModule } from './tabs.router.module';

import { TabsPage } from './tabs.page';

@NgModule({

imports: [

IonicModule,

CommonModule,

FormsModule,

TabsPageRoutingModule

],

declarations: [TabsPage]

})

export class TabsPageModule {}

Además, declara el componente TabsPage cuyo contenido veremos:

declarations: [TabsPage]

* tabs.page.ts:

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

@Component({

selector: 'app-tabs',

templateUrl: 'tabs.page.html',

styleUrls: ['tabs.page.scss']

})

export class TabsPage {

constructor() {}

}

Como cualquier componente, define su selector, su HTML y su SCSS.

* tabs.page.html:

<ion-tabs>

<ion-tab-bar slot="bottom">

<ion-tab-button tab="tab1">

<ion-icon name="flash"></ion-icon>

<ion-label>Tab One</ion-label>

</ion-tab-button>

<ion-tab-button tab="tab2">

<ion-icon name="apps"></ion-icon>

<ion-label>Tab Two</ion-label>

</ion-tab-button>

<ion-tab-button tab="tab3">

<ion-icon name="send"></ion-icon>

<ion-label>Tab Three</ion-label>

</ion-tab-button>

</ion-tab-bar>

</ion-tabs>

Emplea la etiqueta de maquetación ion-tabs para generar las pestañas que serán visualizadas en la parte inferior (<ion-tab-bar slot=“bottom”>) de la aplicación con los tres botones, sus etiquetas e iconos.

* tabs.router.module.ts: el archivo de configuración del enrutamiento del módulo tabs.

import { NgModule } from "@angular/core";

import { RouterModule, Routes } from "@angular/router";

import { TabsPage } from "./tabs.page";

const routes: Routes = [

{

path: "tabs",

component: TabsPage,

children: [

{

path: "tab1",

children: [

{

path: "",

loadChildren: "../tab1/tab1.module#Tab1PageModule"

}

]

},

{

path: "tab2",

children: [

{

path: "",

loadChildren: "../tab2/tab2.module#Tab2PageModule"

}

]

},

{

path: "tab3",

children: [

{

path: "",

loadChildren: "../tab3/tab3.module#Tab3PageModule"

}

]

},

{

path: "",

redirectTo: "/tabs/tab1",

pathMatch: "full"

}

]

},

{

path: "",

redirectTo: "/tabs/tab1",

pathMatch: "full"

}

];

@NgModule({

imports: [RouterModule.forChild(routes)],

exports: [RouterModule]

})

export class TabsPageRoutingModule {}

Vemos cómo define como ruta global tabs, y como subruta tab1, tab2 y tab3 que delega la carga de módulos a las diferentes tabs. Por defecto, siempre carga la tab1.

Veamos una diferencia importante entre app-routing.module.ts definía:

RouterModule.forRoot(routes, { preloadingStrategy: PreloadAllModules })

Es decir, las rutas definidas por routes (que simplemente era la por defecto delegarla a tabs.router.module.ts) y la opción de cargar cuanto antes todos los módulos para toda la aplicación (forRoot). Esta última opción es muy interesante y luego la analizaremos.

Sin embargo, tabs.router.module.ts define:

RouterModule.forChild(routes)

Las rutas definidas por la variable routes como válidas para este módulo y sus hijos y no a nivel de toda la aplicación (forChild).

También es interesante señalar cómo se han definido las rutas en este archivo:

path: 'tabs',

component: TabsPage,

children: [

{

path: 'tab1',

children: [

{

path: '',

loadChildren: '../tab1/tab1.module#Tab1PageModule'

}

]

},

…

Define la ruta raíz como tabs para el componente actual y diseña una lista de hijos (carga en cascada) en donde tenemos las diferentes pestañas que a su vez, cada una, genera una lista de hijas (para subrutas), aunque en nuestro caso solo se definen las rutas ‘’.

Realmente, no es muy intuitiva la forma de estructurar este archivo la primera vez que lo empleas, pero una vez entendido, es la forma más modular y escalable posible para definir rutas y subrutas.

De hecho, como se ve, se define la ruta por defecto y se delega en el módulo de cada pestaña:

loadChildren: ‘../tab1/tab1.module#Tab1PageModule'

Efectivamente, el cliente de Ionic ha creado cada componente de pestaña como un módulo para que podamos controlar subcomponentes dentro de la misma pestaña si quisiéramos compartiendo todos ellos el mismo archivo module.

Si nos fijamos en el archivo tab1.module.ts:

import { IonicModule } from '@ionic/angular';

import { RouterModule } from '@angular/router';

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { Tab1Page } from './tab1.page';

@NgModule({

imports: [

IonicModule,

CommonModule,

FormsModule,

RouterModule.forChild([{ path: '', component: Tab1Page }])

],

declarations: [Tab1Page]

})

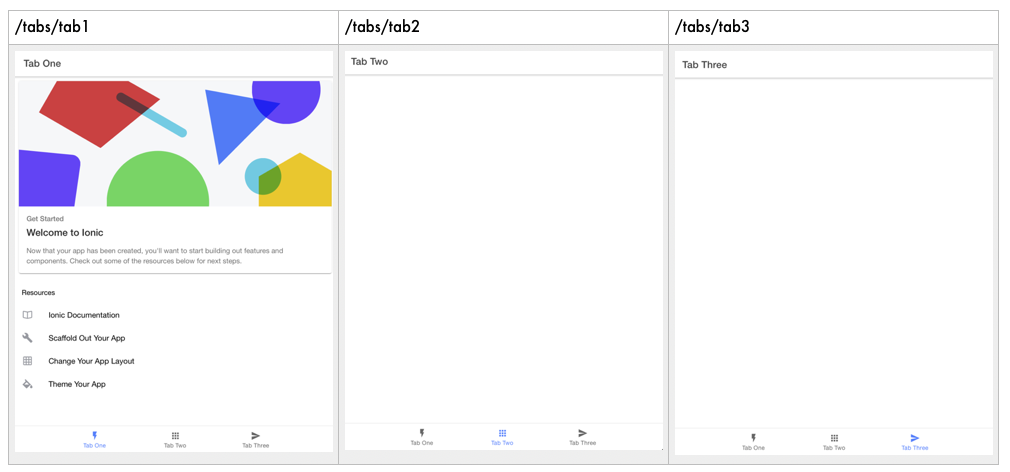
export class Tab1PageModule {}

También se definen rutas por defecto para cargar al, por ahora, único componente de este módulo. Es así, como decimos, por si queremos hacer subpestañas o carga de módulos internos dentro de una misma pestaña.

En lugar, de analizar los diferentes archivos de las pestañas, vamos a verlas en funcionamiento, para ello ejecutamos (dentro de la carpeta de proyecto):

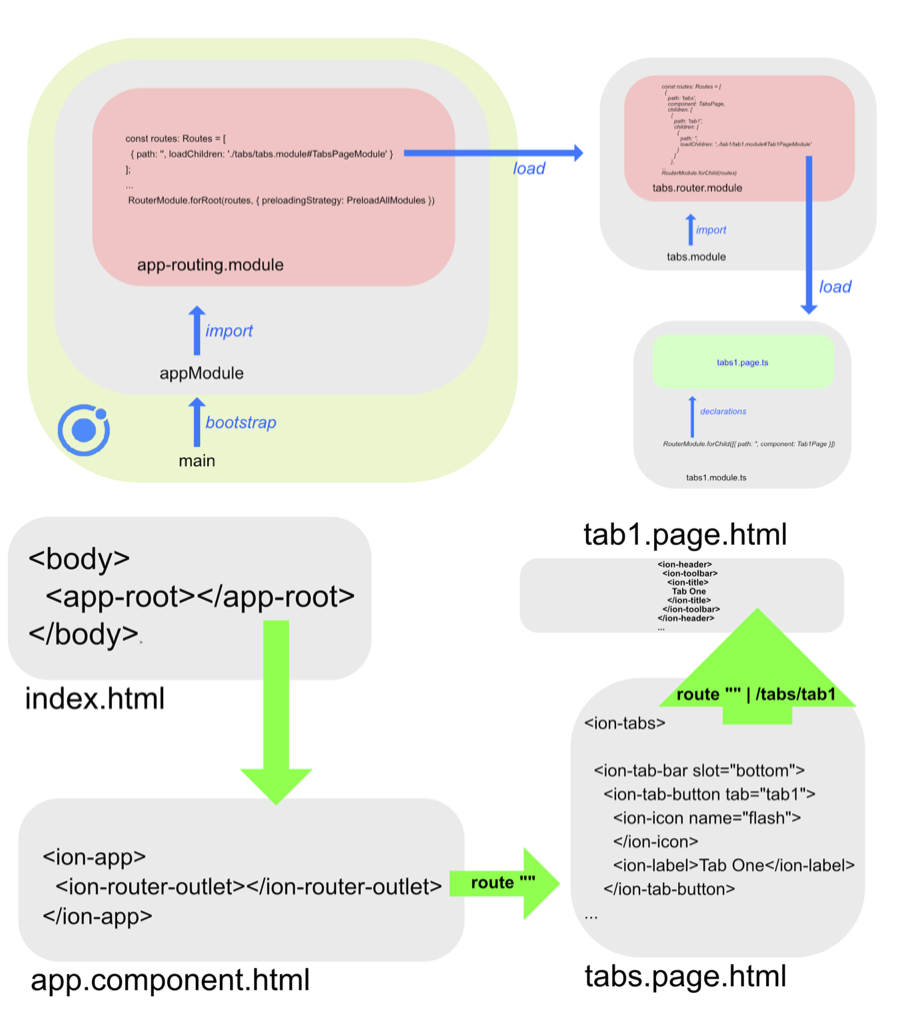
ionic serve

Automáticamente, se levanta un servidor de pruebas en la ruta <http://localhost:8100>:



Esta es, sin duda, una de las ventajas de Ionic. Hasta el momento tenemos desarrollo puro web, por lo tanto, todo lo podemos probar en modo web. Más adelante, cuando empleemos recursos nativos deberemos tomar otro tipo de soluciones.

Angular no posee DOM virtual, pero su rendimiento se mantiene competitivo gracias a su modularidad y carga bajo demanda. Intentemos representar la carga que se realiza en la primera pestaña que es la que se realiza por defecto:

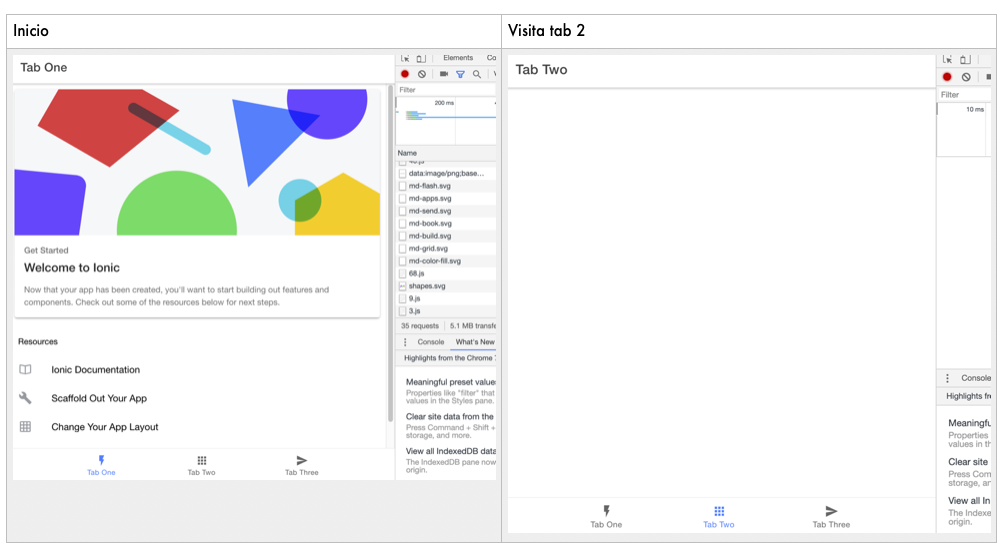


De este modo en la vista se ha ido inyectando los siguientes elementos y sus dependencias de imágenes y css, entre otros.

Antes de comenzar con el ejemplo, veamos por último cómo se realiza la carga de los módulos. Es importante recordar la línea del archivo app-routing.module.ts:

RouterModule.forRoot(routes, { preloadingStrategy: PreloadAllModules })

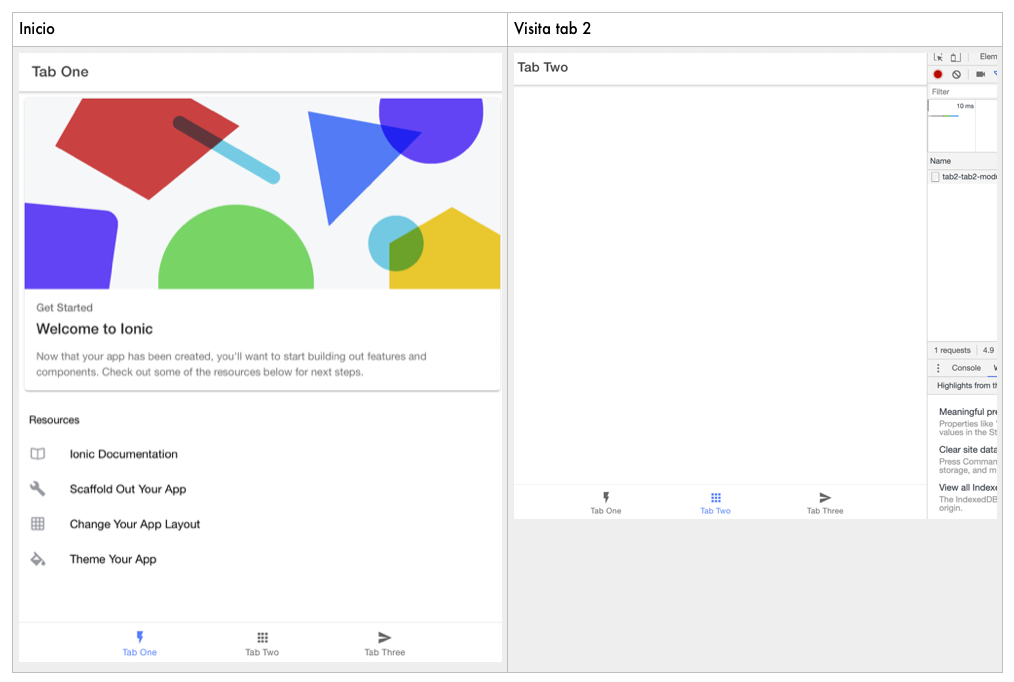
Esta línea indica que cuando se carga la aplicación la primera vez (ruta “ “) se cargarán todos los módulos. Analizando la carga en Google Chrome obtenemos:



Se observa que ya no hay carga de módulos (archivos js) cuando cambiamos de pestaña. Sin embargo, si no existiera el parámetro

{ preloadingStrategy: PreloadAllModules }

O bien cargáramos la aplicación directamente desde otra url inicia el comportamiento sería bien distinto, cargando cada JS bajo demanda:



La estrategia depende de muchos parámetros, en función de si buscamos fluidez o rapidez de carga inicial. Además, también influye el tipo de aplicación que desarrollamos y la carga que supone cada módulo y componente al total. Nosotros, debido a que no tenemos componentes de alta carga y deseamos en la aplicación híbrida la máxima fluidez posible una vez cargada dejaremos la opción tal como estaba por defecto.

Para más información: <https://angular.io/guide/lazy-loading-ngmodules>.

### Capacitor

Antes de nada, diremos que vamos a emplear Capacitor como framework que realiza el puente entre nuestro código “web” y los recursos nativos del sistema. Es el heredero de Cordova, aunque en el momento que se desarrolla este documento aún está en proceso de mejora. Toda la documentación la encontramos en <https://capacitor.ionicframework.com/docs/getting-started/with-ionic/> .



Para hacer que nuestra app tenga habilitado capacitor, ejecutamos:

ionic integrations enable capacitor

npx cap init

Ahora ejecutamos:

ionic build

Para tener nuestra carpeta www generada. Aquí se crea la PWA.

Nuestro proyecto accederá a diferentes recursos nativos, algunos mediante Capacitor y otros mediante Cordova. Vamos a instalar algunos de ellos.

La cámara la emplearemos en la tab2 y no requiere instalación extra de plugins, pero si modificar el archivo /android/app/src/main/AndroidManifest.xml, añadiendo, si ya no están añadidas, las líneas:

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"/>

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

Algunos plugins de Capacitor emplean código relacionado con funcionalidades web, como es el caso de la cámara por lo que se hace necesario instalar esta librería:

npm install @ionic/pwa-elements

Y posteriormente, ejecutamos:

npx cap copy

Copiamos todos los assets de la carpeta built www. Aunque en estos momentos no sea estrictamente necesario, más adelante lo necesitaremos.

Para más información sobre el workflow de Capacitor: <https://capacitor.ionicframework.com/docs/basics/workflow/>

Para esta versión, además se ha requerido algunas modificaciones en el archivo /src/main.ts:

import { defineCustomElements } from '@ionic/pwa-elements/loader';

if (environment.production) {

enableProdMode();

}

platformBrowserDynamic().bootstrapModule(AppModule)

.catch(err => console.log(err));

// for camera capacitor on PWA

defineCustomElements(window);

Para la geolocalización, empleada en tab2 y tab3, emplearemos el plugin de Capacitor: <https://capacitor.ionicframework.com/docs/apis/geolocation/>

Tan solo, necesitamos comprobar que en el archivo AndroidManifest.xml se encuentren las líneas:

<!-- Geolocation API -->

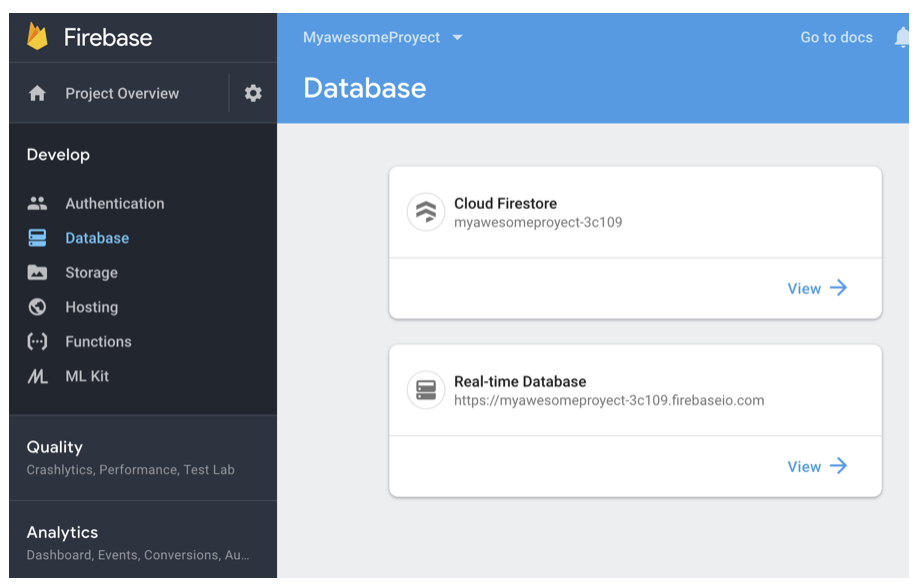
<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<uses-feature android:name="android.hardware.location.gps" />

### Desarrollo de la aplicación

Comenzamos instalando las herramientas de firebase, en esta ocasión emplearemos la base de datos Cloud Firebase, el sistema de almacenamiento Cloud Firestore y el servicio de hosting.



npm install -g firebase-tools

npm install --save firebase @angular/fire

#### Código fuente

Continuamos, creando las interfaces que definen nuestro modelo:

Para almacenar al usuario del sistema, model/user.ts:

export interface User {

email: string;

}

Creamos el contenido del archivo environment.s (y environment.production.ts ,en nuestro caso):

export const environment = {

production: true,

firebaseConfig : {

apiKey: "AIzaSyCAn38A9hzvxxxxxxxxxxxxxxxxxxx",

authDomain: "myawesomeproyect-3c109.firebaseapp.com",

databaseURL: "https://myawesomeproyect-3c109.firebaseio.com",

projectId: "myawesomeproyect-3c109",

storageBucket: "myawesomeproyect-3c109.appspot.com",

messagingSenderId: "998713355227",

appId: "1:998713355227:web:ea553417c5e12f0c"

},

entriesCollection: 'Entry', // collection for entries

};

En este caso es muy importante el campo storageBucket para emplear el sistema de almacenamiento. Por otra parte, la variable entriesCollection, define el nombre que vamos a emplear para la colección de entradas de esta aplicación. Como ya hemos visto, es un sistema de base de datos orientado a documentos. Cada documento, en nuestro caso es una entrada, debe estar ordenado y almacenado dentro de una colección. Es la jerarquía que se emplea en este tipo de bases de datos.

Para almacenar las entradas en nuestro sistema, model/entry.ts:

export interface Entry {

key?: string;

owner: string;

url: string;

image?: any; // see tab2 and cloud service

title: string;

description: string;

lat: number;

long: number;

date: Date;

}

Pasamos ahora a los servicios que emplea nuestra aplicación, en concreto 4.

BackButton que gestiona el botón de atrás en las aplicaciones y en las PWA.

ionic generate service services/backbutton --skipTest

import { Injectable } from '@angular/core';

import { Platform, NavController } from '@ionic/angular';

import { Router, NavigationEnd } from '@angular/router';

import { Plugins, AppState } from '@capacitor/core';

const { App } = Plugins;

@Injectable({

providedIn: 'root'

})

export class BackbuttonService {

currentURL: any = '';

openModal = false;

modalOpened: any;

/\* This service manages backbutton \*/

constructor(

private platform: Platform,

private navCtrl: NavController,

private router: Router

) {}

backButton() {

/\*Cordova solution

this.platform.backButton.subscribe(()=>{

console.log("back button from old scholl")

})\*/

// PWA avoid user to exit with first back button

window.addEventListener('load', () => {

window.history.pushState({ noBackExitsApp: true }, '');

});

// PWA backbutton solution: close modal

window.addEventListener('popstate', (s) => {

if (this.openModal) {

this.modalOpened.closeMe(); //see modal components

this.openModal = false;

} else {

this.router.navigate(['']);

}

});

// App

App.addListener('backButton', (data: any) => {

console.log('back button pressed');

if (!this.openModal) { // any modal is opened

if ( // we are at home

this.currentURL === '/' ||

this.currentURL === '' ||

this.currentURL === '/tabs' ||

this.currentURL === '/tabs/tab1'

) {

App.exitApp();

} else { // we are in any other tab

this.router.navigate(['']);

}

} else { // a modal is open, close it

this.openModal = false; // this solution doesn't work in modals nested. Our app has just a modal at a time

if (this.modalOpened && this.modalOpened.closeMe) {

this.modalOpened.closeMe();

}

}

});

// This code allows us to know current url.

// When app navigates, the current url takes some time to be updated: observable

this.router.events.subscribe(event => {

if (event instanceof NavigationEnd) {

// this is executed when navigation ends

this.currentURL = event.url;

}

});

}

}

Es interesante ver cómo se ha solucionado el comportamiento de cerrar un modal cuando se pulsa el botón de atrás en una PWA.

Commons que proporciona algunas utilidades comunes como son mostrar la pantalla de carga y ocultarla. Aquí podríamos haber aglutinado más utilidades que se emplean en diferentes componentes.

ionic generate service services/commons --skipTest

import { Injectable } from '@angular/core';

import { LoadingController } from '@ionic/angular';

@Injectable({

providedIn: 'root'

})

export class CommonsService {

loadingC = null;

/\*\* Open a loading modal or any other modal window take some time.

\* Sometimes, app can call to show modal and, then close modal before the modal is opened.

\* To prevent keeping modal opened for this reason, we have this flag

\*/

pendingToClose: boolean;

constructor(public loadingController: LoadingController) {}

// public methods

public showLoading(msg?) {

this.pendingToClose = false;

// show the loading modal, see @presentLoading

this.presentLoading(msg);

}

public hideLoading() {

if (this.loadingC != null) { // a loading is opened

this.loadingC.dismiss();

this.loadingC = null;

this.pendingToClose = false;

} else { // any loading is opened, nothing to close

this.pendingToClose = true;

}

}

private async presentLoading(msg?) {

// if a loading is opened, any other loading must be opened

if (this.loadingC) { this.loadingC.dismiss(); }

// Open a native loading modal

this.loadingC = await this.loadingController.create({

message: msg ? msg : 'Wait, please'

});

await this.loadingC.present();

// Now it is opened. If some method out there has already call to close it. Ok, tend. Let's close it.

if (this.pendingToClose) {

this.hideLoading();

}

}

}

También se podrían haber incorporado la carga de alerts o incluso la gestión de componentes modales. Por simplicidad, solo se ha incluido la gestión del modal de carga.

Entries que implementa todas las funcionalidades relacionadas con el sistema CRUD en la base de datos.

ionic generate service services/entries --skipTest

import { Injectable } from '@angular/core';

import { AngularFirestoreCollection } from '@angular/fire/firebase-node';

import { AngularFirestore } from '@angular/fire/firestore';

import { environment } from '../../environments/environment';

import { Entry } from '../model/entry';

import { AuthService } from './auth.service';

import { AngularFireStorage } from '@angular/fire/storage';

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class EntriesService {

entriesCollection: AngularFirestoreCollection<any>; // entries collection

uploadPercent: Observable<number>; // percent of uploading on storage service

downloadURL: Observable<string>; // url to get pics from storage service

entryLoaded = null; // last entry loaded

lastEntryLoaded = null; // very last entry loaded, if this is the same that entryLoaded, there is nothing to load

constructor(

private fireStore: AngularFirestore,

private auth: AuthService,

private storage: AngularFireStorage

) {

this.entriesCollection = fireStore.collection<any>(

environment.entriesCollection // check collection permission

);

}

// Get all entries. It is useful to show all them on the map.

public getAllEntries(center?) {

return this.entriesCollection.ref.get();

}

/\*\*

\* Very importan method. Get all entries of the logged user order by date.

\* @param reload If reload is true, reload 10 entries from the beginning.

\* If it is false, load 10 more from the last loaded. This is very useful for the

\* infinite scroll component: see tab1 component.

\*/

getEntries(reload?): Promise<Entry[]> {

if (reload) {

// reload all from the beginning?

this.lastEntryLoaded = null; // delete the pointer to last entry loaded

}

this.entryLoaded = this.lastEntryLoaded; // pointer to pointer for the next call

// this method is asyncronous, because the response is not instantaneous

return new Promise((resolve, reject) => {

const lreq: Entry[] = []; // an array of entries is the response

if (!this.auth.user) {

reject('No logged in'); // if you are not logged, there is nothing to do here, babe

}

let query;

if (this.entryLoaded == null) {

/\*\*

\* First, we get 10 first entries, order by date desc

\*/

query = this.entriesCollection.ref // it requires create a complex index on console <- IMPORTANT

.where('owner', '==', this.auth.user.email)

.orderBy('date', 'desc')

.limit(10)

.get();

} else {

/\*\*

\* Here, we get 10 more from the last loaded firstly

\*/

query = this.entriesCollection.ref

.where('owner', '==', this.auth.user.email)

.orderBy('date', 'desc')

.startAfter(this.entryLoaded)

.limit(10)

.get();

}

query.then(d => {

// we iterate through the array

d.forEach(u => {

const x = { key: u.id, ...u.data() }; // create an object width key and the other fields populated

lreq.push(x);

});

if (d.docs.length > 0) {

this.lastEntryLoaded = d.docs[d.docs.length - 1]; // last entry loaded

}

resolve(lreq);

});

});

}

/\*\*

\* Save a new entry or update it. When we create new one, fistly we have to store the pic in the storage service,

\* then get the download url and save all the info (title, description, url...) in firestore database.

\* @param data data to be stored or updated

\* @param key (mandatory for updating) Key of the entry to be updated

\*/

public saveEntry(data: Entry, key?: string) {

if (key) { // update an entry

return this.entriesCollection.doc(key).update(data); // if image doesnt exist update fails

} else { // create new one

return new Promise((resolve, reject) => {

const fileRef = this.storage.ref(

'/images/' +

this.auth.user.email +

new Date().getMilliseconds() +

'.jpeg'

);

const blob = this.base64ToBlob(data.image.base64String, 'image/jpeg');

const task = fileRef.put(blob);

this.uploadPercent = task.percentageChanges();

task.then(() => {

fileRef.getDownloadURL().subscribe(

Url => {

data.url = Url;

data.image = null;

// now save entry

this.entriesCollection.add(data).then(k => {

console.log(k.id);

resolve();

});

},

error => {

// Handle error here

// Show popup with errors or just console.error

console.error(error);

reject('error');

}

);

});

});

}

}

/\*\*

\* For Safari on IOS

\* onFetch(event) {

\* if (event.request.url.indexOf('firebasestorage.googleapis.com') !== -1) { return; }

\* ngsw-worker.js after building in www folder

\* @param base64

\* @param mime

\*/

private base64ToBlob(base64, mime) {

mime = mime || '';

const sliceSize = 1024;

const byteChars = window.atob(base64);

const byteArrays = [];

for (

let offset = 0, len = byteChars.length;

offset < len;

offset += sliceSize

) {

const slice = byteChars.slice(offset, offset + sliceSize);

const byteNumbers = new Array(slice.length);

for (let i = 0; i < slice.length; i++) {

byteNumbers[i] = slice.charCodeAt(i);

}

const byteArray = new Uint8Array(byteNumbers);

byteArrays.push(byteArray);

}

return new Blob(byteArrays, { type: mime });

}

public removeEntry(e): Promise<any> {

return new Promise((resolve, reject) => {

this.removeImage(e.url).subscribe(() => {

resolve(this.entriesCollection.ref.doc(e.key).delete());

});

});

}

/\*\*

\* When we remove an entry, this method removes its image on storage

\* We have to transform the download URL to the name of the file in storage

\* @param url

\*/

public removeImage(url): Observable<any> {

let name = url.substr(

url.indexOf('%2F') + 3,

url.indexOf('?') - (url.indexOf('%2F') + 3)

); // gets the name the image from the url

name = name.replace('%40', '@'); // unescapes specials caracters from url

const fileRef = this.storage.ref(`/images/${name}`);

return fileRef.delete();

}

}

Nota: Para PWA en sistemas IOS con Safari se ha detectado un bug que no permite subir el archivo al servidor por fallo de offset. Se ha solucionado insertando la línea:

if (event.request.url.indexOf('firebasestorage.googleapis.com') !== -1) { return; }

Justo después de la sentencia:

onFetch(event) {

Del archivo ngsw-worker.js que se encuentra en la carpeta www una vez se realiza el build.

Es importante señalar que la función:

private base64ToBlob(base64, mime) {

Podría implementarse con la herramienta fetch:

fetch(image) //base64

.then( response => response.blob() )

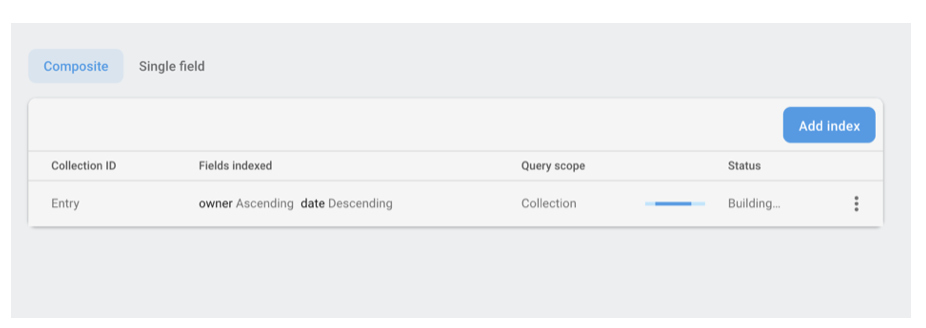
.then( blob =>{

//save blob into storage

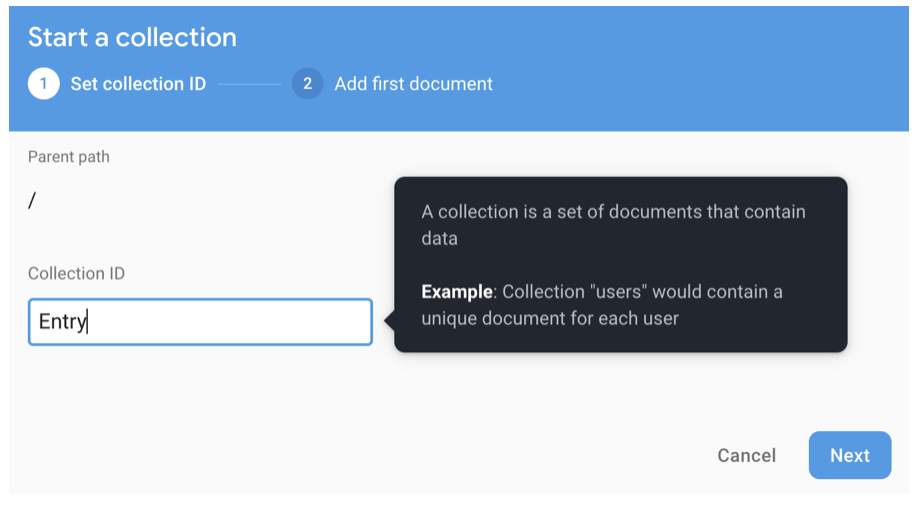
Por otra parte, la primera vez que ejecutemos getEntries dado que emplea la combinación de where con ordeBy nos arrojará el siguiente error:



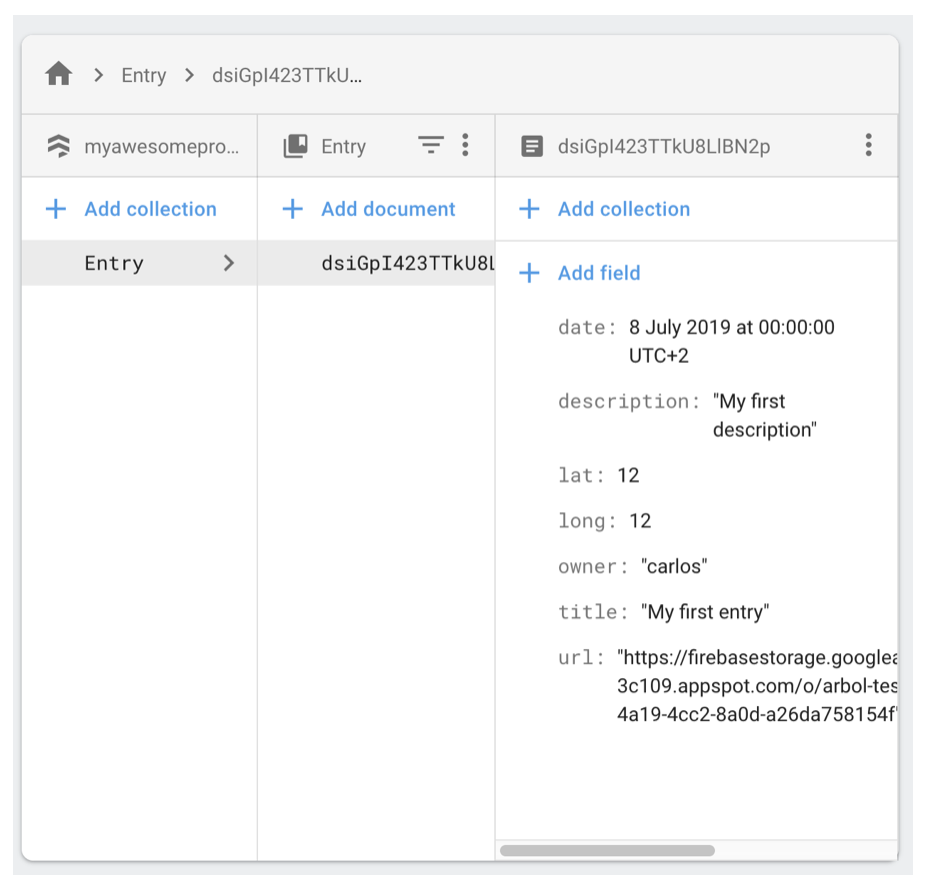
Siguiendo el enlace del error nos llevará a la consola para crear un índice en la base de datos que permite realizar este tipo de consultas.



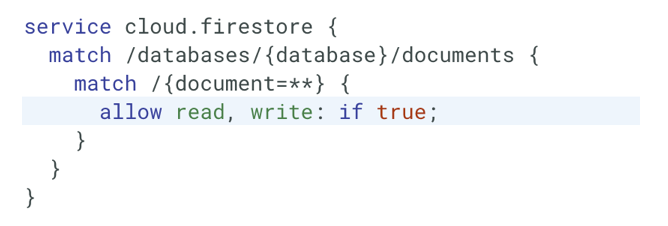
Además, deberemos crear la propia colección base: Entry.



Una vez se ha creado una entrada la estructura de datos es como sigue:



Para el desarrollo de pruebas en este ejemplo vamos a cambiar las reglas de acceso a la base de datos firestore:



Auth que contienen el sistema de autenticación y sesión. Uno de los que más problemas ha presentado durante el desarrollo de este ejemplo.

ionic generate service services/auth --skipTest

import { Injectable } from '@angular/core';

import { auth } from 'firebase';

import { AngularFireAuth } from '@angular/fire/auth';

import { User } from '../model/user';

import { Router } from '@angular/router';

import { AlertController } from '@ionic/angular';

import { GooglePlus } from '@ionic-native/google-plus/ngx';

@Injectable({

providedIn: 'root'

})

export class AuthService {

// keep in RAM user credentials (email if it's logged in, null if it's not logged in, see user model interface)

public user: User;

constructor(

private authS: AngularFireAuth,

private router: Router,

public alertController: AlertController,

private googlePlus: GooglePlus

) {

this.isLogged(); // check user at the beggining

}

/\*\*

\* This method is not used in this app. In this app only Google crendentials are allowed

\* @param user

\* @param pass

\*/

async login(user: string, pass: string) {

return new Promise(async (resolve, reject) => {

try {

// it uses Google auth service by user and password.

const res = await this.authS.auth.signInWithEmailAndPassword(

user,

pass

);

this.user = res.user;

resolve(true);

} catch (err) {

reject(err);

}

});

}

/\*\*

\* It is really interesting method. it tries to login by Google credentials, first by native component

\* in mobile devices, if it fails, it tries to do it by redirecting web mode, for pwa

\*/

async loginGoogle() {

return new Promise(async (resolve, reject) => {

this.googlePlus

.login({}) // trying native mode

.then(res => {

this.user = {

email: res.email // the response is different

};

localStorage.setItem('user', JSON.stringify(this.user));

resolve(true);

})

.catch(async err => {

console.error(err); // trying pwa mode

try {

// it uses Google auth by google credentials with pop up window

console.log('Trying pwa mode');

const provider = new auth.GoogleAuthProvider();

const res = await this.authS.auth.signInWithPopup(provider);

this.user = res.user;

localStorage.setItem('user', JSON.stringify(this.user));

resolve(true);

} catch (err) {

console.log(err);

reject(err);

}

});

});

}

/\*\*

\* it returns true is user is logged in.

\*/

public isLogged(): boolean {

if (this.user != null) {

return true; // it is already logged and stored in RAM (user variable)

} else {

// is it stored in cookies (we keep user logged in by default)?

if (localStorage.getItem('user')) {

this.user = JSON.parse(localStorage.getItem('user')); // ok, let's store in variable

return true;

} else {

// if not in cookies, is it logged in cloud (server keeps cookies too)?

this.authS.authState.subscribe(user => {

if (user) {

this.user = user; // YES, let's store in cookies and in RAM (variable)

localStorage.setItem('user', JSON.stringify(this.user));

return true;

} else {

// it is definitely not logged id. Let's remove from cookies to avoid bugs

localStorage.removeItem('user');

return false;

}

});

}

}

}

/\*\*

\* Sign out in Cloud, in Cookies and in RAM. Then, user is redirected to login page.

\*/

public async logOut() {

try {

await this.googlePlus.logout();

this.user = null;

localStorage.removeItem('user');

this.router.navigate(['']);

} catch (err) {

try {

console.log('logout pwa');

await this.authS.auth.signOut(); // next line executed when is loggout in cloud

this.user = null;

localStorage.removeItem('user');

this.router.navigate(['']);

} catch (err) {

console.log(err);

}

}

}

/\*\*

\* Router Guard. This method is called from routing module.

\* If it returns true, then the page that user is trying to visit will be recheable.

\* The page won't be loaded if it returns false

\*/

public canActivate(): boolean {

if (!this.isLogged()) {

// this avoid bugs if user enters the url http//localhost:8100/tab2 directly.

// The page wouldn't be recheable (thanks to routing module and canActivate call), but a blank

// page'd be loaded. To avoid this, we make sure to redirect user to login page.

this.router.navigate(['/tabs/tab1']);

return false;

}

return true;

}

/\*\*

\* we don't use other type of Router Guard, there are here only for demostration

\*/

public canActivateChild(): boolean {

return true;

}

public canDeactivate(): boolean {

return true;

}

public canLoad(): boolean {

return true;

}

public resolve(): boolean {

return true;

}

async presentAlert() {

const alert = await this.alertController.create({

header: 'Alert',

subHeader: 'No Logged On',

message:

'User Credentials incorrect or service unavailable. Check your internet conection',

buttons: ['OK']

});

await alert.present();

}

}

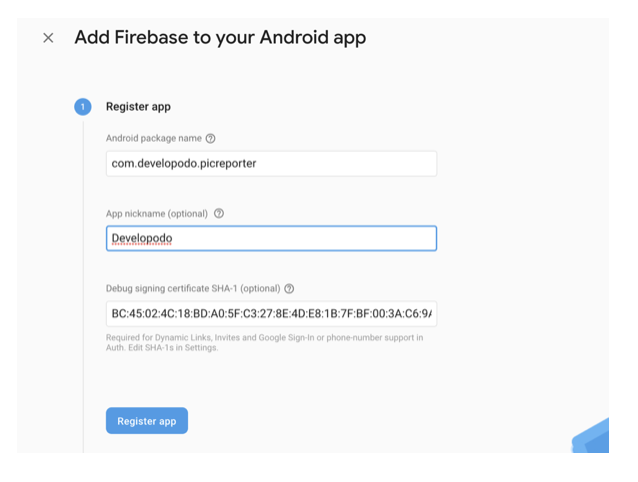
La mayoría del servicio es similar al implementado en el proyecto de Angular donde fue explicado. No obstante, hay una diferencia vital. El inicio de sesión mediante Google no es soportado en app híbridas. Aunque sigue siendo válido para entornos web: PWA.

¿Cómo implementamos un inicio de sesión mediante Google en nativo?

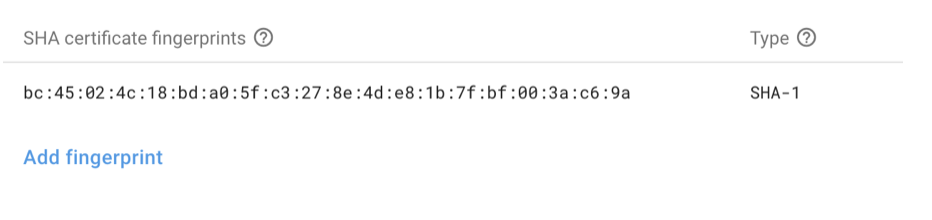
Vamos a emplear un plugin de Cordova, pero su instalación en la versión que hemos empleado en este documento no ha sido nada trivial.

Primero debemos ir a la consola de firebase, más concretamente a su configuración.

Añadiremos una aplicación, en nuestro caso Android.



Tras dar los datos necesarios para su creación, añadiremos un fingerprint.



¿De dónde sale? Es el certificado con el que se firma la app para subir al PlayStore. En nuestro caso, dado que estamos en pruebas lo obtenemos de ejecutar:

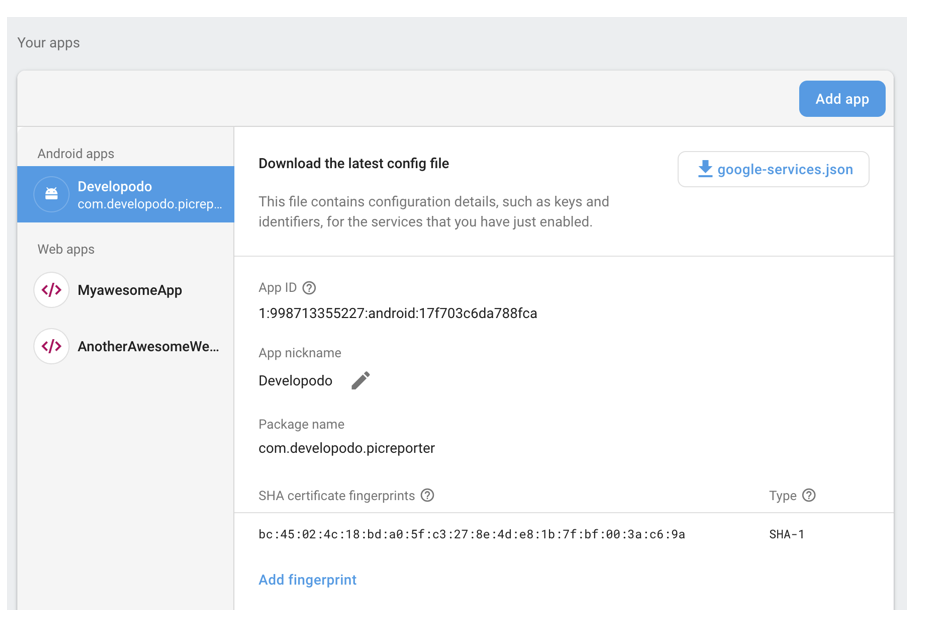
keytool -exportcert -list -v -alias androiddebugkey -keystore ~/.android/debug.keystore

Nota: la ruta del keystore de pruebas puede variar en sistemas Windows.

Antes de continuar, debemos crear la carpeta android en nuestro proyecto donde se generará el código necesario para crear nuestro apk, para ello ejecutamos:

npx cap add android

Una vez añadido, descargamos el archivo google-services.json y lo colocamos en la carpeta /android/app de nuestro proyecto.



En este archivo existen datos que necesitamos para instalar el plugin, como son:

},

"oauth\_client": [

{

"client\_id": "998713355227-d8kjqemanbvijqiqak422jvgevj1f3m4.apps.googleusercontent.com",

Con este client\_id construiremos dos campos:

* WEB\_APPLICATION\_CLIENT\_ID que será justamente nuestro client\_id
* REVERSED\_CLIENT\_ID que será nuestro client\_id invertido: com.googleusercontent…

Ahora si podemos ejecutar la instalación de nuestro plugin:

cordova plugin add cordova-plugin-googleplus --save --variable REVERSED\_CLIENT\_ID=com.googleusercontent.apps.998713355227-d8kjqemanbvijqiqak422jvgevj1f3m4 --variable WEB\_APPLICATION\_CLIENT\_ID=998713355227-d8kjqemanbvijqiqak422jvgevj1f3m4.apps.googleusercontent.com

npm add @ionic-native/google-plus

ionic cap sync

ionic cap update

Nota: cambiar los valores REVERSED\_CLIENT\_ID y WEB\_APPLICATION\_CLIENT\_ID acordes a cada proyecto.

Estos comandos, además de instalar el plugin Cordova y sincronizarlo con Capacitor deben haber modificado el archivo /config.xml incluyendo las siguientes líneas:

<plugin name="cordova-plugin-googleplus" spec="7.0.2">

<variable name="REVERSED\_CLIENT\_ID" value="com.googleusercontent.apps.998713355227-d8kjqemanbvijqiqak422jvgevj1f3m4" />

<variable name="WEB\_APPLICATION\_CLIENT\_ID" value="998713355227-d8kjqemanbvijqiqak422jvgevj1f3m4.apps.googleusercontent.com" />

</plugin>

Con estos pasos debería funcionar el plugin, no obstante, en la versión empleada en este documento y durante el desarrollo de este ejemplo surgieron diferentes errores (código 10 respuesta de Google Plus) que fueron solucionados con los siguientes pasos adicionales:

Añadiendo o modificando la siguiente línea en el archivo /android/app/build.gradle:

dependencies {

…

implementation “com.google.android.gms:play-services-auth:16.0.1"

Añadiendo o modificando las siguientes líneas en el archivo /android/capacitor-cordova-android-plugins/build.gradle:

dependencies {

…

// SUB-PROJECT DEPENDENCIES START

implementation "com.google.android.gms:play-services-auth:12+"

implementation "com.google.android.gms:play-services-identity:12+"

// SUB-PROJECT DEPENDENCIES END

}

Por último, es importante hacer coincidir el Package name que le hemos puesto a nuestra app en la consola de firebase con el id de nuestra aplicación dentro del proyecto. Por ejemplo, en el archivo config.xml:

<widget id="com.developodo.picreporter" version=“0.0.1”…

Así como en /android/app/build.gradle:

android {

compileSdkVersion 28

defaultConfig {

applicationId “com.developodo.picreporter"

Nota: además de añadir la plataforma Android, en capacitor podemos añadir las siguientes más, además de crear una PWA:

npx cap add ios

npx cap add electron

Sigamos con los contenidos de las pestañas. Comenzamos por la pestaña 3 y sus dependencias. Solo aparecen los archivos que requieren modificaciones:

tab3.page.html

<ion-content>

<div id="map" style="width:100%; height:100%;"></div>

</ion-content>

tab3.page.ts

En esta pestaña capturamos nuestra posición actual y mostramos un mapa (leafmap: <https://leafletjs.com/>) centrado en ella con iconos mostrando todas las imágenes que existen en la base de datos geoposicionadas. Para mostrarlas se usa un icono personalizado que al hacer click abre un modal con un componente realizado para visualizar la imagen.

Además del mapa leafmap, empleamos un plugin que nos permite clusterizar los iconos en función de su cercanía y el zoom del mapa. Para instalar tanto el mapa como el plugin, ejecutamos:

npm install leaflet --save

En el archivo /angular.json añadimos:

"assets": [

...,

{

"glob": "\*\*/\*",

"input": "./node\_modules/leaflet/dist/images",

"output": "leaflet/"

}

],

"styles": [

...,

"./node\_modules/leaflet/dist/leaflet.css"

],

Para el plugin(<https://github.com/Leaflet/Leaflet.markercluster>):

npm install @types/leaflet

npm install leaflet.markercluster

npm install --save @types/leaflet.markercluster

npm install --save @types/node

El código de tab3.page.ts:

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

import { Plugins } from '@capacitor/core';

import { EntriesService } from '../services/entries.service';

import { ViewImageComponent } from '../customComponents/view-image/view-image.component';

import { ModalController } from '@ionic/angular';

import { CommonsService } from '../services/commons.service';

/\*\*

\* npm install leaflet --save

\* In angular.json ->

\* "assets": [

\* ...,

\* {

\* "glob": "\*\*\/\*",

\* "input": "./node\_modules/leaflet/dist/images",

\* "output": "leaflet/"

\* }

\* ],

\* "styles": [

\* ...,

\* "./node\_modules/leaflet/dist/leaflet.css"

\* ],

\*/

import \* as L from 'leaflet';

/\*\*

\* https://github.com/Leaflet/Leaflet.markercluster

\* npm install @types/leaflet

\* npm install leaflet.markercluster

\* npm install --save @types/leaflet.markercluster

\* npm install --save @types/node

\*/

import 'leaflet.markercluster';

const { Geolocation } = Plugins;

@Component({

selector: 'app-tab3',

templateUrl: 'tab3.page.html',

styleUrls: ['tab3.page.scss']

})

export class Tab3Page {

/\*\* to load leaflet map \*/

map: L.Map;

constructor(

private cloud: EntriesService,

private modalController: ModalController,

private commonS: CommonsService,

) {}

async ionViewDidEnter() {

this.commonS.showLoading('Enabling location');

let coor;

try {

if (navigator.mediaDevices && navigator.mediaDevices.enumerateDevices) {

coor = await Geolocation.getCurrentPosition(); // original method to get current position with Capacitor

coor = coor.coords;

} else { // for some weid situations, such as Chome on IOS

coor = await this.getW3CPosition();

coor = coor.coords;

}

this.commonS.hideLoading();

this.loadMap(coor)

.then(() => {

// this.loadAllData(); <--if we don't use clusters

this.loadClusters();

})

.catch(err => {

console.log(err);

});

} catch (err) {

coor = {

latitude: 0,

longitude: 0

};

console.log(err);

this.commonS.hideLoading();

this.loadMap(coor)

.then(() => {

console.log('load mapa');

// this.loadAllData(); <--if we don't use clusters

this.loadClusters();

})

.catch(err => {

console.log(err);

});

}

}

async loadMap(coor) {

// We load leafmap centered on our current position, with zoom 14

this.map = new L.Map('map').setView(

[coor.latitude, coor.longitude],

14 // current zoom, about a medium size city, before 12

);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {

attribution: `<div style="text-align:center;display:inline-block">Map data &copy;

<a href="https://www.openstreetmap.org/">OpenStreetMap</a> contributors<br>

<a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, Imagery ©

<a href="https://www.mapbox.com/">Mapbox</a><br> App by Developodo</div>`,

maxZoom: 19 // before 18

}).addTo(this.map); // add to div, see tab3.page.html

}

loadClusters() {

const markers = L.markerClusterGroup({ chunkedLoading: true });

this.cloud

.getAllEntries()

.then(d => {

const customMarkerIcon = L.icon({

iconUrl: '/assets/iconmap.png',

iconSize: [32, 32],

popupAnchor: [0, -20]

});

d.docs.forEach(e => {

const entry = e.data();

const mmarker = L.marker(L.latLng(e.data().lat, e.data().long), {

icon: customMarkerIcon,

title: e.data().title

})

.bindPopup(

e.data().owner +

'<br>' +

e.data().title +

'<br>' +

e.data().date +

'<hr>' +

e.data().description

)

.openPopup()

.on('click', () => this.viewImage(entry));

markers.addLayer(mmarker);

});

this.map.addLayer(markers);

this.map.on('moveend', () => {

/\*\*

\* This is useful for geofire, here this is not useful

\*/

console.log(this.map.getCenter().toString());

console.log(this.map.getZoom().toString());

});

})

.catch(err => {

console.log(err);

});

}

/\*\*

\* this method is not used. It is a sample to make markers without clusters

\*/

loadAllData() {

const customMarkerIcon = L.icon({

iconUrl: '/assets/iconmap.png',

iconSize: [32, 32],

popupAnchor: [0, -20]

});

this.cloud

.getAllEntries()

.then(d => {

d.docs.forEach(e => {

const m = L.marker([e.data().lat, e.data().long], {

icon: customMarkerIcon

})

.addTo(this.map)

.bindPopup(e.data().title + '<br>' + e.data().date, {

autoClose: false

})

.openPopup()

.on('click', () => this.viewImage(e.data().url));

});

this.map.on('moveend', () => {

console.log(this.map.getCenter().toString());

console.log(this.map.getZoom().toString());

});

})

.catch(err => {

console.log(err);

});

}

/\*\*

\* Show the image in the modal ViewImageComponent

\* @param u

\*/

async viewImage(u: any) {

const modal = await this.modalController.create({

component: ViewImageComponent,

componentProps: { entry: u }

});

return await modal.present();

}

/\*\* Remove map when we have multiple map object \*/

ionViewWillLeave() {

if (this.map && this.map.remove) {

this.map.remove();

}

}

private getW3CPosition() {

return new Promise((resolve, reject) => {

// Try W3C Geolocation (Preferred)

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition(

position => {

resolve(position);

},

error => {

reject(error);

},

{}

);

} else {

reject();

}

});

}

}

Es importante señalar, que durante el desarrollo se observaron ciertos problemas al ejecutar la aplicación en Google Chrome sobre IOS. Para solucionarlos hemos creado un método getW3CPosition que nos saca del apuro.

Por otra parte, para que funcione correctamente hay que crear un icono de 32x32 y almacenarlo en assets, será el icono personalizado que se mostrará en el mapa indicando que existe un reporte en esa posición: /assets/iconmap.png

Este componente hace uso de otro como modal para visualizar las imágenes:

ionic g c customComponents/viewImage --spec=false

view-image.component.html

<div class="container">

<div class="vertical-center">

<div class="info1">{{this.entry.title}} | {{this.entry.date | date : "dd/MM/y H:mm"}}</div>

<img [src]="this.entry.url" alt="loading image...">

<div class="info">{{this.entry.description}}<br>By {{this.entry.owner}}</div>

</div>

</div>

view-image.component.scss

.container {

height: 100%;

position: relative;

border: 2px solid #666;

background-color: black

}

.vertical-center {

margin: 0;

position: absolute;

top: 50%;

-ms-transform: translateY(-50%);

transform: translateY(-50%);

}

img {

max-width: 100%;

max-height: 100%;

height:auto;

}

.info1{

position: fixed;

padding:5px;

width:100%;

top:0px;

z-index:9999;

background-color: rgba(0, 0, 0, 0.4);

color:white;

}

.info{

position: fixed;

padding:5px;

width:100%;

bottom:0px;

z-index:9999;

background-color: rgba(0, 0, 0, 0.4);

color:white;

}

view-image.component.ts

import { Component, OnInit } from '@angular/core';

import { NavParams, ModalController, NavController } from '@ionic/angular';

import { Entry } from '../../model/entry';

import { BackbuttonService } from '../../services/backbutton.service';

@Component({

selector: 'app-view-image',

templateUrl: './view-image.component.html',

styleUrls: ['./view-image.component.scss']

})

export class ViewImageComponent implements OnInit {

entry: Entry;

constructor(private navParams: NavParams,

private modalController: ModalController,

private back: BackbuttonService,

private navCtrl: NavController) {

this.back.openModal = true;

}

ngOnInit() {

// hack for pwa. backbutton close the modal: see app.component.ts

history.pushState({modal: true}, 'Opening a modal', 'Modal');

this.back.modalOpened = this;

this.entry = this.navParams.get('entry');

}

public closeMe() {

this.navCtrl.pop();

this.modalController.dismiss({ r: false }); // updating not required

}

}

Es importante señalar que este componente presenta carencias que una versión de producción habría que solventar: carga progresiva de la imagen, maquetación correcta de la información, entre las más destacables.

Es interesante como modifica el historial para los PWA, permitiendo funcionar el botón de atrás como cierre del modal y no como visitar la anterior url.

// new url +/Modal entered

history.pushState({modal: true}, 'Opening a modal', ‘Modal'); this.back.modalOpened = this; // backbutton will close this modal

Este componente es empleado tanto en tab3 como en tab1, por lo que vamos a definirlos en un nivel superior como entryComponent, concretamente en /src/app/tabs/tabs.module.ts:

import { IonicModule } from '@ionic/angular';

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { TabsPageRoutingModule } from './tabs.router.module';

import { TabsPage } from './tabs.page';

import { EditPageComponent } from '../customComponents/edit-page/edit-page.component';

import { ViewImageComponent } from '../customComponents/view-image/view-image.component';

@NgModule({

entryComponents:[EditPageComponent,ViewImageComponent],

imports: [

IonicModule,

CommonModule,

ReactiveFormsModule, // modal form component

FormsModule,

TabsPageRoutingModule

],

declarations: [TabsPage,EditPageComponent,ViewImageComponent]

})

export class TabsPageModule {}

Vemos que además emplea otro componente en modal que veremos en tab1 denominado EditPageComponent. Este último hace uso de módulos como ReactiveFormsModule.

Para completar, vemos como ha quedado el archivo tabs.page.html:

<ion-tabs>

<ion-tab-bar slot="bottom" color="dark">

<ion-tab-button tab="tab1">

<ion-icon name="home"></ion-icon>

<ion-label>Home</ion-label>

</ion-tab-button>

<ion-tab-button tab="tab2">

<ion-icon name="aperture"></ion-icon>

<ion-label>New Pic</ion-label>

</ion-tab-button>

<ion-tab-button tab="tab3">

<ion-icon name="map"></ion-icon>

<ion-label>Map</ion-label>

</ion-tab-button>

</ion-tab-bar>

</ion-tabs>

Pasamos ahora al código de tab2. En ella se obtiene la posición actual y se muestra la cámara de Capacitor. Si realizamos la foto, se muestra un formulario en un modal para añadir la info a la base de datos, mostrando el porcentaje de imagen subida a store.

tab2.page.html

<ion-content>

<img \*ngIf="showPic" [src]="photo" width="100%" />

</ion-content>

tab2.page.ts

import { Component, OnInit } from '@angular/core';

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

/\*\*

\* Next, we need to tell Angular to trust the dynamic image data.

\* To do this, inject DomSanitizer via the Constructor and use sanitizer.

\* bypassSecurityTrustResourceUrl() to allow the image data to be displayed in our app:

\*/

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

import { AlertController, ModalController } from '@ionic/angular';

import { Entry } from '../model/entry';

import { CommonsService } from '../services/commons.service';

import { EditPageComponent } from '../customComponents/edit-page/edit-page.component';

import { AuthService } from '../services/auth.service';

import { Router } from '@angular/router';

import { async } from 'q';

const { Geolocation } = Plugins;

@Component({

selector: 'app-tab2',

templateUrl: 'tab2.page.html',

styleUrls: ['tab2.page.scss']

})

export class Tab2Page implements OnInit {

photo: SafeResourceUrl;

newEntry: Entry;

showPic = false;

accuracy: number;

confirmSave = false;

watcher: any;

constructor(

private sanitizer: DomSanitizer,

private alertController: AlertController,

private auth: AuthService,

private modalController: ModalController,

private router: Router,

private commonS: CommonsService

) {}

ngOnInit() {}

ionViewWillEnter() {

this.accuracy = 0;

this.newEntry = {

owner: '',

url: '',

title: '',

description: '',

date: new Date(),

lat: 0,

long: 0

};

this.showPic = false;

this.photo = null;

}

ionViewWillLeave() {}

async ionViewDidEnter() {

this.commonS.showLoading('Enabling location');

let coor: any;

// Get current position

try {

if (navigator.mediaDevices && navigator.mediaDevices.enumerateDevices) {

coor = await Geolocation.getCurrentPosition();

} else { // this is a weird situation: google chrome on IOS

coor = await this.getW3CPosition();

}

this.newEntry.lat = coor.coords.latitude;

this.newEntry.long = coor.coords.longitude;

this.accuracy = Math.round(coor.coords.accuracy);

this.commonS.hideLoading();

// next step

this.addEntry();

} catch (err) {

this.newEntry.lat = 0;

this.newEntry.long = 0;

console.log(err);

this.commonS.hideLoading();

// next step, even though we dont have position <- this can be done better

this.addEntry();

}

}

// Next step after getting position

async addEntry() {

// We populate some other data to the form

this.newEntry.owner = this.auth.user.email;

this.newEntry.date = new Date();

this.newEntry.url = '';

// Time to launch camera

await this.takePicture();

}

// Lauching camera

async takePicture() {

try {

let image;

if (navigator.mediaDevices && navigator.mediaDevices.enumerateDevices) {

image = await Plugins.Camera.getPhoto({

quality: 60,

allowEditing: false,

saveToGallery:false,

correctOrientation:false,

/\*height: 600,\*/

resultType: CameraResultType.Base64,

source: CameraSource.Camera

});

} else { // Google Chrome on IOS can't lauch camera.

this.presentAlert('Your browser is not compatible with web camera. If you are on IOS, try Safari');

throw new Error("Are you using Chrome on IOS? We can't launch the camera, try it with Safari");

}

this.photo = this.sanitizer.bypassSecurityTrustResourceUrl(

image && 'data:image/jpeg;base64,' + image.base64String

); // important to inject this image in DOM

// asigns this one temporaly

this.newEntry.url = 'data:image/jpeg;base64,' + image.base64String;

this.newEntry.image = image;

this.showPic = true;

this.editModal(this.newEntry); // show form modal

} catch (err) {

this.commonS.hideLoading();

console.log(err);

if (err !== 'User cancelled photos app') { // <- this can be done much better

this.presentAlert();

}

this.router.navigate(['']); // <- if error, go to tab1

}

/\*\*

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

\* It's a separate dependency, so install it using the Terminal (cancel the ionic serve command currently running first):

\* npm install @ionic/pwa-elements

\* Import @ionic/pwa-elements by editing src/main.ts: import { defineCustomElements } from '@ionic/pwa-elements/loader';

\*/

}

async getCurrentPosition() {

try {

const coor = await Geolocation.getCurrentPosition();

this.newEntry.lat = coor.coords.latitude;

this.newEntry.long = coor.coords.longitude;

return true;

} catch (err) {

return false;

}

}

async presentAlert(msg?) {

const mes = msg

? msg

: 'Something was wrong. Have you got a camera in your device? Have you accepted permission to camera access?';

const alert = await this.alertController.create({

header: 'Alert',

subHeader: '',

message: mes,

buttons: ['OK']

});

await alert.present();

}

async editModal(e: Entry) {

const modal = await this.modalController.create({

component: EditPageComponent,

componentProps: e

});

modal.onDidDismiss().then(() => {

this.router.navigate(['']);

});

return await modal.present();

}

private getW3CPosition() {

return new Promise((resolve, reject) => {

// Try W3C Geolocation (Preferred)

if (navigator.geolocation) {

//alert('Geolocation is supported!');

} else {

alert('Geolocation is not supported for this Browser/OS.');

}

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition(

position => {

resolve(position);

},

error => {

alert(error);

reject(error);

},

{}

);

// Try Google Gears Geolocation

} else {

reject();

}

});

}

}

Es interesante como se ha obtenido la posición y la imagen desde la cámara. El componente cámara presenta varias interesantes opciones de personalización. Toda la cumplimentación de información se delega en un componente editPage que se presenta como modal, pasando toda la información captada hasta el momento:

async editModal(e: Entry) {

const modal = await this.modalController.create({

component: EditPageComponent,

componentProps: e

});

…

Para crear el componente editPage:

ionic g c customComponents/editPage --spec=false

Código del componente:

edit-page.component.html

<ion-header>

<ion-toolbar color="dark">

<ion-title>Entry properties</ion-title>

</ion-toolbar>

</ion-header>

<ion-content>

<ion-progress-bar

color="secondary"

[value]="(this.cloud.uploadPercent | async) / 100"

></ion-progress-bar>

<form [formGroup]="entryForm" (ngSubmit)="onSubmit()">

<ion-item>

<ion-label position="floating">Owner</ion-label>

<ion-input

placeholder="Owner"

[value]="this.entry.owner"

readonly

formControlName="owner"

></ion-input>

<ion-note slot="end">

<ion-avatar style="margin-left:10px">

<img [src]="this.pic" />

</ion-avatar>

</ion-note>

</ion-item>

<ion-item>

<ion-label position="floating">Title</ion-label>

<ion-input

placeholder="Title"

[value]="this.entry.title"

formControlName="title"

></ion-input>

</ion-item>

<ion-item>

<ion-label position="floating">Description</ion-label>

<ion-input

placeholder="Description"

[value]="this.entry.description"

formControlName="description"

></ion-input>

</ion-item>

<ion-item>

<ion-label>DD/MM/YYYY H:mm</ion-label>

<ion-datetime

displayFormat="DD/MM/YYYY H:mm"

formControlName="date"

></ion-datetime>

</ion-item>

<ion-button

type="submit"

expand="full"

color="primary"

disable="this.submitting"

>Save</ion-button

>

</form>

<ion-button

type="button"

expand="full"

color="danger"

(click)="this.closeMe()"

>Cancel</ion-button

>

<ion-progress-bar type="indeterminate" \*ngIf="submitting"></ion-progress-bar>

</ion-content>

Interesante como se ha reutilizado este modal tanto para crear la entrada, como para editarla, como veremos en tab1.

También resulta útil el uso de ion-progress-bar y el observable que devuelve el servicio cloud al subir una imagen:

[value]="(this.cloud.uploadPercent | async) / 100”

Como vemos el pipe async nos permite inyectar directamente un observable en un elemento del DOM.

edit-page.component.ts

import { Component, OnInit, Input } from '@angular/core';

import { NavParams, ModalController, NavController } from '@ionic/angular';

import { Entry } from '../../model/entry';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { EntriesService } from '../../services/entries.service';

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

import { CommonsService } from '../../services/commons.service';

import { BackbuttonService } from '../../services/backbutton.service';

@Component({

selector: 'app-edit-page',

templateUrl: './edit-page.component.html',

styleUrls: ['./edit-page.component.scss']

})

export class EditPageComponent implements OnInit {

pic: any;

entry: Entry;

today: Date;

submitting = false;

public entryForm: FormGroup;

constructor(

navParams: NavParams,

private formBuilder: FormBuilder,

private modalController: ModalController,

public cloud: EntriesService,

private sanitizer: DomSanitizer,

private commonS: CommonsService,

private back: BackbuttonService,

private navCtrl: NavController

) {

// avoid to close app when back button is pressed

this.back.openModal = true;

this.back.modalOpened = this;

history.pushState({modal: true}, 'Opening a modal', ' '); // hack for pwa. backbutton close the modal: see app.component.ts

this.entry = {

key: navParams.get('key'),

owner: navParams.get('owner'),

url: navParams.get('url'),

title: navParams.get('title'),

description: navParams.get('description'),

date: navParams.get('date'),

lat: navParams.get('lat'),

long: navParams.get('long'),

image: navParams.get('image')

};

if (this.entry.image) {

//When entry is just created, the url field doesn't have nice value here yet

console.log('On slow device');

this.pic = this.sanitizer.bypassSecurityTrustResourceUrl(

'data:image/jpeg;base64,' + this.entry.image.base64String

);

} else {

//When editing, yes

this.pic = this.entry.url;

}

this.today = new Date();

}

ngOnInit() {

this.entryForm = this.formBuilder.group({

owner: [this.entry.owner, [Validators.required, Validators.email]],

title: [

this.entry.title,

[Validators.required, Validators.maxLength(20)]

],

description: [

this.entry.description,

[Validators.required, Validators.maxLength(100)]

],

date: [

//if it's a date we convert it to string. From database is already a string.

this.entry.date.toISOString

? this.entry.date.toISOString()

: this.entry.date,

[Validators.required]

]

});

}

// convenience getter for easy access to form fields

public get f() {

return this.entryForm.controls;

}

onSubmit() {

console.log(this.f)

if (this.submitting) {

return;

}

this.submitting = true;

// stop here if form is invalid

if (this.entryForm.invalid) {

console.log('Invalid Form');

this.submitting = false;

// alert

return;

}

// update o create.

const newEntry = {

owner: this.entry.owner,

url: this.entry.url,

title: this.f.title.value,

description: this.f.description.value,

date: this.f.date.value,

lat: this.entry.lat,

long: this.entry.long,

image: this.entry.image

};

this.commonS.showLoading();

if (this.entry.key) {

newEntry.image = null;

this.cloud

.saveEntry(newEntry, this.entry.key)

.then(() => {

this.commonS.hideLoading();

this.modalController.dismiss({ r: true }); // updating required, it is read on dismiss this modal

})

.catch(err => {

console.log(err);

this.commonS.hideLoading();

});

} else {

this.cloud

.saveEntry(newEntry)

.then(() => {

this.commonS.hideLoading();

this.modalController.dismiss({ r: true }); // updating required

})

.catch(err => {

console.log(err);

this.commonS.hideLoading();

});

}

}

public closeMe() {

this.navCtrl.pop();

this.modalController.dismiss({ r: false }); // updating not required

}

}

Útil es el estudio del manejo del servicio cloud por parte de este componente y la gestión de la interfaz y su asincronía en las peticiones.

Otro aspecto interesante, es cómo recibe los parámetros pasados desde el componente padre:

this.entry = {

key: navParams.get('key'),

Y como devuelve valores para ser gestionados por el componente padre una vez cerrado el modal:

this.modalController.dismiss({ r: false })

En el componente padre, como se puede observar en tab2.page.ts, se gestionan los datos de la siguiente forma, aunque en este caso el valor de r es irrelevante:

modal.onDidDismiss().then((response) => {

this.router.navigate(['']);

});

Pasamos ahora a tab1 que a su vez emplea dos compenentes como vemos en su archivo

tab1.page.html:

<ion-header>

<ion-toolbar color="dark">

<ion-title>

Welcome

</ion-title>

</ion-toolbar>

</ion-header>

<ion-content>

<app-login \*ngIf="!this.auth.user"></app-login>

<app-list-images \*ngIf="this.auth.user"></app-list-images>

</ion-content>

Renderiza el componente Login si el usuario no ha iniciado sesión y el componente listImages si ya tiene la sesión abierta.

Para registrar estos componentes y emplearlos en este componente, modificamos tab1.modules.ts:

import { IonicModule } from '@ionic/angular';

import { RouterModule } from '@angular/router';

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { Tab1Page } from './tab1.page';

import { LoginComponent } from './login/login.component';

import { ListImagesComponent } from './list-images/list-images.component';

@NgModule({

imports: [

IonicModule,

CommonModule,

ReactiveFormsModule, // login component

FormsModule,

RouterModule.forChild([{ path: '', component: Tab1Page }])

],

declarations: [Tab1Page, LoginComponent, ListImagesComponent]

})

export class Tab1PageModule {

}

Ahora creamos los componentes:

ionic g c tab1/login --spec=false

ionic g c tab1/listImages --spec=false

Comencemos por el componente Login:

login.component.html

<form [formGroup]="registerForm" (ngSubmit)="onSubmit()">

<ion-input placeholder="Email Address" formControlName="email"></ion-input>

<div class="error" \*ngIf="this.submited && this.f.email.invalid">

Email is required

</div>

<ion-input

type="password"

placeholder="Password"

formControlName="password"

></ion-input>

<div class="error" \*ngIf="this.submited && this.f.password.invalid">

Password is required

</div>

<ion-button

type="submit"

expand="full"

color="primary"

disable="this.submitting"

>Log In</ion-button

>

</form>

<div class="gSignInWrapper">

<div id="customBtn" class="customGPlusSignIn" (click)="this.logingGoogle()">

<span class="icon">

<svg

version="1.1"

xmlns="http://www.w3.org/2000/svg"

width="18px"

height="18px"

viewBox="0 0 48 48"

class="abcRioButtonSvg"

>

<g>

<path

fill="#EA4335"

d="M24 9.5c3.54 0 6.71 1.22 9.21 3.6l6.85-6.85C35.9 2.38 30.47 0 24 0 14.62 0 6.51 5.38 2.56 13.22l7.98 6.19C12.43 13.72 17.74 9.5 24 9.5z"

></path>

<path

fill="#4285F4"

d="M46.98 24.55c0-1.57-.15-3.09-.38-4.55H24v9.02h12.94c-.58 2.96-2.26 5.48-4.78 7.18l7.73 6c4.51-4.18 7.09-10.36 7.09-17.65z"

></path>

<path

fill="#FBBC05"

d="M10.53 28.59c-.48-1.45-.76-2.99-.76-4.59s.27-3.14.76-4.59l-7.98-6.19C.92 16.46 0 20.12 0 24c0 3.88.92 7.54 2.56 10.78l7.97-6.19z"

></path>

<path

fill="#34A853"

d="M24 48c6.48 0 11.93-2.13 15.89-5.81l-7.73-6c-2.15 1.45-4.92 2.3-8.16 2.3-6.26 0-11.57-4.22-13.47-9.91l-7.98 6.19C6.51 42.62 14.62 48 24 48z"

></path>

<path fill="none" d="M0 0h48v48H0z"></path>

</g>

</svg>

</span>

<span class="buttonText">Singin with Google</span>

</div>

</div>

login.component.scss

.error {

color: red;

}

form {

margin: 8%;

}

.gSignInWrapper {

text-align: center;

border-top: 1px solid #888;

margin: 5%;

padding: 10px;

}

#customBtn {

display: inline-block;

background: white;

color: #444;

width: 190px;

border-radius: 5px;

border: thin solid #888;

box-shadow: 1px 1px 1px grey;

white-space: nowrap;

}

#customBtn:hover {

cursor: pointer;

}

span.label {

font-family: serif;

font-weight: normal;

}

span.icon {

display: inline-block;

padding-top: 10px;

text-align: center;

width: 42px;

height: 42px;

}

span.buttonText {

display: inline-block;

vertical-align: text-bottom;

font-size: 14px;

font-weight: bold;

/\* Use the Roboto font that is loaded in the <head> \*/

font-family: "Roboto", sans-serif;

}

login.component.ts

import { Component, OnInit } from '@angular/core';

import { FormGroup, FormBuilder, Validators } from '@angular/forms';

import { AuthService } from '../../services/auth.service';

import { Router } from '@angular/router';

import { CommonsService } from '../../services/commons.service';

@Component({

selector: 'app-login',

templateUrl: './login.component.html',

styleUrls: ['./login.component.scss']

})

export class LoginComponent implements OnInit {

public registerForm: FormGroup;

public submited: boolean;

public submitting: boolean;

constructor(

private formBuilder: FormBuilder,

private auth: AuthService,

private router: Router,

private commonS: CommonsService

) {

this.submitting = false;

}

ngOnInit() {

if (this.auth.isLogged()) {

this.router.navigate(['']);

}

this.registerForm = this.formBuilder.group({

email: ['', [Validators.required, Validators.email]],

password: ['', [Validators.required, Validators.minLength(6)]]

});

this.submited = false;

}

public isLogging() {

return this.submitting;

}

// convenience getter for easy access to form fields

public get f() {

return this.registerForm.controls;

}

onSubmit() {

if (this.submitting) {

return;

}

this.commonS.showLoading();

this.submited = true;

this.submitting = true;

// stop here if form is invalid

if (this.registerForm.invalid) {

console.log('Invalid Form');

this.submitting = false;

return;

}

this.auth

.login(this.f.email.value, this.f.password.value)

.then(res => {

this.commonS.hideLoading();

this.router.navigate(['']);

this.submitting = false;

})

.catch(err => {

this.commonS.hideLoading();

this.auth.presentAlert(); // alert changed to fit device mode

this.submitting = false;

});

}

public logingGoogle() {

if (this.submitting) { return; }

this.commonS.showLoading();

this.submitting = true;

this.auth

.loginGoogle()

.then(res => {

this.commonS.hideLoading();

this.submitting = false;

this.router.navigate(['']);

})

.catch(err => {

this.commonS.hideLoading();

this.submitting = false;

this.auth.presentAlert(); // alert changed to fit device mode

console.log(err);

});

}

}

Fundamentalmente, hace uso del servicio de autenticación que ya vimos anteriormente y gestiona las rutas en caso de éxito o error.

Para el componente listImages que muestra la lista de entradas del usuario. Deslizándolos hacia la derecha permite eliminarlos y hacia la izquierda editarlos mediante el modal editPage que vimos anteriormente.

list-images.component.html

<ion-refresher slot="fixed" (ionRefresh)="refreshEntries($event)">

<ion-refresher-content

pullingIcon="arrow-dropdown"

pullingText="Pull to refresh"

refreshingSpinner="circles"

refreshingText="Refreshing..."

>

</ion-refresher-content>

</ion-refresher>

<ion-list class="topP">

<ion-item>

<ion-label>{{ this.auth.user.email }}</ion-label>

<ion-toggle checked (ionChange)="this.logOut()"></ion-toggle>

</ion-item>

</ion-list>

<ion-virtual-scroll [items]="this.myEntries">

<!-- Sliding item with text options on both sides -->

<ion-item-sliding \*virtualItem="let item">

<ion-item-options side="start">

<ion-item-option (click)="editModal(item)">

<ion-icon slot="start" name="create"></ion-icon>

Edit</ion-item-option

>

</ion-item-options>

<ion-item>

<ion-label>

<h2>{{ item.title }}</h2>

<p>{{ item.description }}</p>

</ion-label>

<ion-note slot="end">

<span \*ngIf="item.date">{{

item.date.toLocaleString() | date: "dd/MM/y H:mm"

}}</span>

<ion-avatar style="margin-left:10px" (click)="this.viewImage(item)">

<img [src]="item.url" />

</ion-avatar>

</ion-note>

</ion-item>

<ion-item-options side="end">

<ion-item-option color="danger" (click)="removeEntry(item)">

<ion-icon slot="top" name="archive"></ion-icon>

Remove</ion-item-option

>

</ion-item-options>

</ion-item-sliding>

</ion-virtual-scroll>

<ion-infinite-scroll

#infiniteScroll

threshold="100px"

(ionInfinite)="loadMoreEntries($event)"

>

<ion-infinite-scroll-content

loadingSpinner="bubbles"

loadingText="Loading more data..."

>

</ion-infinite-scroll-content>

</ion-infinite-scroll>

Existen algunos componentes realmente interesantes en este código:

ion-refresher permite actualizar la lista de entradas al deslizar la lista hacia abajo a modo de recarga.

ion-infinite-scroll permite llamar a una función cuando se realiza un scroll cercano al final de la pantalla (se puede personalizar con el parámetro threshold), permitiendo ir cargando elementos a medida que se necesiten. Este método ejecutado hace uso del servicio entries que carga de 10 en 10 entradas.

ion-virtual-scroll permite mostrar una lista de items con la ventaja de que solo se renderizan en el DOM aquellos que son visibles por el usuario en cada momento. Optimizando los recursos en caso de listas que contengan contenidos multimedia. Es interesante como se realiza la iteración sobre la lista de datos que existen en el controlador (archivo ts):

<ion-virtual-scroll [items]="this.myEntries">

<ion-virtual-scroll [items]= "this.myEntries">

<ion-item-sliding \*virtualItem="let item">

La variable array es this.myEntries que es iterada y almacenada en cada ciclo en la variable item.

ion-item-sliding que permite mostrar elementos en una lista con capacidad de deslizamiento para gestionar acciones.

Por último, señalar al elemento:

<span \*ngIf="item.date">{{

item.date.toLocaleString() | date: "dd/MM/y H:mm"

}}</span>

Que por una parte presenta la directiva ngIf que permite valorar si el valor de una variable es verdadero o distinto de null para renderizar al elemento. Y por otra parte, el pipe date que permite dar formato a fechas directamente en el archivo HTML.

list-images.component.scss

.topP {

position: sticky;

top: 0;

left: 0;

right: 0;

z-index: 9999;

}

list-images.component.ts

import { Component, OnInit, ViewChild } from '@angular/core';

import { AuthService } from '../../services/auth.service';

import { Entry } from '../../model/entry';

import { EntriesService } from '../../services/entries.service';

import {

IonInfiniteScroll,

ModalController,

AlertController

} from '@ionic/angular';

import { EditPageComponent } from '../../customComponents/edit-page/edit-page.component';

import { IonVirtualScroll } from '@ionic/angular';

import { ViewImageComponent } from 'src/app/customComponents/view-image/view-image.component';

import { CommonsService } from '../../services/commons.service';

@Component({

selector: 'app-list-images',

templateUrl: './list-images.component.html',

styleUrls: ['./list-images.component.scss']

})

export class ListImagesComponent implements OnInit {

@ViewChild('infiniteScroll') ionInfiniteScroll: IonInfiniteScroll;

@ViewChild(IonVirtualScroll) virtualScroll: IonVirtualScroll;

resetInfinityScroll: IonInfiniteScroll;

myEntries: Entry[] = [];

constructor(

public auth: AuthService,

public cloud: EntriesService,

public modalController: ModalController,

private alertController: AlertController,

private commonS: CommonsService

) { }

ngOnInit() {

this.loadEntries();

// important to do it here, when dom is made. infinite scroll need this solution

}

public logOut() {

this.auth.logOut();

}

public async loadEntries() {

this.commonS.showLoading();

this.myEntries = await this.cloud.getEntries(true);

// next line is very important to update dinamic content into virtualscroll

this.virtualScroll.checkEnd();

this.commonS.hideLoading();

// hide loading

}

public async refreshEntries(e) {

await this.loadEntries();

// close the refresher

e.target.complete();

}

public async loadMoreEntries(e) {

this.commonS.showLoading();

const newEntries = await this.cloud.getEntries(false); // to add more

if (newEntries.length > 0) {

this.myEntries = this.myEntries.concat(newEntries);

this.virtualScroll.checkEnd();

}

this.commonS.hideLoading();

e.target.complete();

}

async editModal(e: Entry) {

const modal = await this.modalController.create({

component: EditPageComponent,

componentProps: e

});

modal.onDidDismiss().then(d => {

if (d && d.data && d.data.r) {

this.loadEntries();

}

});

return await modal.present();

}

async viewImage(e: Entry) {

const modal = await this.modalController.create({

component: ViewImageComponent,

componentProps: { entry: e }

});

modal.onDidDismiss().then(d => { });

return await modal.present();

}

public removeEntry(e) {

this.presentAlertConfirm(e);

}

async presentAlertConfirm(e) {

const alert = await this.alertController.create({

header: 'Confirm!',

message: 'Remove <strong>' + e.title + '</strong>!!!',

buttons: [

{

text: 'Cancel',

role: 'cancel',

cssClass: 'secondary',

handler: blah => {

console.log('Confirm Cancel: blah');

}

},

{

text: 'Okay',

handler: () => {

this.commonS.showLoading();

this.cloud.removeEntry(e).then(() => {

this.commonS.hideLoading();

this.loadEntries();

});

}

}

]

});

await alert.present();

}

}

Antes de continuar es importante señalar que los anteriores componentes deben ser definidos en un módulo del nivel superior para realizar una carga adecuada, en este caso en el archivo /src/app/tab1/tabs1.module.ts:

import { IonicModule } from '@ionic/angular';

import { RouterModule } from '@angular/router';

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { Tab1Page } from './tab1.page';

import { LoginComponent } from './login/login.component';

import { ListImagesComponent } from './list-images/list-images.component';

@NgModule({

imports: [

IonicModule,

CommonModule,

ReactiveFormsModule, // login component

FormsModule,

RouterModule.forChild([{ path: '', component: Tab1Page }])

],

declarations: [Tab1Page, LoginComponent, ListImagesComponent]

})

export class Tab1PageModule {

}

Es importante, como hicimos en el proyecto de Angular proteger las rutas que no deben ser accesibles cuando un usuario no está logueado. Para ello, tenemos en el servicio de autenticación el método canActivate. Para su configuración accedemos al archivo /src/app/tabs/tabs,router.module.ts:

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { TabsPage } from './tabs.page';

import { AuthService } from '../services/auth.service';

const routes: Routes = [

{

path: 'tabs',

component: TabsPage,

children: [

{

path: 'tab1',

children: [

{

path: '',

loadChildren: '../tab1/tab1.module#Tab1PageModule'

}

]

},

{

path: 'tab2',

children: [

{

path: '',

loadChildren: '../tab2/tab2.module#Tab2PageModule',

canActivate: [AuthService]

}

]

},

{

path: 'tab3',

children: [

{

path: '',

loadChildren: '../tab3/tab3.module#Tab3PageModule',

canActivate: [AuthService]

}

]

},

{

path: '',

redirectTo: '/tabs/tab1',

pathMatch: 'full'

}

]

},

{

path: '',

redirectTo: '/tabs/tab1',

pathMatch: 'full'

}

];

@NgModule({

imports: [

RouterModule.forChild(routes)

],

exports: [RouterModule]

})

export class TabsPageRoutingModule {}

Ypara terminar, revisamos el código de los archivos:

app.component.ts

export class AppComponent {

constructor(

private platform: Platform,

private splashScreen: SplashScreen,

private statusBar: StatusBar,

public back: BackbuttonService

) {

console.log('here we are');

this.back.backButton();

this.initializeApp();

}

initializeApp() {

this.platform.ready().then(() => {

this.statusBar.styleDefault();

this.splashScreen.hide();

});

}

}

app.module.ts

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { RouteReuseStrategy } from '@angular/router';

import { IonicModule, IonicRouteStrategy } from '@ionic/angular';

import { SplashScreen } from '@ionic-native/splash-screen/ngx';

import { StatusBar } from '@ionic-native/status-bar/ngx';

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

import { AngularFireAuthModule } from '@angular/fire/auth';

import { AuthService } from './services/auth.service';

import { AngularFireModule } from '@angular/fire';

import { environment } from '../environments/environment';

import { EntriesService } from './services/entries.service';

import { AngularFirestoreModule } from '@angular/fire/firestore';

import { AngularFireStorageModule } from '@angular/fire/storage';

import { GooglePlus } from '@ionic-native/google-plus/ngx';

import { ViewImageComponent } from './customComponents/view-image/view-image.component';

import { CommonsService } from './services/commons.service';

import { BackbuttonService } from './services/backbutton.service';

import { ServiceWorkerModule } from '@angular/service-worker';

@NgModule({

declarations: [AppComponent],

entryComponents: [],

imports: [BrowserModule,

IonicModule.forRoot(),

AppRoutingModule,

AngularFireModule.initializeApp(environment.firebaseConfig),

AngularFireAuthModule, //to use auth

AngularFirestoreModule, //to use firestore

AngularFireStorageModule, ServiceWorkerModule.register('ngsw-worker.js', { enabled: environment.production }), //to use storage

],

providers: [

StatusBar,

SplashScreen,

{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy },

AuthService,

EntriesService,

GooglePlus,

CommonsService,

BackbuttonService

],

bootstrap: [AppComponent]

})

export class AppModule {}

index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8"/>

<title>Ionic App</title>

<base href="/"/>

<meta name="viewport" content="viewport-fit=cover, 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/png" href="assets/icon/favicon.png"/>

<!-- add to homescreen for ios -->

<meta name="apple-mobile-web-app-capable" content="yes"/>

<meta name="apple-mobile-web-app-status-bar-style" content="black"/>

<script src="/assets/leaflet.js"></script>

<script src="/assets/leaflet.markercluster.js"></script>

<link rel="manifest" href="manifest.json">

<meta name="theme-color" content="#1976d2">

</head>

<body>

<app-root></app-root>

<noscript>Please enable JavaScript to continue using this application.</noscript>

</body>

</html>

### Pruebas

Para realizar el despliegue en un servidor de pruebas y poder visualizarlo mediante un navegador podemos ejecutar:

ionic serve

Automáticamente, la aplicación se presentará, por defecto, en la url <http://localhost:8101>. Cada cambio que realicemos en el código forzará una recarga de la web actualizando los cambios.

Es importante, hacer hincapié en que algunos plugin de Capacitor y Cordova que hacen uso de recursos nativos deben ser presentados por el protocolo https. Para ello podemos ejecutar:

ionic serve --ssl

Este comando genera automáticamente los certificados para que la aplicación se presente en la url: <https://localhost:8101>.

Además, si queremos visualizar la aplicación web mediante otros dispositivos móviles que tengamos en la misma red local, deberemos hacer que el servidor de pruebas sea visible desde rutas que no sean locales, para ello el comando completo queda de la siguiente forma:

ionic serve --devapp --ssl

Si visitamos la web desde un navegador podremos visualizar el comportamiento de la PWA y cómo va quedando la interfaz también para la APP. La filosofía de Ionic es escribe una vez y ejecuta en cualquier lado. Veamos las pantallas.

|  |  |
| --- | --- |
| Pantalla de inicio: tab1. Sin inicio de sesión se muestra el componente login. Al intentar acceder a cualquier pestaña nada sucede. | |
|  | |
| Si iniciamos sesión mediante las credenciales de Google se abre un popup que nos redirige al servicio de autenticación de Google. Una vez iniciado sesión seguimos en la pestaña tab1, pero se muestra el componente listaImages. Cada item puede editarse en un modal (editPage) si se desliza hacia la derecha o eliminar si se desliza hacia la izquierda. Además, podemos refrescar toda la lista arrastrándola hacia abajo y observar cómo funciona el infiniteScroll si tenemos más de 10 items.  En la parte superior hay un slider que permite cerrar la sesión. | |
|  |  |
|  |  |
| Si se accede a la pestaña tab2, se solicitan permisos de ubicación y de cámara. Una vez aceptados, se presenta la cámara que permite realizar una foto. Una vez capturada, la app nos presenta como modal el componente editPage para que terminemos de añadir la información de la entrada y la creemos. Si hacemos click en Save veremos una barra de progreso inferior indeterminada y en la parte superior una barra de progreso que indica que el porcentaje de imagen subida a Google Store. Una vez terminado el proceso volvemos a la tab1 donde deberemos refrescar la lista (arrastrando hacia abajo) para ver la nueva entrada. | |
|  |  |
|  |  |
| Si accedemos a la tab3 cargará el mapa leaflet centrado en nuestra posición y mostrará clusterizadas por ubicación las entradas. Si hacemos click en una entrada visualizaremos la imagen en tamaño completo, al igual que si lo hacemos en la lista de entradas. | |
|  |  |
|  |  |

### Generación de la app

Escribir una vez y ejecutar en todos los entornos es un buen lema, pero no es tan directo como parece y siempre necesitaremos ejecutar nuestra aplicación en dispositivos reales o emuladores, ya sea para probar el acceso a recursos nativos o la propia interfaz.

En Capacitor toda la ejecución en nativo se delega de los principales frameworks e IDES de desarrollo, en nuestro caso: Android Studio. Para poder ver nuestra aplicación, tanto en un emulador como en nuestros dispositivos debemos seguir los siguientes pasos.

En cualquier caso, comprobar que Capacitor tiene todos los plugins sincronizados y los archivos actualizados:

npx cap sync

npx cap updates

Se supone que tenemos carpeta android y capacitor está iniciado, ya hemos ejecutado previamente:

npx cap add android

Nota: suponemos que tenemos una carpeta www con los assets que requiere tanto la PWA como la app (para crear la carpeta ioic build).

Copiamos todos los assets a nuestra carpeta android:

npx cap copy android

Por último, abrimos Android Studio con nuestro proyecto:

npx cap open android

Si vamos a probarlo en el dispositivo ejecutamos el proyecto con el símbolo play verde. Si queremos realizar nuestra app (sin firmar) deberemos acceder al menú Build->Build APK.

Para ejecutarlo en emulador abriremos directamente uno desde Android Studio, por ejemplo. Sin embargo, para ejecutarlo en nuestro termina, éste deberá estar conectado al equipo por USB y con las opciones de desarrollador y depuración activadas. Una vez pulsamos play y termina Android Studio de generar la aplicación se instalará automáticamente en nuestro terminal y se abrirá.

|  |  |
| --- | --- |
|  |  |

En la carpeta res que aparece seleccionada en la primera imagen podremos cambiar los iconos y la splash screen de la aplicación. Para más información de este tema <https://capacitor.ionicframework.com/docs/apis/splash-screen/>

Para más información de cómo firmar la apk y subirla a Play Store: <https://support.google.com/googleplay/android-developer/answer/113469?hl=es-419&ref_topic=7072031>

### Despliegue de la PWA

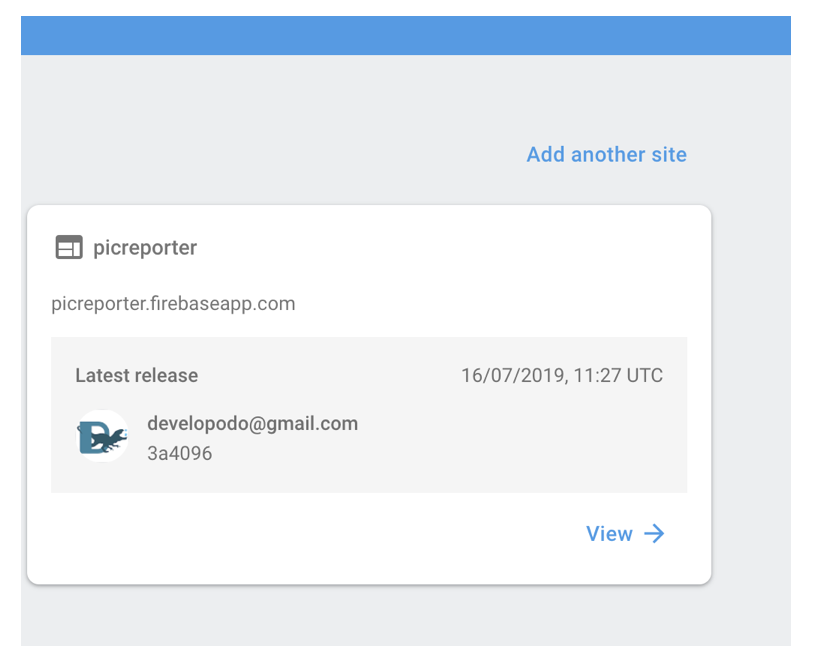
Ya hemos visualizado la PWA anteriormente, pero ahora vamos a subirla a un entorno de producción con todas las capacidades de una PWA. Para ello, seguimos los siguientes pasos:

ng add @angular/pwa

Una vez habilitado los workservices y creado el archivo manifest, creamos la aplicación de producción con el siguiente comando:

ioic build --prod --service-worker

Para desplegarlo en firebase vamos a seguir los mismos pasos que en el proyecto de Angular. Lo único que vamos a añadir en este caso es la peculiaridad de poder subir más de una PWA a un mismo proyecto. Para ello nos vamos a la sección hosting de la consola de firebase. Y añadimos otro sitio, en nuestro caso picrecporter.firebaseapp.com.



Suponemos tener instaladas las herramientas de firebase y el archivo de firebase.json debe quedar así, según nuestro ejemplo (muy importante, el nombre del hosting y la carpeta pública):

{

"hosting": {

"site":"picreporter",

"public": "www",

"ignore": [

"firebase.json",

"\*\*/.\*",

"\*\*/node\_modules/\*\*"

],

"rewrites": [

{

"source": "\*\*",

"destination": "/index.html"

}

]

}

}

Para realizar el despliegue ejecutamos:

firebase deploy --only hosting:picreporter

Y ya tenemos nuestra aplicación PWA en producción.

### Cloud Functions

Una de las funcionalidades más interesantes de firebase es la capacidad de programar funciones que se ejecuten automáticamente, según unos eventos, en el servidor de Google automáticamente.

Esto nos permite desde realizar relaciones entre colecciones para asegurar coherencia en datos (por ejemplo, simular restricciones en claves foráneas), hasta realizar complejos cálculos analíticos que nos permiten crear variables estadísticas en nuestra base de datos en función de las entradas.

Además, existen funciones relacionadas con el almacenamiento que nos permiten moderar automáticamente texto e imágenes. Ésta última se encuentra implementada en el repositorio <https://github.com/firebase/functions-samples/tree/master/moderate-images> y hace uso de Google Cloud Vision API que requiere de la activación de billing. Es una API de inteligencia artificial y, en el momento de la creación de este documento, los costes de uso asociados son:



#### Mejoras del proyecto

Este proyecto, como el resto de este documento, tienen fines académicos, pero en este caso resulta interesante algunas mejoras posibles en el caso de querer realizar una aplicación con características similares.

La primera mejora es la posibilidad mediante Cloud Function de realizar thumbnail de las imágenes que se suben para que se visualicen a tamaño pequeño y no se requiera la carga completa para los avatares. Solo en el caso de visualizarla en tamaño completo se emplearía la url de la imagen original. Un artículo interesante sobre esta funcionalidad puede ser: <https://angularfirebase.com/lessons/image-thumbnail-resizer-cloud-function/>