Contenido

[Instalación 2](#_Toc42960599)

[Servicios 3](#_Toc42960600)

[Android studio 12](#_Toc42960601)

# Instalación

npm install -g @ionic/cli

Ionic start : inicia un proyecto

Ionic serve : ejecuta un servidor de desarrollo

Ionic generate : Herramienta para agregar modulos, componentes, paginas,clases pipes etc…

Empezamos creando una pagina nueva llamada places.

En places.page.ts hacemos un array de objetos simulando la data que llegaría desde la base de datos.

En la plantilla places.page.html mostramos esos datos

<ion-header>

  <ion-toolbar>

    <ion-title>places</ion-title>

  </ion-toolbar>

</ion-header>

<ion-content>

<ion-list>

  <ion-item \*ngFor="let place of places">

    <ion-avatar slot="start">

      <ion-img [src]="place.img"></ion-img>

    </ion-avatar>

    <ion-label>

      {{place.name}}

    </ion-label>

  </ion-item>

</ion-list>

</ion-content>

Como se observa los datos del componente está automáticamente linkado con la vista.

A continuación creamos otra página llamada place-detail dentro de places que mostrará la descripción del lugar.

Para ello tenemos que emplear rutas hijos de una ruta padre. Las rutas hijo serán las ids de los datos ficticios y la ruta padre será el listado mostrado de la página places.

Para ello vamos al enrutador principal y modificamos la sección de la página de places de la siguiente forma: Le añadimos una propiedad children que será un array.

Este array contendrá el enrutado inicial en un objeto, y en otro objeto tendrá la referencia a los hijos. Para la ruta hijo hace falta añadir antes dos puntos :

 {

    path: 'places',

    children: [

      {

        path:'',

        loadChildren: () => import('./places/places.module').then( m => m.PlacesPageModule)

      },

      {

        path:':placeID',

        loadChildren: () => import('./places/place-detail/place-detail.module').then(m=>m.PlaceDetailPageModule)

      }

    ]

  },

Ahora volvemos a la plantilla de places.html y añadimos un routerlink con la id de los hijos

<ion-list>

  <ion-item \*ngFor="let place of places" [routerLink]="['/places',place.id]">

Servicios:

Vamos a crear un servicio que simule las operaciones crud.

Para ello usamos ionic generate. Le damos a servicio y lo movemos dentro de la carpeta places.

Para tener los datos mas accesibles, movemos los datos ficticios de la base de datos de places.page.ts al servicio nuevo.

Como haríamos con la base de datos real vamos a crear un modelo para definir estos datos.

Para ello creamos un nuevo archivo que llamaremos place.model.ts

export interface Place{

    id:string,

    name:string,

    img:string,

    desc:string

}

Y importamos el modelo en el servicio.

import { Place } from "./place.model";

Arreglamos los datos para asignarle el nuevo modelo :

 private places:Place[] =[

    {

      id:'1',

Hacemos las funciones de crud:

  getPlaces()

  {

    return [this.places]

  };

  getPlace(placeID:string)

  {

    return this.places.find(place=>{return place.id === placeID})

  };

  newPlace(name:string,img:string)

  {

    this.places.push({

      id: (this.places.length +2).toString(),

      name:name,

      img:img,

      desc:''

    })

  };

  deletePlace(placeID:string)

  {

    this.places = this.places.filter(place => {return place.id !== placeID})

  };

Ahora tenemos que importar al archivo places.page.ts el servicio (para que lo coja desde la plantilla) y lo instanciamos en el constructor.

Una vez hecho esto la plantilla necesita datos llamados places. Para ello creamos un array vacío llamado places.

En el ngOnInit (función que se ejecuta al cargar el componente) añadiremos la función para que cargue todo los lugares getPlaces().

Una vez hecho tenemos que escribir el código que mostrará la descripción de su lugar correspondiente, nos vamos a place-detail.page.ts e importamos el router, que es el componente de angular que nos ayudará a capturar la ruta (con la id actual) para pintar el dato correspondiente en la plantilla.

Lo importamos y lo instanciamos

import { ActivatedRoute } from "@angular/router";

  constructor(private activatedRoute:ActivatedRoute) { }

Importamos el servicio creado anteriormente y la clase modelo

import { PlacesService } from '../places.service';

import { Place } from '../place.model';

los instanciamos

 place:Place; //declaro variable place con clase Place

  constructor(private activatedRoute:ActivatedRoute, private placeService:PlacesService) { }

En la función de inicio (ngOnInit) usamos una propiedad del modulo enrutador para que nos devuelva el número de ID que aparece en la ruta del navegador

  ngOnInit() {

    this.activatedRoute.paramMap.subscribe(

      paramMap => {

        const recipeID = paramMap.get('placeID') // el mismo que está en app-routing module

        this.place = this.placeService.getPlace(recipeID) // añado a la variable place los datos obtenidos por la funcion getplace (datos de un unico lugar)

      }

    )

  }

Vamos ahora a la plantilla html de place-detail.

<ion-header>

  <ion-toolbar>

    <ion-back-button slot="start" defaultHref="/places"></ion-back-button>

    <ion-title>Maravilla: {{place.name}}</ion-title>

  </ion-toolbar>

</ion-header>

<ion-content>

<ion-grid class="ion-no-padding">

  <ion-row class="ion-no-padding">

    <ion-col class="ion-no-padding">

      <ion-img [src]="place.img">

      </ion-img>

    </ion-col>

  </ion-row>

  <ion-row class="ion-text-center">

    <ion-col>

      <h1>{{place.name}}</h1>

    </ion-col>

  </ion-row>

  <ion-row>

    <ion-col class="ion-padding">

      {{place.desc}}

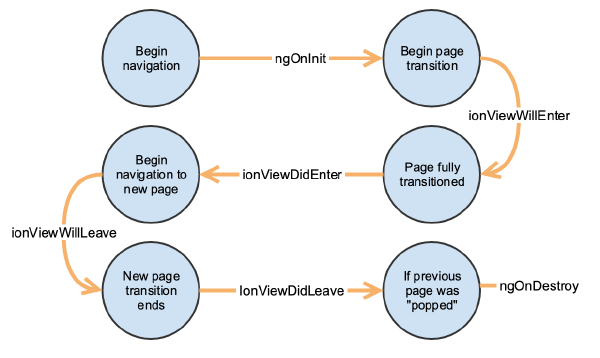
    </ion-col>

  </ion-row>

</ion-grid>

</ion-content>

ngOnInit solo se ejecuta al cargar el componente, si queremos un evento cada vez que entra en un componente (vista) debemos usar otro como por ejemplo



Para gestionar alerts, hay que importar un modulo de Ionic llamado alertcontroller

Para configurarlo:

async presentAlert() {

const alert = await this.alertController.create({

cssClass: 'my-custom-class',

header: 'Alert',

subHeader: 'Subtitle',

message: 'This is an alert message.',

buttons: ['OK',{text:’cancel’,role:’cancel’]

handler: () => {this.router.navigate([‘/places’])

});

Para usarlo :

presentAlert.present();

Creamos una página más con la función de añadir lugares.

Para ello ionic generate > page > places/place-add

Esta página funcionará con la ruta newplace por ejemplo, nos vamos al enrutador principal (app-routing.module.ts y añadimos la nueva ruta.

Vamos a crear un botón en la pagina principal que nos lleve a esta nueva ruta.

Para ello en la plantilla html de la página principal places.page.html

<ion-header>

  <ion-toolbar>

      <ion-back-button slot="start" defaultHref="/places"></ion-back-button>

    <ion-title>Añadir nueva maravilla</ion-title>

  </ion-toolbar>

</ion-header>

<ion-content>

<ion-row>

  <ion-col>

    <ion-card>

      <ion-card-content>

        <ion-input type="text" placeholder="Escribe un titulo" #title>

        <ion-input type="text" placeholder="http://img-website.com" #imgURL>

          <ion-input type="text" placeholder="Escribe una descripción"#description></ion-input>

          <ion-button (click)="saveNewPlace(title,imgURL,description)">

            Guardar

          </ion-button>

        </ion-input>

      </ion-card-content>

    </ion-card>

  </ion-col>

</ion-row>

</ion-content>

Hemos creado unos inputs y una función que recoge los datos con un botón.

En el componente creamos la función que le pase estos dos valores al servicio y este creara un nuevo dato.

En place-add.page.ts

import { PlacesService } from "../places.service";

Importamos el servicio de places.service.ts en el constructor:

  constructor(private placeService:PlacesService) { }

Creamos la función

  saveNewPlace(title,imgURL,description)

  {

    const titulo = title.value;

    const img = imgURL.value;

    const desc = description.value;

    this.placeService.newPlace(titulo,img,desc);

    this.router.navigate(['/places'])

  }

Y en la penúltima línea ejecutamos la función del servicio y en la ultima lo redireccionamos a la pagina principal

La función del servicio que llamamos de Places.service.ts:

  newPlace(name:string,img:string,desc:string)

  {

    this.places.push({

      id: (this.places.length +2).toString(),

      name:name,

      img:img,

      desc:desc

    })

  };

Esta función del servicio añadirá al array places los datos que le llegan de la función saveNewPlace del archivo place-add.page.ts. que serán los datos introducidos en el input.

Vamos a simular una petición de datos http. Para ello emplearemos la herramienta online jsonplaceholder que simula una bd.

Reutilizaremos el componente de home para esta tarea.

Lo primero es hacer una ruta hacia el componente, crearemos un nuevo botón en la vista de places.

Colocaremos dentro del toolbar de la plantilla places.page.html

  <ion-buttons slot="end">

    <ion-button (click)="goHome()" >

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

    </ion-button>

  </ion-buttons>

Y en la plantilla [home.page.html](http://home.page.html) un botón de vuelta

Ahora podemos trabajar con la página.

Creamos un servicio dentro del componente home, que nos servirá para conexión http.

Ionic generate > servicio

Lo llamaremos photos y lo crearemos dentro de la carpeta home > home/photos

En el servicio creado importamos el modulo httpclient, este modulo permite hacer peticiones típicas http(get,post,put,delete). Lo importamos y declaramos en el constructor :

import { HttpClient } from "@angular/common/http";

…

constructor(

    private http:HttpClient

  ) { }

OJO: este modulo hay que importarlo tambien en app.modules.ts (archivo de modulos principal de la aplicación) para que funcione.

import { HttpClientModule } from "@angular/common/http";

Y lo ponemos en los imports:

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

Es importante que sea el HttpClientModule.

Volvemos al servicio photos.

Ahora hacemos una función para traer los datos

  getphotos(){

    return this.http.get<any>('https://jsonplaceholder.typicode.com/photos?\_limit=15') //?\_limit=15  limita la cantidad de datos a 15.

  }

}

OJO: <any> lo usamos porque no tenemos definido ningún modelo de datos. Deberiamos crear un modelo nuevo con estos datos…

Ahora el servicio que hemos hecho debemos instanciarlo en el componente home, para ello vamos a [home.page.ts](http://home.page.ts) (import + constructor)

import { PhotosService } from './photos.service';

…

 private photosService:PhotosService,

Ahora en ngOnInit Mostramos estos datos

export class HomePage {

  photos = {};

  constructor(

    private photosService:PhotosService,

  ) {}

    ngOnInit()

    {

     this.photosService.getphotos().subscribe(data => this.photos = data)

    }

Ahora nos vamos a la plantilla de home y mostramos los datos:

<ion-content>

<ion-list>

  <ion-card \*ngFor="let photo of photos">

    <img [src]="photo.thumbnailUrl">

    <ion-card-content>

      <p>{{photo.title}}</p>

      <p>{{photo.albumId}}</p>

    </ion-card-content>

  </ion-card>

</ion-list>

</ion-content>

# Android studio

Ahora vamos a la parte de producción. Para ello necesitamos una herramienta llamada Android studio y capacitor.

Una vez instalado Android studio hay que ver la API versión para que concuerde con la de capacitor.

Para añadir capacitor, al crear el proyecto ionic nos indico si queríamos instalarlo y le dijimos que si.

Para ejecutar capcitor, escribimos npx cap init en la consola de proyecto.

Le damos un nombre y una id. pj nombre : ionic , id : com.ionics.app

Y nos creara un archivo llamado capacitor.config.json

Ahora construimos el proyecto typamos en consola : ionic build

Esto construye la carpeta www. Que usara capacitor

Ahora escribimos npx cap add Android (si tenemos mac tb podemos add IOS)

Para generar el proyecto de Android, que contendrá la aplicación móvil.

Para abrirlo en Android studio usamos el comando npx cap open Android

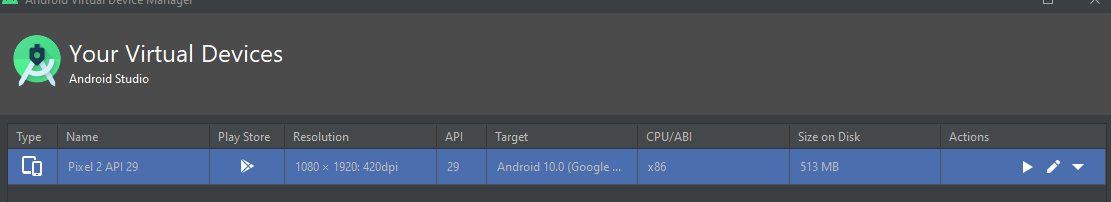
Una vez hecho esto nos dará errores, seleccionamos uno de dependencias y las instalamos haciendo click en el link.

Una vez este succesfull, le damos al botón de arriba a la derecha que dice AVD MANAGER para seleccionar un dispositivo con la que probar la app.

Una vez seleccionado, nos pedirá elegir una versión de Android, la primera vez tendremos que darle a download y esperar.

Para que funcione debemos activar en la BIOS la virtualización VTx. (Windows)

Le damos a AVD manager. Seleccionamos la configuración del dispositivo que queramos y le damos a play.



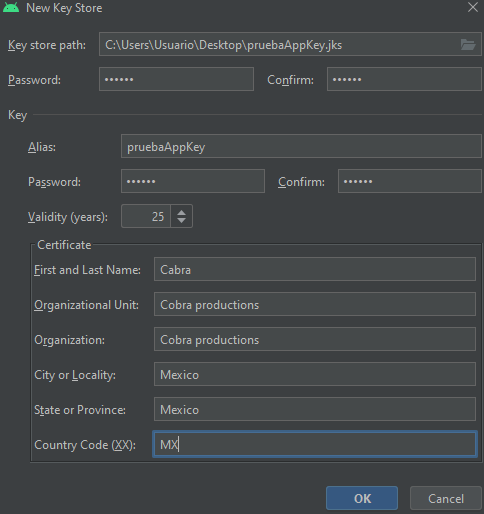
Una vez abierto el emulador le tenemos que cargar la aplicación para ello en Android studio le damos a run app (icono play)(mayus+f10)

Para generar el apk le damos a build > generate single bundle or apk

En el menú desplegado le damos a apk.

En la siguiente ventana nos pide un key store path. Si no tenemos ninguno le damos a créate

En la nueva ventana localizamos en el primer input donde ira el proyecto y el nombre del mismo.



Este cuadro es para firmar la aplicación.

Al darle a OK y siguiente nos creara el archivo en la ubicación, el archivo configurado es como un certificado digital de nuestra app.

En la siguiente ventana seleccionamos reléase o debug y V2 y le damos a finish

Una vez terminado le damos a locate y buscamos la carpeta app y la carpeta debug/reléase, dentro estará nuestro archivo apk.