PMDM04.- Utilización de librerías multimedia integradas.

1.- INTRODUCCIÓN.

En esta unidad veremos cómo se puede añadir contenido multimedia a nuestras aplicaciones. Lo primero que debemos tener en cuenta es que añadir imágenes o videos será tan fácil como en una página HTML. Por lo tanto, lo primero que vamos a hacer es un pequeño recordatorio de cómo se utilizan vídeos e imágenes en HTML.

La etiqueta img tiene como mínimo dos atributos que deben tenerse en cuenta: src y alt. Haciendo uso de src indicamos la ruta a la imagen que queremos mostrar y haciendo uso de alt indicamos es texto alternativo que debería mostrarse en caso de que no se muestre la imagen.

```
<img src="assets/imgs/logo.png" alt="">
```

Si os fijáis las imágenes se almacenan en la carpeta assets/img que se encuentra en la carpeta src.

Por lo que se refiere a los vídeos, no debéis olvidaros del atributo controls si queréis que el usuario pueda tener acceso a los controles del reproductor. Para indicar las fuentes de los vídeos se debe utilizar el elemento source. Este elemento tiene dos atributos imprescindibles: src y type. Se pueden indicar tantos sources como se quiera. Se intentará reproducir en el orden en el que se encuentran.

```
<video width="80%" controls>
    <source src="assets/imgs/video.mp4" type="video/mp4">
</video>
```

Es cierto, que las imágenes y los vídeos pueden hacer que nuestras aplicaciones sean más atractivas, pero habrá casos en los que necesitaremos sacar fotos, utilizar un mapa, incluir gráficos o utilizar el reproductor de vídeo nativo del dispositivo. Mucha de esta funcionalidad es nativa y por ello necesitaremos un

emulador o dispositivo real para poder realizar las pruebas. En nuestro caso utilizaremos Android Studio para emular un dispositivo Android. Aunque en la documentación de cordova indica un gran número de pasos a seguir para poder emular un dispositivo Android, Android Studio se encargará de instalar cada uno los requisitos que no estén instalados.

Tal y como ya se ha mencionado antes, Ionic provee acceso a la funcionalidad nativa a través de Promises u Observables. Es decir, realizaremos una petición y en función de si el retorno es una Promise o un Observable encadenaremos un then() y un catch() en el primer caso o indicaremos la función de CallBack de éxito y de error en segundo caso a modo de parámetros.

2.- ACCESO A LA CÁMARA.

Poder acceder a la funcionalidad nativa del dispositivo haciendo uso del wrapper que no facilita Ionic es una tarea sencilla. A pesar de ello, se deben tener en cuenta ciertos aspectos como el formato de entrada, la calidad de la foto, el formato de salido, etc. A continuación, vamos a ver cómo utilizar la librería.

2.1.- INSTALACIÓN.

Para utilizar la camara utilizaremos el Plugins Camera de Capacitor. Una de las ventajas de utilizar el plugin de Capacitor es poder utilizar a cámara en el navegador. Para ello tendremos que instalar el plugin PWA Elements:

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

Modificamos el fichero main.ts como se muestra a continuación:

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

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
```

```
import { AppModule } from './app/app.module';

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

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



if (environment.production) {

  enableProdMode();

}



platformBrowserDynamic().bootstrapModule(AppModule)

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

  defineCustomElements(window);
```

2.2.- CAMERAOPTIONS.

Antes de llamar a la función que se encarga de gestionar la llamada a la funcionalidad nativa se deben configurar las opciones de la cámara. Estas opciones son dependientes del plugin de cordova ya que lo único que hace Ionic es ofrecernos un wraper. Por lo tanto, si se desea consultar la lista completa de opciones se debe consultar la documentación de cordova. Entre las opciones más importantes se encuentran las siguientes:

quality	(numérico) Parámetro que permite establecer la calidad de la fotografía. Rango de valores 0-100
resultType	(numérico) Parámetro que permite establecer el formato de retorno. Se dispone del enum CameraResultType para fijar sus valores.
allowEditing	(booleano) Parámetro para fijar si se permite editar la foto o no.
shaveToGallery	(booleano) Parámetro que permite indicar si se quiere se guarda la imagen en el álbum del dispositivo
preserveAspectRatio	(booleano) Parámetro que permite indicar si se quiere mantener el ratio ancho-alto de la imagen.

Se debe prestar especial atención al parámetro destinationType ya que en caso de utilizar CameraResultType.dataUrl pueden surgir problemas de memoria porque la promise va a retornar una cadena de caracteres que contiene la imagen codificada.

2.3.- CAMERA.

El wrapper camera cuenta con la función **getPhoto(options)** que nos da la posibilidad de sacar una foto o cargar una imagen de la librería del dispositivo. La función retorna una **Promise<CameraPhoto>**. Por lo tanto, encadenaremos una función de callback para el caso de éxito utilizando **getPhoto(options).then()** y una función de callback para el caso de error utilizando **getPhoto(options).catch().** Otra opción sera utilizar **await** en la llamana a **getPhoto() .**

2.4.- EJEMPLO.

En el siguiente ejemplo se muestra como capturar una imagen con y enlazarla a la propiedad src de una imagen. Fijaros que para evitar que se cargue una imagen vacía se incluye la directiva ngIf. Además se ha utilizado DomSanitizer para que no haya problemas con la url de la imagen.

Fichero html:

```
<ion-header [translucent]="true">
```

```
    <ion-toolbar>
        <ion-title>
            Ejemplo camara
        </ion-title>
    </ion-toolbar>
</ion-header>


<ion-content [fullscreen]="true">
    <img [src]="domSanitizer.bypassSecurityTrustUrl(imagen)" *ngIf="imagen" width="100%" height="100%">
    <ion-button expand="block " (click)="sacarFoto() ">Sacar foto</ion-button>
</ion-content>
```

Fichero ts:

```
import { Component } from '@angular/core';
import { Plugins,CameraResultType} from '@capacitor/core';
import { DomSanitizer } from '@angular/platform-browser';
const {Camera} = Plugins;
```

```
@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
})
export class HomePage {

  public imagen;
  constructor(public domSanitizer:DomSanitizer) {}
  sacarFoto() {

    const image = Camera.getPhoto({
      quality: 90,
      allowEditing: true,
      resultType: CameraResultType.Uri
    }).then((image)=>{
```

```
      var imageUrl = image.webPath;

      // se asigna la url a la variable que se enlaza desde la vista

      this.imagen = imageUrl;

    })

  }


}
```

3.- VIDEOPLAYER.

Hay veces que utilizar la etiqueta video de HTML no cumple con los requisitos de la aplicación. En esos casos se puede utilizar el wrapper que nos ofrece Ionic para acceder a la funcionalidad nativa del reproductor del dispositivo. El reproductor nos va a permitir reproducir en pantalla completa un vídeo. La función retorna una **Promise<any>**.

Por lo tanto, encadenaremos una función de callback para el caso de éxito utilizando **getPicture(options).then()** y una función de callback para el caso de error utilizando **getPicture(options).catch()**

3.1.- INSTALACIÓN.

En caso de no haber realizado la instalación del paquete de ionic-native indicado anteriormente, esta será la primera instalación que llevaremos a cabo. El resto de las instalaciones que llevaremos a cabo son las siguientes: paquete de cordova y wrapper de Ionic.

```
inpm install capacitor-video-player
npx cap sync
```

Para realizar las pruebas en un navegador no hace falta realizar ninguna instalación. Si queremos emular o instalar el proyecto en dispositivo Android tendremos que modificiar el archivo android\app\src\main\java\io\ionic\starter\MainActivity.java e incluir lo siguiente:

```
package io.ionic.starter;



import android.os.Bundle;

import com.getcapacitor.BridgeActivity;
import com.getcapacitor.Plugin;

import java.util.ArrayList;
import com.jeep.plugin.capacitor.CapacitorVideoPlayer;



public class MainActivity extends BridgeActivity {

  @Override

  public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);



    // Initializes the Bridge
```

```
    this.init(savedInstanceState, new ArrayList<Class<? extends Plugin>>() {{
      // Additional plugins you've installed go here
      // Ex: add(TotallyAwesomePlugin.class);
      add(CapacitorVideoPlayer.class);
    }});
  }
}
```

3.4.- EJEMPLO.

En el siguiente ejemplo vamos a ver cómo utilizar el wrapper que nos acceso al reproductor del dispositivo:

```
import { Plugins } from '@capacitor/core';
import * as WebVPPlugin from 'capacitor-video-player';
```

```
const { CapacitorVideoPlayer, Toast } = Plugins;

...



async ngOnInit() {



    this._testApi = this.testApi ? this.testApi : false;

    if (this.platform === "ios" || this.platform === "android") {

      this._videoPlayer = CapacitorVideoPlayer;



    } else {

      this._videoPlayer = WebVPPlugin.CapacitorVideoPlayer;

    }

    this._url = this.url

    this._addListenersToPlayerPlugin();

  }
```

```
async ionViewDidEnter() {
    // comprobaciones previas
    if(this._url != null) {
           // inicializar el reproducto
           const res:any = await this._videoPlayer.initPlayer({mode:"fullscreen",url:this._url,playerId:"fullscreen"
    // mostrar error en ios y android
    if(!res.result && (this.platform === "ios" || this.platform === "android")) {
        await Toast.show({
        text: res.message,
        duration: 'long',
        position: 'bottom'
        });
    }
    console.log("res.result ",res.result) ;
    if(!res.result) console.log("res.message",res.message);
```

```
}
```

Los aspectos a tener en cuenta en el ejemplo son:

- Importas las librerías necesarias (rojo)
- Asignar el reproductor en función de la plataforma (verde)
- Inicializamos el reproductor (azul)

4.- NG2-CHARTS.

Para aquellas aplicaciones en las que los usuarios deban tratar con datos es importante que la visualización de los datos sea adecuada y facilite a los usuarios la comprensión de estos. Los dispositivos móviles cuentan con pantallas de tamaños reducidos en algunos casos. Por lo tanto, si se quiere presentar la información de una tabla de datos a los usuarios una tabla puede que no sea la forma más adecuada. En cambio, si utilizamos un gráfico que ayude en la visualización a aquellos usuarios que estén utilizando un dispositivo con pantalla de tamaño reducido, la información podrá visualizarse de una forma efectiva.

La librería va a construir gráficos a partir de ciertos datos que se van a encontrar en los controladores o en los servicios. Los nombres de los atributos que utilicemos en las vistas deben ser los que se indican en la documentación de la librería ya que si no fuera así los enlazados no funcionarían.

4.1.- INSTALACIONES.

Para poder utilizar ng2-charts-x tendremos que utilizar los siguientes comandos:

```
npm install --save ng2-charts
npm install --save chart.js
```

Una vez las instalaciones añadiremos la librería al fichero xxxx.module.ts:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { IonicModule } from '@ionic/angular';
import { FormsModule } from '@angular/forms';
import { HomePage } from './home.page';

import { HomePageRoutingModule } from './home-routing.module';
import {ChartsModule} from 'ng2-charts';

@NgModule({
  imports: [
    ChartsModule,
    CommonModule,
    FormsModule,
    IonicModule,
    HomePageRoutingModule
```

```
  ],
  declarations: [HomePage]
})
export class HomePageModule {}
```

4.2.- ATRIBUTOS/PROPIEDADES DE LOS GRÁFICOS

4.2.- ATRIBUTOS/PROPIEDADES DE LOS GRÁFICOS

Entre las propiedades de los gráficos se encuentra las siguientes propiedades:

data	Propiedad que enlazaremos con el conjunto de puntos que queremos visualizar. En función del tipo grafico la estructura que contiene los datos será diferente.
datasets	Propiedad que enlazaremos con el conjunto de puntos que queremos visualizar y sus etiquetas en el gráfico.
labels	Propiedad que enlazaremos los con etiquetas que queremos que se muestren en el eje x
chartType	Propiedad que enlazaremos con el tipo de gráfico que queremos visualizar. Los valores validos son los siguientes: line, bar, radar, pie, polarArea, doughnut
options	Propiedad que enlazaremos con las opciones que queremos establecer al gráfico. Para la consulta de las opciones se puede consultar el siguiente enlace: Chart.js
colors	Propiedad que enlazaremos con los colores que queremos que se utilicen para la visualización. Si no se indica, se

	utilizan los colores aleatorios.
legend	Propiedad que enlazaremos con la variable que indica si queremos que visualiza la legenda.

4.3.- EJEMPLO.

En el siguiente ejemplo se muestra cómo utilizar un gráfico. Para ello, además de la página se ha utilizado un servicio para que el controlador de la vista no contenga código que no se encargue de capturar eventos generados en la vista.

En la vista necesitaremos un canvas o lienzo para poder insertar el gráfico. Los atributos del canvas son los que se mencionaron en el punto anterior.

```
<ion-header>
    <ion-toolbar>
        <ion-title>
            ng2-charts
        </ion-title>
    </ion-toolbar>
</ion-header>


<ion-content padding>
    <div class="row">
```

```
        <div class="col-md-6">
            <div style="display: block;">
                <canvas baseChart width="300" height="400"
                  [datasets]="servicio.lineChartData"
                  [labels]="servicio.lineChartLabels"
                  [options]="servicio.lineChartOptions"
                  [colors]="servicio.lineChartColors"
                  [legend]="servicio.lineChartLegend"
                  [chartType]="servicio.lineChartType"
                    (chartHover)="chartHovered($event)" (chartClick)="chartClicked($event)"></canvas>
            </div>
        </div>
    </div>
    <ion-button expand="block" (click)="anadirDatos()">Añadir datos</ion-button>
</ion-content>
```

En el controlador únicamente se incluye la captura de los eventos que ocurren en la vista. Por un lado, está la captura de evento generados en el gráfico. Por otro lado, está la captura del evento generado en el botón que sirve para añadir datos.

```
import { Component} from "@angular/core";
import { ServicioGraficosService } from '../servicio-graficos.service';


@Component({
  selector: "app-home",
  templateUrl: "home.page.html",
  styleUrls: ["home.page.scss"]
})
export class HomePage{



  constructor(public servicio:ServicioGraficosService) {}
  public chartClicked(e:any):void {
    console.log(e);
  }


  public chartHovered(e:any):void {
```

```
    console.log(e);
  }
  public anadirDatos()
  {
    this.servicio.anadirDatos();
  }
```

En el servicio se encuentran todas las propiedades que se enlazan con la vista y dos métodos que sirven como ejemplo para ver como modificar los datos que se están visualizando. Para evitar problemas con los enlazados se deben utilizar las instancias mutables con operaciones que hagan que funcionen como si fueran inmutables.

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


@Injectable({
  providedIn: 'root'
})
export class ServicioGraficosService {


  constructor() { }
```

```
public lineChartData:Array<any> = [
  {data: [65, 59, 80, 81, 56, 55, 40], label: 'Series A'},
  {data: [28, 48, 40, 19, 86, 27, 90], label: 'Series B'},
  {data: [18, 48, 77, 9, 100, 27, 40], label: 'Series C'}
];
public lineChartLabels:Array<any> = ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 'Julio'];
public lineChartOptions:any = {
  responsive: true
};
public lineChartColors:Array<any> = [
  { // grey
    backgroundColor: 'rgba(148,159,177,0.2)',
    borderColor: 'rgba(148,159,177,1)',
    pointBackgroundColor: 'rgba(148,159,177,1)',
    pointBorderColor: '#fff',
    pointHoverBackgroundColor: '#fff',
    pointHoverBorderColor: 'rgba(148,159,177,0.8)'
  },
```

```
  { // dark grey
    backgroundColor: 'rgba(77,83,96,0.2)',
    borderColor: 'rgba(77,83,96,1)',
    pointBackgroundColor: 'rgba(77,83,96,1)',
    pointBorderColor: '#fff',
    pointHoverBackgroundColor: '#fff',
    pointHoverBorderColor: 'rgba(77,83,96,1)'
  },
  { // grey
    backgroundColor: 'rgba(148,159,177,0.2)',
    borderColor: 'rgba(148,159,177,1)',
    pointBackgroundColor: 'rgba(148,159,177,1)',
    pointBorderColor: '#fff',
    pointHoverBackgroundColor: '#fff',
    pointHoverBorderColor: 'rgba(148,159,177,0.8)'
  }
];
```

```
public lineChartLegend:boolean = true;

public lineChartType:string = 'line';



public randomize():void {

  let _lineChartData:Array<any> = new Array(this.lineChartData.length);

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

    _lineChartData[i] = {data: new Array(this.lineChartData[i].data.length), label: this.lineChartData[i].label};

    for (let j = 0; j < this.lineChartData[i].data.length; j++) {

      _lineChartData[i].data[j] = Math.floor((Math.random() * 100) + 1);

    }

  }

  this.lineChartData = _lineChartData;

}

public anadirDatos() {

  this.lineChartLabels = [...this.lineChartLabels,'Agosto'];

  this.lineChartData[0]['data'].push(200);

  this.lineChartData[1]['data'].push(200);

  this.lineChartData[2]['data'].push(200);
```

```
  }
}
```

5.- ANDROID STUDIO

Para terminar con esta unidad veremos cómo se emular un proyecto haciendo uso de Android Studio. Los pasos a seguir serán los siguientes:

- Instalaciones cd @capacitor/core y @capacitor/cli
```
npm install --save @capacitor/core @capacitor/cli
```
- Inicialización de Capacitor en el proyecto
```
npx cap init
```
- Creación de la carpeta www dentro del proyecto
- Creación del fichero index.html
- Añadir la plataforma Android
```
npx cap add android
```
- Build
```
ionic capacitor build android
```
- Apertura del proyecto
```
npx cap open android
```

Para actualizar el proyecto en Android Studio tenemos las siguientes opciones:

- Actualizar el código
```
npx cap copy
```
- Actualizar los plugins
```
npx cap update
```
- Actualizar el códgio y los plugins
```
npx cap sync
```