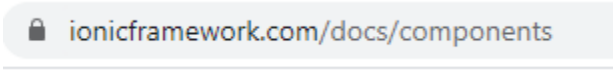
# TEMA III

## COMPONENTES IONIC Y SU USO EN APLICACIONES

### 3.2. Componentes

Las aplicaciones ionic están hechas de bloques de construcción llamados Componentes, que nos permiten construir rápidamente la interfaz de usuario para nuestra aplicación.
La API de ionic nos provee de muchos componentes y subcomponentes que podemos utilizar sin necesidad de implementar todos nuevos.




Todos los componentes, tal y como podemos ver en la documentación tienen propiedades que definirán tanto su uso como su aspecto y propiedades personalizadas css. Además podremos recurrir al css tradicional para cambiar el aspecto de dichos componentes.

Veamos algunos de ellos que nos serán muy útiles

### Componente ionic-toolbar

Es una barra de herramientas general que se puede utilizar en una aplicación como un encabezado, un subencabezado, un pie de página. No importa cuántas barras de herramientas pongamos, se mostrarán correctamente y el contenido se ajustará.

Se colocan encima o debajo del contenido. Cuando la barra de herramientas esté ubicada en **<ion-header>**, aparecerá fija en la parte superior del contenido, y cuando esté en **<ion-footer>** aparecerá establecida en la parte inferior.

Cuando se encuentra dentro de **<ion-content>** , las barras de herramientas se desplazarán con el contenido.

Dentro un una toolbar podemos usar los siguientes componentes:

**Botones:** se pueden incluir en las barras de herramientas de encabezado y pie de página. Para agregar un botón en una barra de herramientas, primero se agregar un elemento **<ion-buttons>** a modo de contenedor de todos los botones.

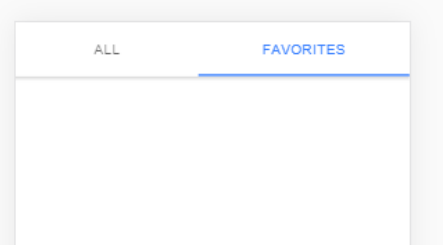
> Una propiedad importante de los botones es **slot**, la usaremos para posicionarlo dentro de la barra. Por ejemplo slot="start" indica que el botón se coloca a la izquierda, al principio de la barra

| Slot | Description |
|---|---|
| `secondary` | Positions element to the `left` of the content in `ios` mode, and directly to the `right` in `md` mode. |
| `primary` | Positions element to the `right` of the content in `ios` mode, and to the far `right` in `md` mode. |
| `start` | Positions to the `left` of the content in LTR, and to the `right` in RTL. |
| `end` | Positions to the `right` of the content in LTR, and to the `left` in RTL. |

**Título**: <**Ion-title**> es el componente que establece el nombre del título de la barra de herramientas.

> **Botón de retroceso**: **<ion-back-button>** se utiliza para crear un botón de retroceso, que vuelve al registro de la aplicación cuando se hace clic en él. Es lo "suficientemente inteligente" como para saber qué ofrecer, según el modo y cuándo mostrar de acuerdo con la pila de navegación.

**Segmentos:** son la mejor manera de permitir al usuario cambiar entre diferentes conjuntos de datos.




**Barra de búsqueda:** para incluir una barra de búsqueda dentro de la barra de herramientas.

**Menús, Varias colecciones de ion-buttons…..**

## Componente ionic-list

Es un componente muy habitual en las app. Típica lista con filas que pueden contener icono, texto, botones,…

Las listas admiten varias interacciones, incluido deslizar elementos para revelar opciones, arrastrar para reordenar elementos dentro de la lista y eliminar elementos.

```
<ion-list>
        <ion-item></ion-item>
        ……..
</ion-list>
```

Tiene que ir dentro de la sección <ion-contect> de la página.

**Ejemplo_3** : Sobre el ejemplo anterior, cambiaremos los botones por una lista

En el archivo de la clase del componente inicio:

- Declaramos una interfaz para definir el tipo de datos del <ion-item>

- Declaramos un array de elementos del tipo definido por la interfaz, de este array sacaremos los elementos para automatizar la generación de los <ion-item>

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

interface ItemLista{
  icon: string;
  name:string;
  redirectTo: string;
}

@Component({
  selector: 'app-inicio',
  templateUrl: './inicio.page.html',
  styleUrls: ['./inicio.page.scss'],
})
export class InicioPage implements OnInit {

 ElementosLista: ItemLista[]=[
   {
    icon:'american-football-outline',
    name: 'Action Sheet',
    redirectTo: '/action'
   },
   {
     icon:'',
     name: 'Alert',
     redirectTo:'/alert'
   }
   ] ;
  constructor() { }

  ngOnInit() {
  }

}
```

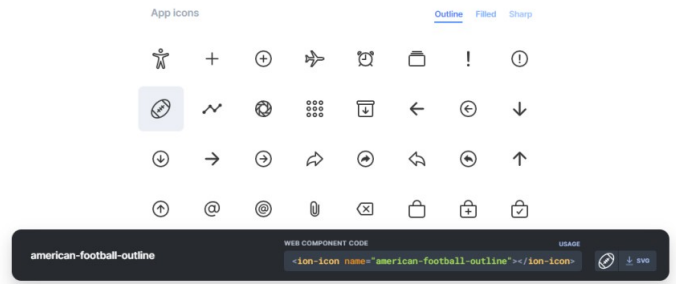
En el template del componente inicio:

- Al incluir el <ion-list> automáticamente se nos genera el código del componente, añadimos nuestras variables al *ngFor

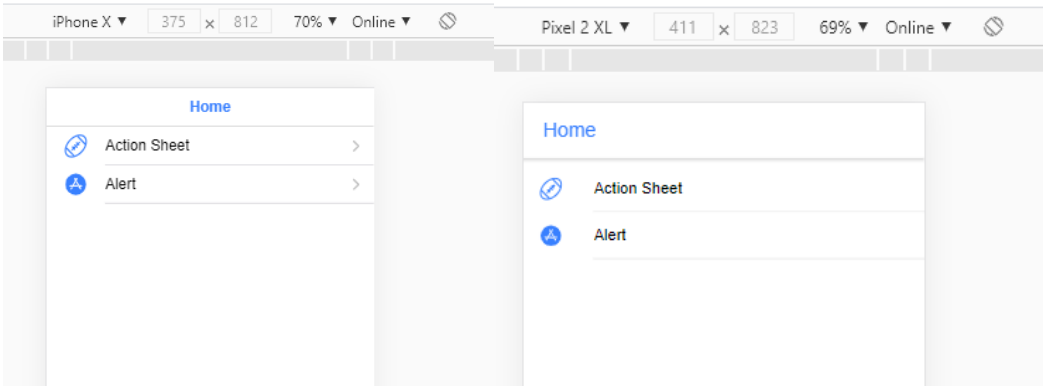
```
<ion-header>
  <ion-toolbar>
    <ion-title color="primary">Home</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <ion-list >
    <ion-item *ngFor="let li of ElementosLista"  [routerLink]="li.redirectTo">
      <ion-icon [name]="li.icon" slot="start" color="primary"></ion-icon>
      {{li.name}}
    </ion-item>
  </ion-list>
</ion-content>
```

Para añadir los iconos: En la documentación, en la parte izquierda de la pantalla tenemos todos los componentes que podemos usar, buscamos la opción iconos


Ionic nos muestra iconos disponibles para nuestras aplicaciones. Si seleccionamos uno de ellos, se nos nuestra el código que debemos incluir para que se vea




La ejecución quedaría




Si queremos que aparezca la flecha de la derecha también en android añadimos el atributo detail al <ion-item>

Si queremos eliminar las líneas entre elementos utilizamos el atributo lines(none)

En la documentación podemos consultar todos los atributos

## Componente action-sheet

Es como un menú emergente. Veamos en la documentación qué es y cómo se muestra

**Ejemplo_4:** Añadimos este componente a nuestro Ejemplo_3

1. En el template de la página action:

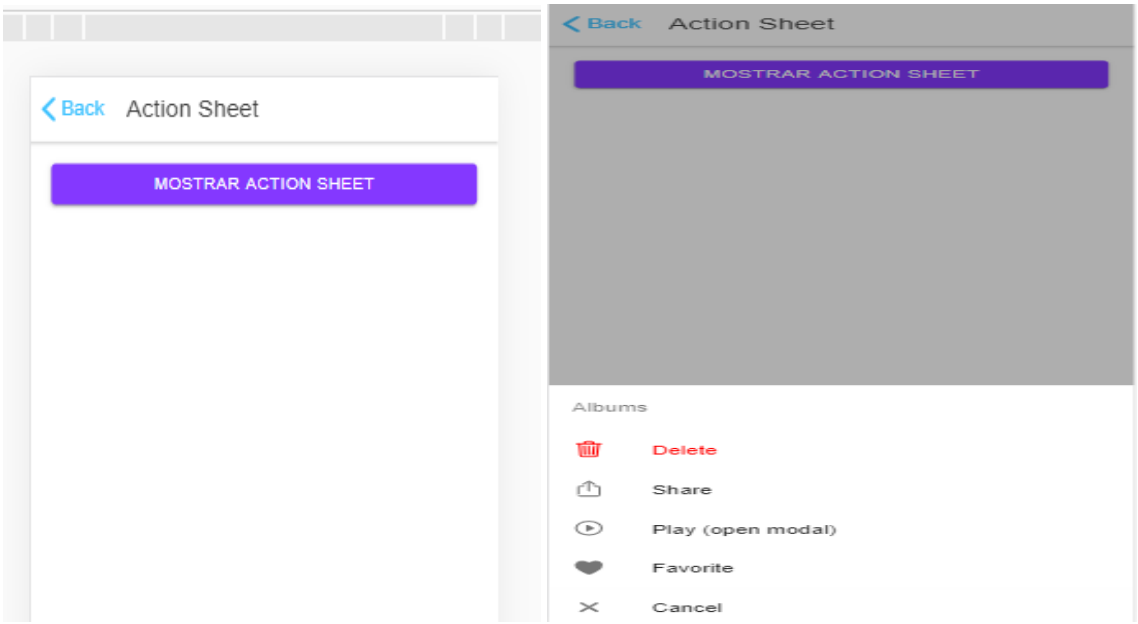
- Colocamos un botón para mostrar el menú (Añadimos un i-button)

```
<ion-button (click)="onClick()" expand="block" >
    Mostrar Action Sheet
</ion-button>
```

- Añadimos i-padding

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

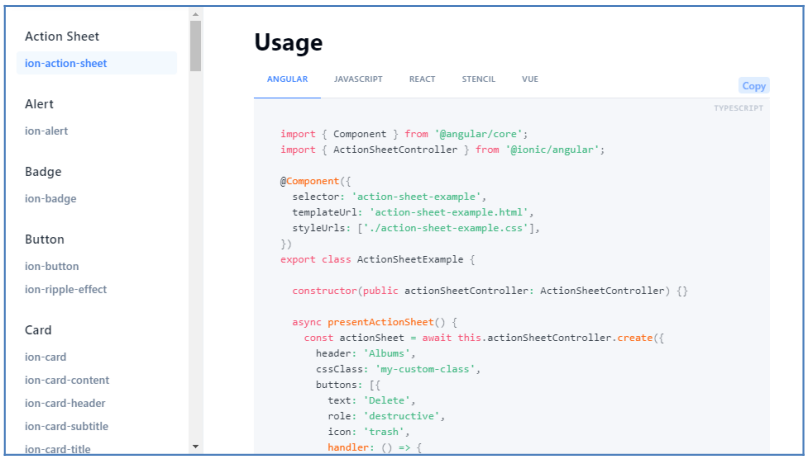
El resultado debe ser algo parecido



Para que nos aparezca el componente al pulsar el botón, tenemos que implementar el evento onClick del mismo

2. En la clase de la página action

- Codifico el evento: voy a la documentación de ionic, al componente action-sheet. En la parte donde nos explica su uso




- Copiamos todo el método presentActionSheet, le llamamos desde el evento onClick

```
export class ActionPage implements OnInit {

  constructor(private actionSheetController: ActionSheetController) { }

  ngOnInit() {
  }

  onClick(){
    this.presentActionSheet();
}


  async presentActionSheet() {
    const actionSheet = await this.actionSheetController.create({
      header: 'Albums',
      cssClass: 'my-custom-class',
      buttons: [{
        text: 'Delete',
        role: 'destructive',
        icon: 'trash-outline',
```

```
        cssClass: 'rojo',
        handler: () => {
          console.log('Delete clicked');
        }
      }, {
        text: 'Share',
        icon: 'share-outline',
        handler: () => {
          console.log('Share clicked');
        }
      }, {
        text: 'Play (open modal)',
        icon: 'caret-forward-circle-outline',
        handler: () => {
          console.log('Play clicked');
        }
      }, {
        text: 'Favorite',
        icon: 'heart',
        handler: () => {
          console.log('Favorite clicked');
        }
      }, {
```

- Añadimos la variable a nuestro constructor e importamos el ActionSheetController

```
import { ActionSheetController } from '@ionic/angular';
```

- En mi caso prefiero los iconos outline, puedo cambiarlos porque no vienen estos por defecto.
- Puedo cambiar el header para que aparezca otro texto
- Puedo en el handler llamar a algún método de mi clase, o bien quitar el console.log
- He personalizado la papelera para que se quede en rojo : en el archivo global.css

```
.rojo, .rojo .sc-ion-action-sheet-md{
    color: red !important;
}
```

- Puedo cambiar los colores del tema, en el archivo variables.css en la parte :root

```
/** Ionic CSS Variables **/
:root {
  /** primary **/
  --ion-color-primary: #d138ff;
  --ion-color-primary-rgb: 56, 128, 255;
  --ion-color-primary-contrast: #ffffff;
  --ion-color-primary-contrast-rgb: 255, 255, 255;
  --ion-color-primary-shade: #3171e0;
  --ion-color-primary-tint: #4c8dff;
```

- Puedo controlar que no se cierre el menú si clikeo en cualquier lugar de la pantalla, de esta manera obligo al usuario a pulsar cancelar o cualquier otra opción. Para ello añado al método presentActionSheet la propiedad backdropDismiss a false

```
async presentActionSheet() {
      const actionSheet = await this.actionSheetController.create({
     header: 'Albums',
     cssClass: 'my-custom-class',
     backdropDismiss:false,
```

## Componente ion-alert

Este componente se usa en ionic tanto para mostrar un cuadro de diálogo como para solicitar información al usuario a través de entradas(texto, casillas de verificación, botones de radio…). Aparece encima del contenido de la aplicación.
Los cuadros de diálogo con un botón son las alertas más sencillas, a partir de ahí pueden ser más complejas

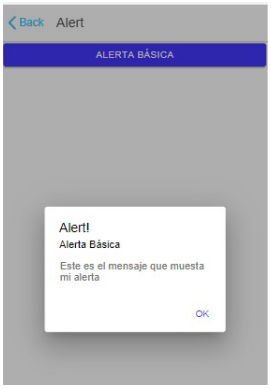
### Alertas básicas

Se utilizan para notificar alguna información, contienen un único botón

```
<app-header titulo="Alert"></app-header>
<ion-content class="ion-padding">
<ion-button (click)="presentAlert()" expand="block" >
    Alerta Básica
</ion-button>
</ion-content>
```

```
constructor(private alertController: AlertController) { }
ngOnInit() {
}
async presentAlert() {
  const alert = await this.alertController.create({
    backdropDismiss:false,
    header: 'Alert!',
    subHeader: 'Alerta Básica',
    message: 'Este es el mensaje que muesta mi alerta',
    buttons: ['OK']
  });

  await alert.present();
}
```

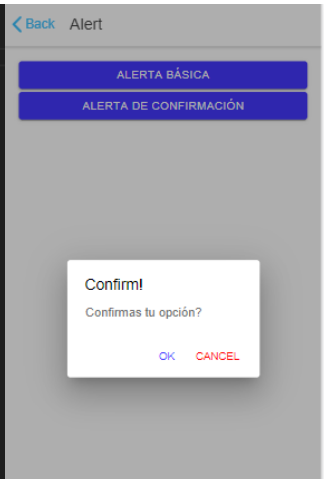
El código de la clase lo copiamos de la documentación. No debemos olvidar importar el componente AlertController de @ionic/angular




**Alertas de confirmación**

Se utilizan para confirmar alguna elección, típico cuadro de diálogo con dos botones Cancelar y Aceptar

```
async presentAlertConfirm() {
  const alert = await this.alertController.create({
    backdropDismiss:false,
    header: 'Confirm!',
    message: 'Confirmas tu opción?',
    buttons: [
      {
        text: 'Ok',
        handler: () => {
          console.log('Confirm Okay');
        }
      },
      {
        text: 'Cancel',
        role: 'cancel',
        cssClass: 'rojo',
        handler: (blah) => {
          console.log('Confirm Cancel: blah');
        }
      }
    ]
  });

  await alert.present();
}
```



Al igual que en el caso anterior hemos copiado el código de la documentación y hecho pequeños cambios.

**Alertas de múltiples botones**
Análogo a los anteriores pero con varios botones

**Alertas que solicitan información**
Copiamos el código correspondiente a presentAlertPrompt. Para tener acceso a los datos que nos rellenan

```
buttons: [
       {
         text: 'Cancel',
         role: 'cancel',
         cssClass: 'secondary',

       }, {
         text: 'Ok',
         handler: (data:any) => {
           console.log(data);
         }
       }
     ]
   });
```

Más adelante veremos qué hacer con esos datos en lugar de mostrarlos sólo por consola

**Alertas de radio**
Similares a las de confirmación pero en lugar de botones, aparecen opciones de radio

**Alertas de verificación**
Como las anteriores pero con casillas de verificación

Ninguna alerta tiene de momento funcionalidad completa, ya veremos cómo invocar a métodos de nuestras clases para que se ejecuten según pulsemos los botones.

## Componente ionic-avatar

Componentes circulares en los que colocar una imagen que representa una persona u objeto. Se pueden colocar dentro de un ion-item, ion-chip…. Su tamaño se ajustará a su contenedor y también el de la imagen.
Las imágenes las pondremos en la carpeta assets y la ruta será

```
<ion-header>
    <ion-toolbar>
        <ion-title>avatar</ion-title>
        <ion-buttons slot="end">
            <ion-avatar>
                <img src="/assets/images/image3.jpeg" />
            </ion-avatar>
        </ion-buttons>

    </ion-toolbar>
</ion-header>
```

Se pueden colocar  en el contenido

```
<ion-content class="ion-padding">
    <ion-chip color="primary" outline="true">
        <ion-avatar>
            <img src="/assets/images/image1.png" />
        </ion-avatar>
        <ion-label>Imagen animada</ion-label>
    </ion-chip>

    <ion-list>
        <ion-item>
            <ion-avatar slot="start">
                <img src="/assets/images/image2.png" />
            </ion-avatar>
            <ion-label>Awesome Label</ion-label>
        </ion-item>
        <ion-item>
            <ion-avatar slot="start">
                <img src="/assets/images/image2.png" />
            </ion-avatar>
            <ion-label>Awesome Label</ion-label>
        </ion-item>
    </ion-list>
</ion-content>
```

## Componente ionic-button

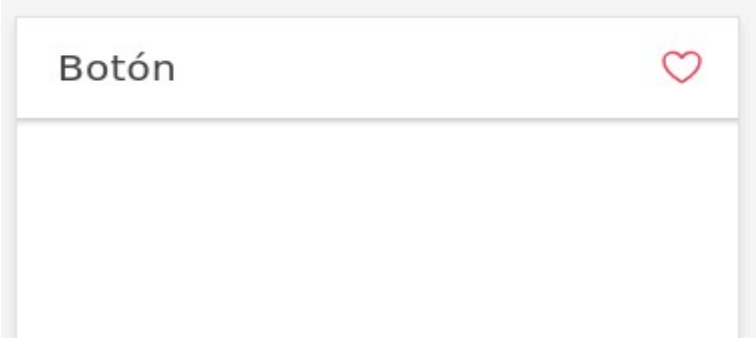
Elemento sobre el que hacer click que se puede colocar en cualquier sitio. Pueden mostrar texto, un icono o ambos.

A través de sus propiedades podemos determinar su aspecto: si está expandido(ocupa toda la pantalla), si tiene relleno, el tamaño. Ya los hemos usado en algún ejemplo y ejercicio. Vamos a añadir funcionalidad al click

**Ejemplo_6:** Añadimos un botón con un icono en la barra de herramientas, que cambiará al hacer clik. Incluimos un ion-button que contiene solamente un ion-icon con su propiedad slot = "icon-only" (Para que al pulsar no se vea efecto cuadrado)

```
<ion-header>
    <ion-toolbar>
        <ion-title>Botón</ion-title>
        <ion-buttons slot="end">
            <ion-button (click)="onClick()" color="danger">
                <ion-icon slot="icon-only" name="heart-outline"></ion-icon>
            </ion-button>
        </ion-buttons>

    </ion-toolbar>
</ion-header>

<ion-content>

</ion-content>
```

Al ejecutar la app quedaría



Queremos que el icono cambie al hacer click, añadimos una variable en la clase que cambie de valor según clikeamos el botón. Usaremos este valor para en el template poner un icono u otro

```
export class BotonPage implements OnInit {

  favorito: boolean =false;
  constructor() { }

  ngOnInit() {
  }

  onClick(){

    this.favorito=!this.favorito;
  }
}
```

En el template, el código quedaría

```
<ion-header>
    <ion-toolbar>
        <ion-title>Botón</ion-title>
        <ion-buttons slot="end">
            <ion-button (click)="onClick()" color="danger">
                <ion-icon slot="icon-only" [name]="favorito ? 'heart' : 'heart-outline' "></ion-icon>
            </ion-button>
        </ion-buttons>

    </ion-toolbar>
</ion-header>

<ion-content>

</ion-content>
```

## Componente <ion-check>

Veamos como incluir checkbox en nuestras app. Para que funcione correctamente debe ir dentro de un ion-item

```
<ion-item>
<ion-label>Programador</ion-label>
<ion-checkbox ></ion-checkbox>
</ion-item>
```

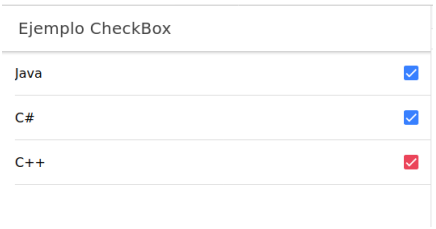
Tiene la propiedad checked, para utilizarla definimos una variable de la clase de tipo booleano y la siguiente sintaxis

```
<ion-item>
<ion-label>Programador</ion-label>
<ion-checkbox [(ngModel)]=”propiedad”></ion-checkbox>
</ion-item>]
```

Podemos decidir que por defecto se active o no con la propiedad de la clase o con su propiedad checked

[(propiedad)]: cuándo colocamos una propiedad entre corchetes indicamos que su valor no será literal sino que lo obtenemos a partir de una variable. Si añadimos (), ésto supone que los cambios en esa propiedad en la plantilla se verán reflejados en la variable de la clase. Es decir el valor pasa de la clase a la plantilla y viceversa de la plantilla a la clase(binding)

Ejemplo_7: Añadimos una página en la que mostramos los lenguajes de programación que conocemos


Si queremos saber sobre qué elemento del ion-item nos han clikeado utilizamos el evento click del componente

```
<ion-content [fullscreen]="true">
    <div class="container">
        <ion-item *ngFor="let l of lenguajes" (click)="onClick(l)">
            <ion-label>{{l.nombre}}</ion-label>
            <ion-checkbox [color]="l.color" [(ngModel)]="l.selected"></ion-checkbox>
        </ion-item>

    </div>
</ion-content>
```

En la clase implementamos el evento onClick

onClick( l : item){
console.log(l) }

Por ejemplo, vamos a mostrar una alerta básica que nos diga que nos han elegido o desmarcado

Por último recordar que lo recomendable es poner los ion-item dentro de un ion-list

## Componente <ion-datetime>

Es recomendable colocarlo dentro de un ion-item

```
<ion-list>
    <ion-list-header>
        <ion-label>Ejemplo Fecha</ion-label>

    </ion-list-header>

    <ion-item>
        <ion-label>Fecha de Nacimiento</ion-label>
        <ion-item>
            <ion-label>Date</ion-label>
            <ion-datetime display-format="DD.MM.YYYY HH:mm"></ion-datetime>
        </ion-item>
    </ion-item>
</ion-list>
```

Si queremos que aparezca un fecha por defecto, declaro una variable en la clase con el valor deseado. Para utilizar la del sistema

```
fNacimiento: Date= new Date();
```

Este es un tipo fecha y hora, si queremos poner en un template una fecha tendremos que convertir esa variable a string para que pueda ser interpretada en html

fNacimiento.toISOString()

```
<ion-item>
    <ion-datetime display-format="DD.MM.YYYY" [value]="fNacimiento.toISOString()"></ion-datetime>
</ion-item>
```

Con el evento ionChange podemos manejar el cambio de fecha

```
<ion-datetime display-format="YYYY.MMMM.DD" [value]="f.toISOString()"
(ionChange)="Ver(this.f)" id="r"></ion-datetime>
```

La clase quedaría

```
Ver(fecha){
console.log(fecha);
}
```

Podemos determinar fecha mínima y máxima para seleccionar en un rango, lo hacemos con las propiedades min y max. Por ejemplo
min=”2000-01-01” max=”2050-01-01”

Estas propiedades están ligadas al formato en que aparece la fecha, si sólo mostramos años, esas propiedades acotan sólo años

Hay otras opciones que podemos investigar en la documentación oficial.

## Componente <ion-fab>

Los botones flotantes que vemos a menudo en las apps. Se pueden colocar en cualquier lugar y se les pueden añadir ciertas funcionalidades.