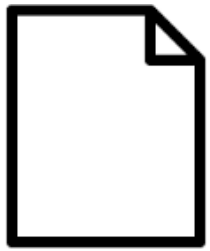
Creación de aplicaciones con Ionic

UT4 - Introducción al desarrollo de aplicaciones hibridas con Ionic

Objetivos de aprendizaje

- Conocer las características principales de Ionic
- Crear una primera aplicación usando Ionic
- Distinguir los principales módulos y componentes que componen una aplicación Ionic



Creación de páginas

Creación de aplicaciones con Ionic

Páginas en Ionic

- Ionic permite manejar el concepto de página
- Una página es un componente que se va a mostrar en una pantalla completa, habitualmente con su propia ruta.
- Las páginas suelen representar vistas completas, mientras que los componentes representan elementos más pequeños y reutilizables

Creando una página

```
ionic generate page contactos
```

- Crea todos los elementos que componen la página Contactos, dentro de una carpeta llamada contactos con los siguientes archivos:

Archivo	Explicación
`contactos.page.ts`	Definición del componente `ContactosPage`
`contactos.page.html`	Vista del componente
`contactos.page.scss`	Estilos del componente

Además, genera una ruta `/contactos` en app.routes.ts que permite navegar a dicha página

Páginas y rutas

```ts
const routes: Routes = [
  {
    path: contactos',
    loadComponent: () => import('./contactos/contactos.page')
        .then( m => m.ContactosPage)
  }
];
```

Carga en diferido (lazy loading). La página solo se carga en el momento en que la utilizamos (mejor rendimiento)

app/app.routes.ts

Generando sub-páginas

```
ionic generate page padre
```

```
ionic generate page padre/hija2
```

```
ionic generate page padre/hija2
```

Las rutas no se crean como rutas de pendientes de padre/, el prefijo simplemente determina la estructura de carpetas

Para indicar rutas anidadas usaremos children o loadChildren()

Generando sub-páginas

- Actualizamos el archivo de routing

```ts
{
    path: 'padre', component:PadrePage,

    children:[
      { path: '', redirectTo: 'hija1', pathMatch: 'full' },
      { path: 'hija1', loadComponent: () => import('./padre/hija1/hija1.page').then( m => m.Hija1Page) },
      { path: 'hija2', loadComponent: () => import('./padre/hija2/hija2.page').then( m => m.Hija2Page) }
    ]
  }
```

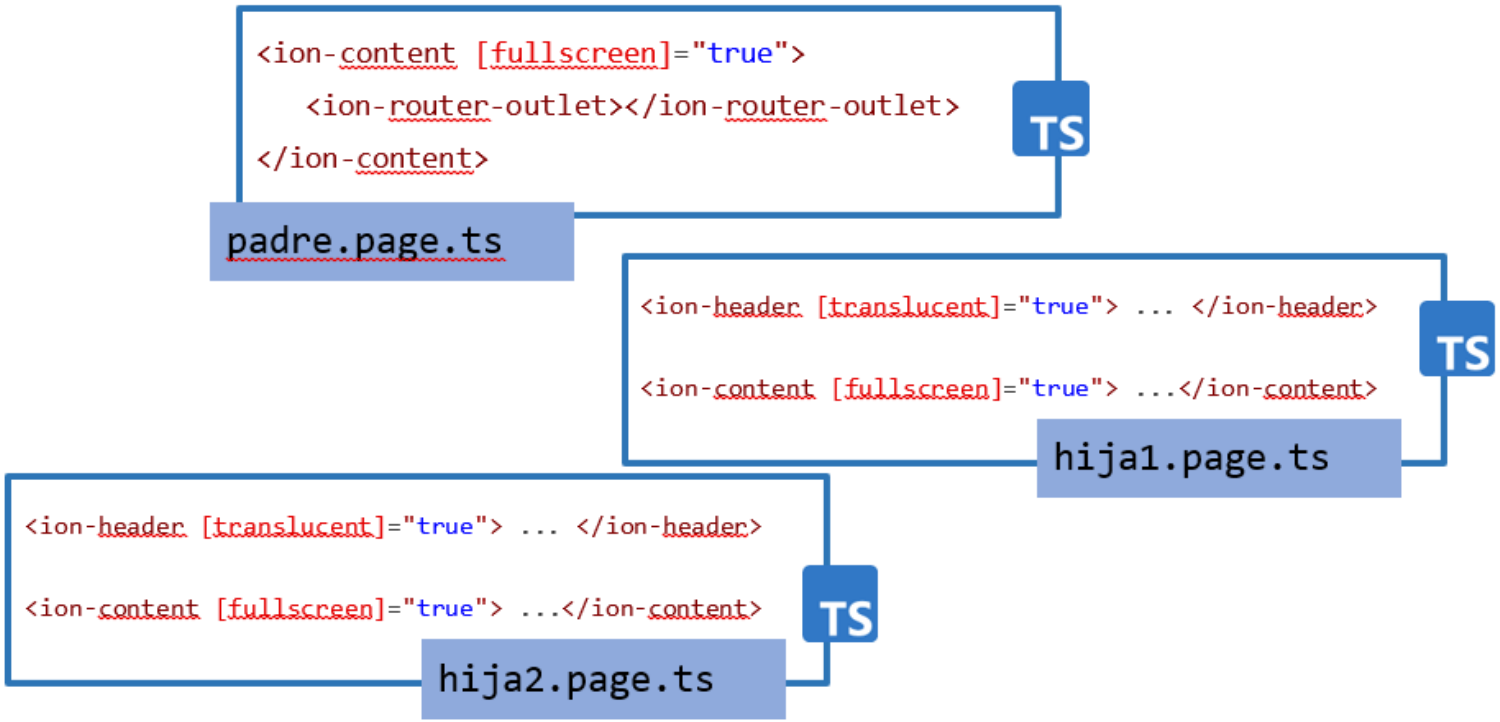
Carga directa

app.routes.ts

Rutas que dependen del padre
/padre/hija1
/padre/hija2

Generando sub-páginas

- Para que funcione, el componente padre debería tener una única sección ion-content con un router-outlet



Generando sub-páginas

- El lazy loading en las páginas hijas no es obligatorio:

```ts
{
    path: 'padre', component:PadrePage,
    children:[
      { path: '', redirectTo: 'hija1', pathMatch: 'full' },
      { path: 'hija1', component: Hija1Page },
      { path: 'hija2', component: Hija2Page }
    ]
  }
```

app.routes.ts

Peor rendimiento pero facilita la lectura de las rutas

Generando sub-páginas

- Otra opción es definir un archivo con las subrutas

```ts
import { Routes } from '@angular/router';


export const routes: Routes = [
      { path: 'hija1', loadComponent: () =>  import('./hija1/hija1.page').then((m) => m.Hija1Page) },
      { path: 'hija2',loadComponent: () => import('./hija2/hija2.page').then((m) => m.Hija2Page) },
      { path: '', redirectTo: 'hija1', pathMatch: 'full'}
]
```

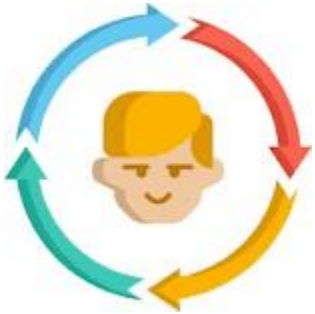
padre.routes.ts

```ts
  {
    path: 'padre',
    loadChildren: () => import('./padre/padre.routes').then((m) => m.routes)
}
```

app.routes.ts

Las rutas hijas de la dada se cargan desde el archivo descrito

Carga diferida



Ciclo de vida

Comenzando con Ionic

Ciclo de vida

- Cuando entramos a una vista, Ionic construye su componente relacionado con él, trabaja con él y cuando dejamos de usarlo se ocupa de destruirlo
- A lo largo del ciclo de vida ocurrirán una serie de eventos que podemos capturar y añadir la lógica que necesitemos
- El ciclo de vida es muy similar al de Angular pero añade algún elemento más

Eventos ligados al ciclo de vida

`constructor`	Se ejecuta cuando creamos el componente (como en cualquier clase)
`OnChanges`	Se ejecuta al inicio y cada vez que Angular detecte cambios en los inputs que tienen alguna propiedad *data-binding*.
`ngOnInit`	Se ejecuta cuando creamos el componente ha sido inicializado Se lanza una única vez (**la primera vez que aparece en pantalla**)
`ionViewWillEnter`	Antes de iniciar la navegación hacia la página.
`ionViewDidEnter`	Una vez que la página está cargada Permite hacer interacciones antes de la carga de la página.
`ionViewWillLeave`	Antes de navegar hacia otra página.
`ionViewDidLeave`	Después de navegar hacia otra página.
`ngOnDestroy`	Se ejecuta cuando el componente se destruye. Elimina posibles subscripciones.

Elementos de navegación

Comenzando con Ionic

Elementos de navegación

- Ionic proporciona varios elementos para facilitar la navegación entre páginas:
  1. RouterOutlet
  2. TabBar
  3. NavController

1. Navegación con RouterOutlet

- La página app.component.html ya tiene integrado el `<ion-router-outlet>`

```
<ion-app>
  <ion-router-outlet></ion-router-outlet>
</ion-app>
```

app.component.html

<ion-router-oulet> tiene la misma funcionalidad del <router-outlet> de Angular pero tiene algún efecto de navegación añadido

1. Navegación con RouterOutlet (II)

- Podemos usar el atributo `routerLink` en los elementos de la plantilla para referenciar a una ruta incluida en los archivos de rutas

```html
<ion-header>
  <ion-toolbar>
    <ion-title>Contactos</ion-title>
    <ion-buttons slot="start">
      <ion-button [routerLink]=["/mapa"]  color="warning">Mapa</ion-button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>
```

1. Navegación con RouterOutlet (III)

- También podemos navegar con router si queremos realizar alguna acción antes de movernos entre páginas

```html
<ion-button (click)="navegarMapa()">Mapa</ion-button>
```

```ts
constructor(private router:Router) { }

public navegarMapa(){
  this.router.navigate("['/mapa'] ");
}
```

Inyección dependencias

Navegamos a una ruta

1. Navegación con RouterOutlet (IV)

ion-router-outlet NO es solo un contenedor: **añade comportamiento nativo tipo app móvil** encima del router de Angular:

- Animaciones de transición (tipo móvil)
- Gestos táctiles
- Swipe back gesture (iOS)
- Historial de navegación tipo app móvil
- Gestión automática del botón "Back"
- Optimizado móvil

Inyección de dependencias con `inject()`

- A partir de Angular 14 se introduce inject() como alternativa para inyección de dependencias.
- Es una función que permite inyectar servicios sin necesidad de declararlos en el constructor

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

constructor(private router: Router) { }
```

```ts
import { inject } from '@angular/core';
import { Router } from '@angular/router';

private router = inject(Router);

constructor() { }
```

Recordemos: Añadiendo parámetros a las rutas

Definición de la ruta

Parámetro

```ts
{path: 'entrada/:id',component:EntradaComponent}
```

Pasando el parámetro

```html
<div id="entrada" routerLink="/entrada/{{entrada.id}}"></div>
```

```html
<div id="entrada" [routerLink]="['entrada',entrada.id]"></div>
```

Recordemos: Añadiendo parámetros I

- Inyectamos `ActivatedRoute`

```ts
export class EntradaComponent {
  public id!:number;

  constructor (private activatedRoute:ActivatedRoute){
    this.activatedRoute.params.subscribe(params=>{
      this.id=+params["id"]; //El + para pasar a entero
    })
  }
}
```

Recordemos: Añadiendo parámetros II

```ts
export class EntradaComponent {
    activatedRoute = inject(ActivatedRoute);

    id = Number(this.activatedRoute.snapshot.paramMap.get('id'));
}
```

Pasando objetos completos

- También podemos proporcionar objetos completos en lugar de un parámetro

TS

```ts
this.router.navigate(["/product"],{state:{product:product}});
```

Componente origen

TS

```ts
constructor() {
  const state = this.router.getCurrentNavigation()?.extras.state;
  if(state){
    this.product=state["product"];
    console.log(this.product);
  }
}
```

Componente destino

<ion-back-button>

- Proporciona funcionalidad de volver a página anterior
  - Muestra un icono y texto por defecto que podemos cambiar
- Si no hay navegación (es la primera página), no se muestra, salvo que tenga su propiedad defaultHRef activa.

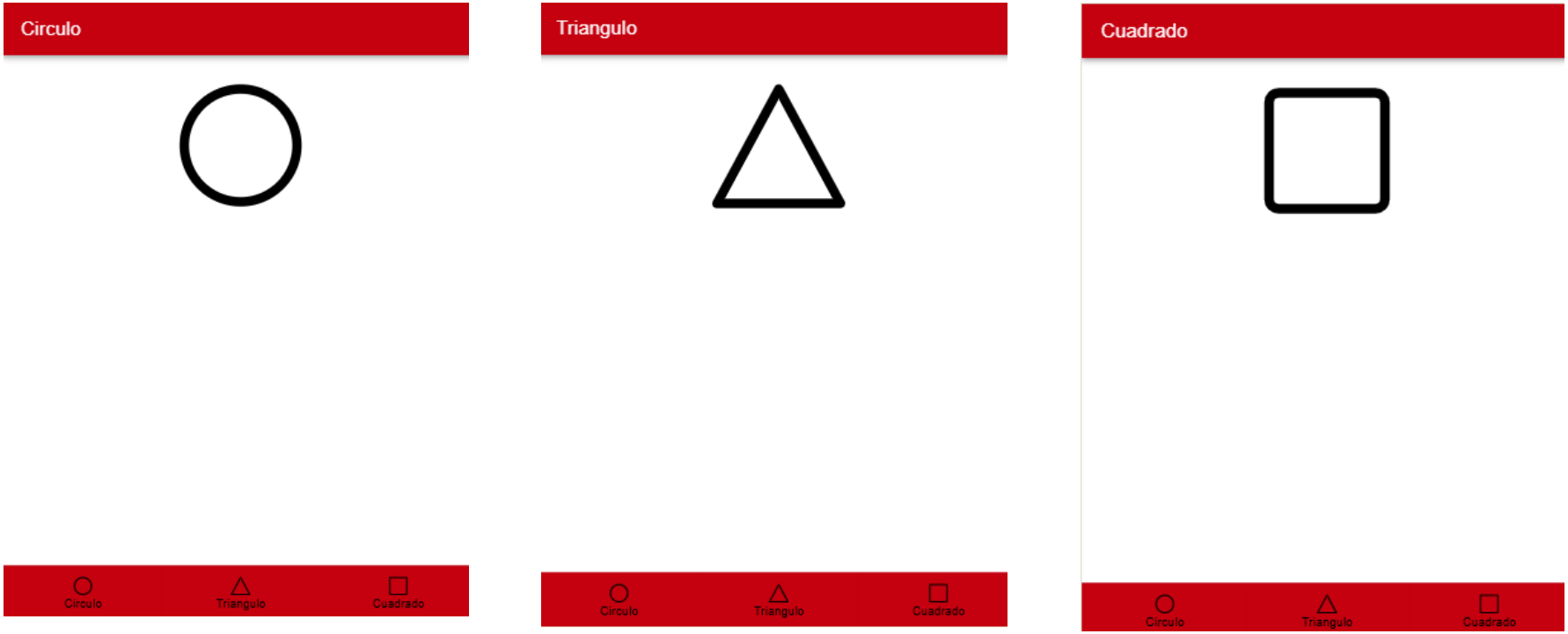
```
<ion-header>
  <ion-toolbar>
    <ion-buttons slot="start" >
      <ion-back-button color="warning" text="Volver"></ion-back-
      button>
    </ion-buttons>
    <ion-title>Transportes</ion-title>
  </ion-toolbar>
</ion-header>
```

2. Navegación a través de TabBar

- Es una barra con pestañas que nos permite cambiar entre vistas/páginas
- Habitualmente se coloca en la parte inferior de la página

Componente	Explicación
`<ion-tabs>`	Componente de alto nivel que contiene varias pestañas
`<ion-tab>`	Pestaña individual
`<ion-tab-bar>`	Contiene un grupo de botones
`<ion-tab-button>`	Botón individual

TabBar: Ejemplo



2. Navegación a través de TabBar (básico)

- Podemos utilizar un TabBar para mostrar contenido sin necesidad de cambiar la URL

```
<ion-tabs>
  <ion-tab tab="circulo">
    <div id="circulo-page">
      <ion-header>
        <ion-toolbar color="danger">    <ion-title>Circulo</ion-title> </ion-toolbar>
      </ion-header>
      <ion-content>
        <div class="example-content"><ion-icon name="ellipse-outline"></ion-icon> </div>
      </ion-content>
    </div>
  </ion-tab>
  <ion-tab tab="cuadrado">… </ion-tab>
  <ion-tab tab="triangulo">… </ion-tab>
```

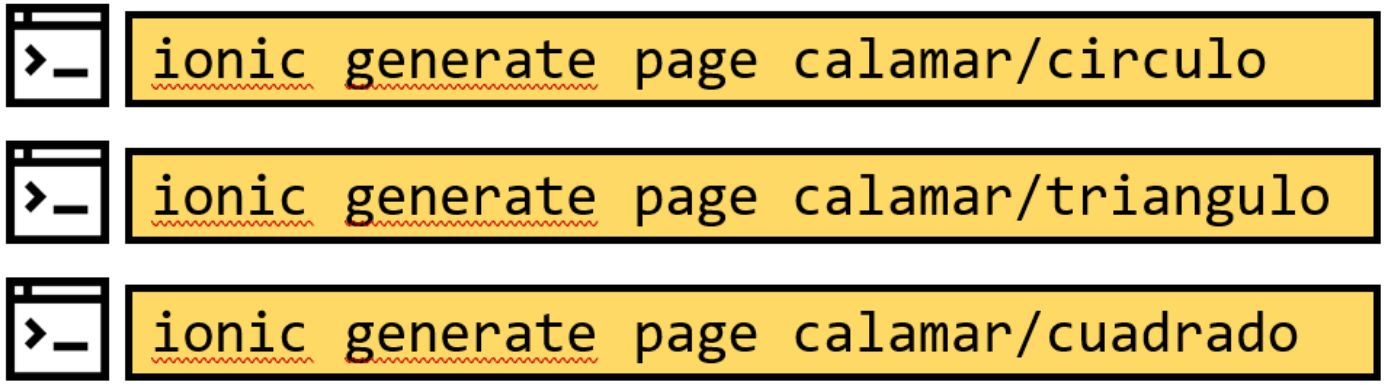
ID de pestaña

Primero definimos el contenido de las pestañas con `ion-tab`

calamar.page.html

2. Navegación a través de TabBar

- También podemos utilizar el router para implementar navegación basada en pestañas.
- Previamente generamos las páginas hijas



Las rutas no se crean como rutas de calamar/, el prefijo simplemente determina la estructura de carpetas

2. Navegación a través de TabBar (II)

- Una vez creadas las páginas definimos nuestro sistema de rutas para indicar que las páginas son hijas de una página principal:

```ts
{
  path: 'calamar', component: CalamarPage,
  children:[
    { path: '', redirectTo: 'circulo’, pathMatch: 'full' }
    { path:"circulo", loadComponent: () => import('./circulo/circulo.page').then( m => m.CirculoPage) },
    { path: 'cuadrado’,loadComponent: () => import('./cuadrado/cuadrado.page').then( m => m.CuadradoPage) },
    { path: 'triangulo’, loadComponent: () => import('./triangulo/triangulo.page').then( m => m.TrianguloPage)}
  ]
}
```

app.routes.ts

2. Navegación a través de TabBar (III)

- Definimos las tabs en la página contenedora

```html
<ion-tabs>
  <ion-tab-bar color="danger"  slot="bottom">
    <ion-tab-button tab="circulo"> <ion-icon name="ellipse-outline"></ion-icon> Circulo  </ion-tab-button>
    <ion-tab-button tab="triangulo"> <ion-icon name="triangle-outline"></ion-icon> Triangulo ion-tab-button>
    <ion-tab-button tab="cuadrado"> <ion-icon name="square-outline"></ion-icon> Cuadrado </ion-tab-button>
  </ion-tab-bar>
</ion-tabs>
```

Ruta a la que lleva al pulsar.
Se añade a la actual (ej: /calamar/circulo)

calamar.page.html

```ts
constructor() {
    addIcons({squareOutline , ellipseOutline, triangleOutline });
 }
```

calamar.page.ts

3. Navegación a través de NavController

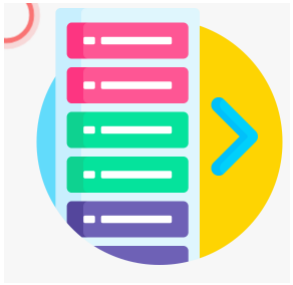
- NavController es un objeto que nos facilita la navegación entre elementos
- Es un servicio inyectable

Método	Explicación
`navigateForward(ruta)`	Navega hacia una nueva ruta `this.navController.navigateForward("/contactos/detalle");`
`navigateBack(ruta)`	Vuelve hacia una ruta ya cargada `this.navController.navigateBack("/contactos/listado");`
`pop()`	Va a la última ruta (desde donde se me ha llamado) Útil para botón volver

3. Botón de cierre con NavController

```html
<ion-header [translucent]="true">
  <ion-toolbar>
    <ion-title>DetalleContacto</ion-title>
    <ion-buttons slot="primary">
        <ion-icon name="close" slot="icon-only" (click)="onClickVolver()"></ion-icon>
    </ion-buttons>
  </ion-toolbar>
</ion-header>
```

```typescript
public onClickVolver(){
  this.navController.pop();
}
```



Otros componentes

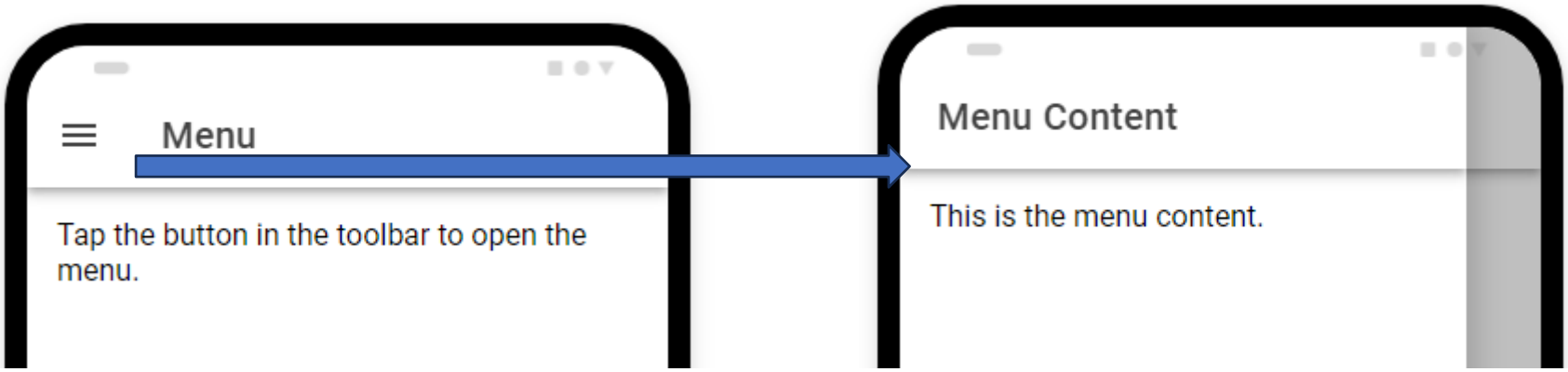
Creación de compomentes con Ionic

Otros componentes

- Menús laterales
- Modales
- Alertas
- Elementos arrastrables

Uso de menús laterales

- Se abren y cierran según deseemos
- Lo colocamos por encima de router-outlet por que se muestra por encima de todo el contenido



Uso de menú lateral

```
<ion-app>
  <ion-menu slot="start" menuId="menu1" contentId="main">
    <ion-header>
        <ion-toolbar>
          <ion-title>Agenda</ion-title>
        </ion-toolbar>
    </ion-header>
    <ion-content>
      <ion-list>
        <ion-menu-toggle>
          <ion-item>
            <ion-label routerLink="/contactos/detalle-contacto">Detalle</ion-label>
          </ion-item>
        </ion-menu-toggle>
      </ion-list>
    </ion-content>
  </ion-menu>
  <ion-router-outlet id="main"></ion-router-outlet>
</ion-app>
```

Asociamos el menú al contenido principal (ID del router outlet)

El ion-menu-toggle hace que se abre y se cierre

app.page.html

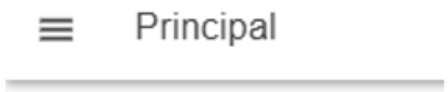
Abriendo el menú lateral (opción A)

En el elemento donde queremos abrir el menú

```
<ion-buttons slot="start">
    <ion-menu-button menu="menu1"></ion-menu-button>
</ion-buttons>
```

ID del menú

principal.page.html



Abriendo el menú lateral (opción B)

En el elemento donde queremos abrir el menú

```html
<ion-buttons slot="start">
      <ion-button (click)="openMenu()">Abre menú</ion-button>
</ion-buttons>
```

```typescript
constructor(private menuController:MenuController) { }

public openMenu(){
  this.menuController.toggle();
}
```

Componentes modales

- Son vistas que presentan información destinada a eliminarse (cerrarse).
- Se pueden gestionar con un controlador específico
- Creación de un componente

```
ionic generate component modals/Info
```

No generamos página porque no va asociado a una ruta

Definición de un modal

```
<ion-header>
  <ion-toolbar>
    <ion-title>Información </ion-title>
  </ion-toolbar>
</ion-header>
<ion-content class="ion-padding">
    <h2>Información de la aplicación</h2>
</ion-content>
```

info.component.html

Abriendo el modal

```html
<ion-button (click)="openModal()">Abre modal</ion-button>
```

```ts
constructor(private modalController:ModalController) { }

 public onOpenModal(){
   this.modalController.create({
     component:InfoComponent,
   }).then(modal=>{
     modal.present();
   })
 }
```

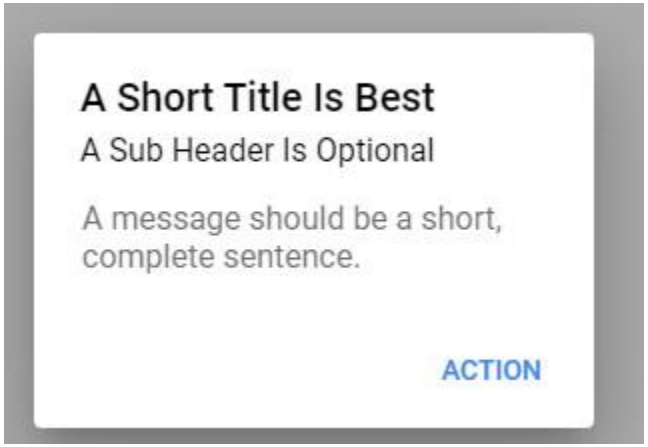
Pasando datos a un modal con input

```ts
this.modalController.create({
      component:InfoComponent,
      componentProps:{
        nombre:"Godzilla",
        puntosVida:300,
        color:"verde"
      }
    }).then(modal=>{
      modal.present();
    })
```

```ts
export class InfoComponent  {
  nombre = input.required<string>();
  puntosVida = input.required<number>();
  color = input.required<string>();
```

Alertas

- Permiten mostrar mensajes de manera sencilla, que se muestran como un modal
- Permiten personalizar los botones



Alerta simple (controlador)

```ts
constructor(private listaCompraService:ListaCompraService,
            private alertController:AlertController) { }

async muestraAlerta(){
    const alert = await this.alertController.create({
      header: 'Cabecera',
      message: 'Mensaje',
      buttons:['OK']
    })
    await alert.present();
  }
```

Alerta simple (inline)

```
<ion-button id="alerta">Pulsame</ion-button>
    <ion-alert
      trigger="alerta"
      header="Titulo"
      subHeader="Subtitulo"
      message="Mensaje."
      [buttons]="['OK']">
    </ion-alert>
```

Alertas complejas

```ts
const alert=await this.alertController.create({
        header:'Confirmación',
        message:'Seguro?',
        buttons:[
          {
            text:"Si",
            handler:()=>{
              /* Lo que quiero hacer en la acción */
            }
          },
          {
            text:"No",
            handler:()=>{
              alert.dismiss();
            }
          }
        ]

    });
    await alert.present();
```

Elementos arrastrables

- Permiten crear ítems de un listado arrastrables

```
<ion-list>
  <ion-item-sliding>
    <ion-item>
      <ion-label>Sliding Item with End Options</ion-label>
    </ion-item>

    <ion-item-options>
      <ion-item-option>Favorite</ion-item-option>
      <ion-item-option color="danger">Delete</ion-item-option>
    </ion-item-options>
  </ion-item-sliding>
</ion-list>
```

Formularios

Creación de aplicaciones con Ionic

Formularios

- Vamos a utilizar formularios utilizando el modelo reactivo
  - Son formularios que creamos mediante código en el controlador.
- Requisito previo: Importar ReactiveFormsModule en la página donde vamos a usarlo

```ts
@NgModule({
  imports: [
    CommonModule, ReactiveFormsModule, IonicModule, NuevoContactoPageRoutingModule
  ],
  declarations: [NuevoContactoPage]
})
export class NuevoContactoPageModule {}
```

Clases/etiquetas usadas

Clase	Uso
`FormGroup`	Permite crear el formulario
`FormControl`	Cada uno de los campos del formulario
`<ion-input>`	Campos del formulario (vista)

Definiendo la vista del formulario

```
<form [formGroup]="formulario" (ngSubmit)="onSubmit()">
    <ion-item>
      <ion-input label= "Nombre" type="text" formControlName="nombre"></ion-input>
    </ion-item>
    <ion-item>
      <ion-input label= "Edad" type="number" formControlName="edad"></ion-input>
    </ion-item>
    <ion-item>
      <ion-input label= "EMail"  type="email" formControlName="email"></ion-input>
    </ion-item>
    <ion-item>
      <ion-input label= "Password" type="password" formControlName="password"></ion-input>
    </ion-item>
    <ion-item>
      <ion-input label= "DNI" type="text" formControlName="dni"></ion-input>
    </ion-item>
      <ion-button type="submit" [disabled]="!formulario.valid" class="ion-margin">Nuevo contacto</ion-button>
```

Mostrando errores (refactorizando)

```html
<ion-item>
  <ion-label>Nombre</ion-label>
  <ion-input #nombre type="text" formControlName="nombre"></ion-input>
  <ion-note slot="error" *ngIf="compruebaError('nombre','required')">
       El campo nombre es requerido</ion-note>
  <ion-note slot="error" *ngIf="compruebaError('nombre','minlength')">
       El campo nombre debe tener como mínimo 10 caracteres</ion-note>
</ion-item>
```

```ts
public compruebaError(campo:string,error:string){

        return this.formulario.get(campo)?.hasError(error) &&
               this.formulario.get(campo)?.touched

}
```

Validadores

Validador	Significado
`required`	Campo obligatorio
`maxlength`	Longitud máxima. Recibe un entero
`minlength`	Longitud mínima. Recibe un entero
`email`	El campo es un email
`max`	Valor máximo. Recibe un entero
`min`	Valor mínimo. Recibe un entero
`pattern`	Patron. Recibe expresión regular