UA . MASTER MOVILES

MÁSTER UNIVERSITARIO EN DESARROLLO DE SOFTWARE PARA DISPOSITIVOS MÓVILES

PROGRAMACIÓN HIPERMEDIA PARA DISPOSITIVOS MÓVILES

Ionic v4 - TypeScript, Arquitectura y Navegación

CONTENIDO

1. TypeScript:
    1. Introducción
    2. Variables, arrays
    3. Clases
    4. Importación y exportación
2. Arquitectura Angular
    1. Módulos
    2. Páginas
3. Navegación
4. Ciclo de vida de las páginas

TYPESCRIPT

TYPESCRIPT

- Es el lenguaje que utilizaremos en Angular / Ionic.
- TypeScript es un superconjunto de JavaScript (permitiendo utilizar también JavaScript).
- Los ficheros tendrán la extensión "`.ts`".
- Para utilizarlo es necesario ***transpilarlo*** a código JavaScript.

https://www.typescriptlang.org/

https://www.cheatography.com/gregfinzer/cheat-sheets/typescript/pdf/

TYPESCRIPT: VARIABLES TIPADAS

TypeScript permite definir el tipo de las variables:

```
let a: string = 'Test variable';
let b: number = 1;
let c: boolean = true;
let d: any = 'any type';

b += 1;
a = 5;      // ERROR!

const e: number = 100;
e = 5;      // ERROR!
```

TYPESCRIPT: TIPOS COMPLEJOS

También podemos definir arrays y objetos tipados:

```
let ar: string[] = ['aaa', 'bbb', 'ccc'];
let br: number[] = [1, 2, 3];
let cr: boolean[] = [false, true, true];
let dr: any[] = [1, 'aaa', false];

// Tipado con objetos
let address: { street: string, city: string };
address = {street: 'Main st', city:'Boston', state:'MA'};

// Tipado con clases
let obj: MiClase = new MiClase();
```

TYPESCRIPT: CLASES

TypeScript permite declarar clases usando la palabra reservada "`class`":

```
class MiClase {
    constructor() {
        // Constructor de la clase
    }
    miFuncion() {}
}

let a = new MiClase();      // Instanciamos la clase
a.miFuncion();              // Usamos una función
```

Para la definición de funciones **dentro de una clase NO** hay que usar "`function`"

TYPESCRIPT: DEFINICIÓN DE ÁMBITO

Con TypeScript podemos definir el ámbito de las variables o funciones:

```
class MiClase {
    private valor: number;
    constructor(valor: number) {
        this.valor = valor;
    }
    public getValor(): number {
        return this.valor;
    }
}
```

Definimos el tipo del argumento

Usamos **this** para acceder a la variable "valor" de la clase.

Definimos el tipo del valor de retorno

TYPESCRIPT: EXPORTACIÓN E IMPORTACIÓN

- Para poder importar una clase desde otro fichero es necesario “exportarla”:

```
export class HomePage { }
```

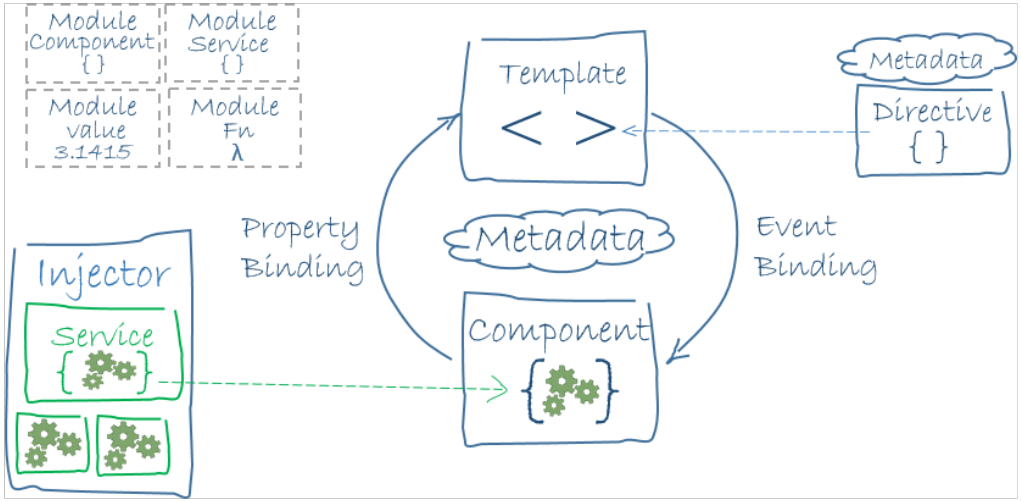
- De esta forma después podremos:

```
import { HomePage } from '../home/home';
```

ARQUITECTURA ANGULAR

ARQUITECTURA ANGULAR

- Ionic está perfectamente integrado con **Angular**.
- Los componentes que incluye son en realidad directivas de Angular.
- Definiremos nuestra aplicación y sus pantallas utilizando los siguientes elementos:



- Módulos
- Páginas o componentes
- *Templates*, plantillas o vistas
- Servicios o proveedores de contenidos
- Metadatos
- *Data binding*
- Directivas
- Inyección de dependencias

MÓDULOS

- Un módulo agrupa un conjunto de código utilizado en un ámbito concreto de la aplicación.
- Se definen usando el decorador `@NgModule` con los siguientes metadatos:
  - `declarations` – Vistas que pertenecen al módulo (páginas, directivas o pipes).
  - `imports` – Otros módulos requeridos por los componentes del módulo.
  - `entryComponents` – Componentes utilizados en las plantillas.
  - `providers` – Proveedores de contenido utilizados.
  - `bootstrap` – Define la vista raíz. Utilizado solo por el root module.

MÓDULOS – ROOT MODULE

- Toda aplicación tiene un módulo principal o root module.
- Por convenio se llama "`AppModule`" y lo podemos encontrar en "`src/app/app.module.ts`".
- Carga toda la aplicación, incluido: el componente principal (`app.component.ts`) y el módulo de rutas (`app-routing.module.ts`).
- Tendremos que añadir a este módulo los módulos que vayamos a utilizar en toda la aplicación.

PÁGINAS O COMPONENTES

- Las pantallas de una aplicación en Ionic se crean mediante páginas o componentes.
- Cada página se compondrá de:
  - `.html` → Plantilla o vista.
  - `.ts` → Clase en TypeScript con la definición de la clase.
  - `.scss` → Definición de estilos de la clase en SASS.
- Para crear una página usamos el CLI de Ionic:

```
$ ionic g page pagina2
```

- Esto nos creará la página "`Pagina2Page`" en la carpeta "`src/app/pagina2`".

PÁGINAS: CLASE TYPESCRIPT

- El fichero con la clase TypeScript contendrá por defecto:

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

@Component({
  selector: 'app-pagina2',
  templateUrl: './pagina2.page.html',
  styleUrls: ['./pagina2.page.scss'],
})
export class Pagina2Page implements OnInit {

  constructor() { }

  ngOnInit() { }

}
```

Decorador con metadatos: selector, plantilla y SCSS

Constructor

La vista se ha cargado

PÁGINAS: PLANTILLA O TEMPLATE

- Si abrimos la plantilla asociada a la página veremos que por defecto contiene:

```
<ion-header>
  <ion-toolbar>
    <ion-title>Pagina2</ion-title>
  </ion-toolbar>
</ion-header>

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

</ion-content>
```

PÁGINAS: RUTA

- Al crear una página automáticamente se añade una entrada al fichero de rutas "`app-routing.module.ts`".
- Dentro de este fichero podremos ver que hay un array con las páginas de la aplicación:

```
const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  {
    path: 'home',
    loadChildren: () => import('./home/home.module')
                              .then( m => m.HomePageModule)},
  {
    path: 'pagina2',
    loadChildren: () => import('./pagina2/pagina2.module')
                              .then( m => m.AutorPageModule)
  },
];
```

FICHERO DE RUTAS

- En el fichero de rutas "`src/app/app-routing.module.ts`" podemos encontrar tres tipos de rutas:
  - `{path: 'home', redirectTo: 'page2' }` → Redirección a una nueva ruta. La ruta a la que se redirecciona ("`page2`") deberá de estar definida.
  - `{path: 'home', loadChildren: '../lazy..' }` → Definición de la ruta "`home`" mediante el modo *lazy loading* (la página se cargará en el momento en el que se solicite).
  - `{path: 'home', component: HomePage }` → Definición de la ruta "home" asignada al componente o página "`HomePage`". En este caso la página estará pre-cargada desde el inicio.

NAVEGACIÓN

NAVEGACIÓN

- Ionic utiliza el sistema de navegación basado en pila:
  - Al abrir una página se añade a la pila.
  - Al cerrar una página se quita de la pila.
- El modo de carga seleccionado en el fichero de rutas afectará a la carga y cierre de las páginas.
- Si utilizamos el modo "*lazy loading*":
  - La página se creará y cargará cuando se solicite su apertura.
  - Al cerrar una página se destruirá.
  - Dependiendo de la memoria disponible y las páginas cargadas es posible que se destruyan páginas no visibles (apiladas).

NAVEGACIÓN: ABRIR PÁGINAS

- Para abrir una nueva página al pulsar un botón utilizaremos el atributo "`routerLink`":

```
<ion-button [routerLink]="['/pagina2']">Abrir</ion-button>
```

- También podemos añadir el atributo "`routerLink`" a otros elementos pulsables, como un "`ion-item`":

```
<ion-item [routerLink]="['/pagina2']">
   <ion-label>Abrir página 2</ion-label>
</ion-item>
```

- Opcionalmente podemos definir la dirección de la animación mediante el atributo "`routerDirection`", el cual puede valer "`back`", "`forward`" o "`root`".

NAVEGACIÓN: ABRIR PÁGINAS

- La acción de abrir páginas también se puede realizar desde el controlador (fichero "`.ts`" asociado).
- Para esto desde la vista tendremos que indicar que al pulsar sobre el botón se ejecute una función del controlador, por ejemplo:

```
<ion-button (click)="abrirPagina2()">Abrir</ion-button>
```

- En el controlador asociado tendremos que definir la función "`abrirPagina2()`".

NAVEGACIÓN: ABRIR PÁGINAS

- Para abrir una página desde el controlador utilizaremos el método "`navigate`" de la clase "`Router`":

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

@Component({
  …
})
export class HomePage implements OnInit {

  constructor(private router: Router) {}

  abrirPagina2(): void {
    this.router.navigate(['/pagina2']);
  }
}
```

Importamos la clase

Inyección de dependencias

- También podemos utilizar el método "`this.router.navigateByUrl(`/detail`);`" que recibe un *string* con la URL completa.

NAVEGACIÓN: PARÁMETROS

- Para la navegación con parámetros en primer lugar hay que modificar el fichero de rutas:

```
{ path: 'detail', loadChildren: '...' },
{ path: 'detail/:id', loadChildren: '...' },
```

- También podemos añadir varios parámetros:

```
{ path: 'detail/:p1/:p2/:p3', loadChildren: '...' },
```

- **IMPORTANTE**: Estos parámetros son **obligatorios**, por lo que si se accede a la ruta sin facilitar dichos parámetros se producirá un error.

NAVEGACIÓN: PARÁMETROS

- Si utilizamos un botón para abrir la página tendremos que añadir los parámetros de llamada de la siguiente forma:

```
<ion-button [routerLink]="['/detail', id]">
```

- También podemos añadir los parámetros al abrir la página desde el controlador:

```
this.router.navigate(['/detail', id]);
```

- En ambos casos en lugar de indicar un array también podemos especificar una URL con la ruta completa:

```
<ion-button routerLink="/detail/1">
this.router.navigateByUrl('/detail/1');
```

NAVEGACIÓN: RECUPERAR PARÁMETROS

- Para leer los parámetros recibidos desde otra página usaremos la clase **`ActivatedRoute`**:

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

@Component({
   ...
})
export class Pagina2Page implements OnInit {
   id: any;

   constructor(private activatedRoute: ActivatedRoute) {}

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

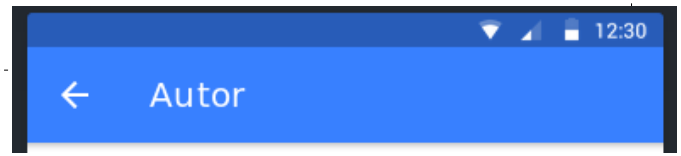
Importamos la clase

Inyección de dependencias

NAVEGACIÓN: BACK BUTTON

- Para completar el sistema de navegación vamos a añadir al toolbar superior un botón que permita volver a la página anterior.

```
<ion-header>
  <ion-toolbar>
    <ion-buttons slot="start">
      <ion-back-button defaultHref="home"></ion-back-button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>
```



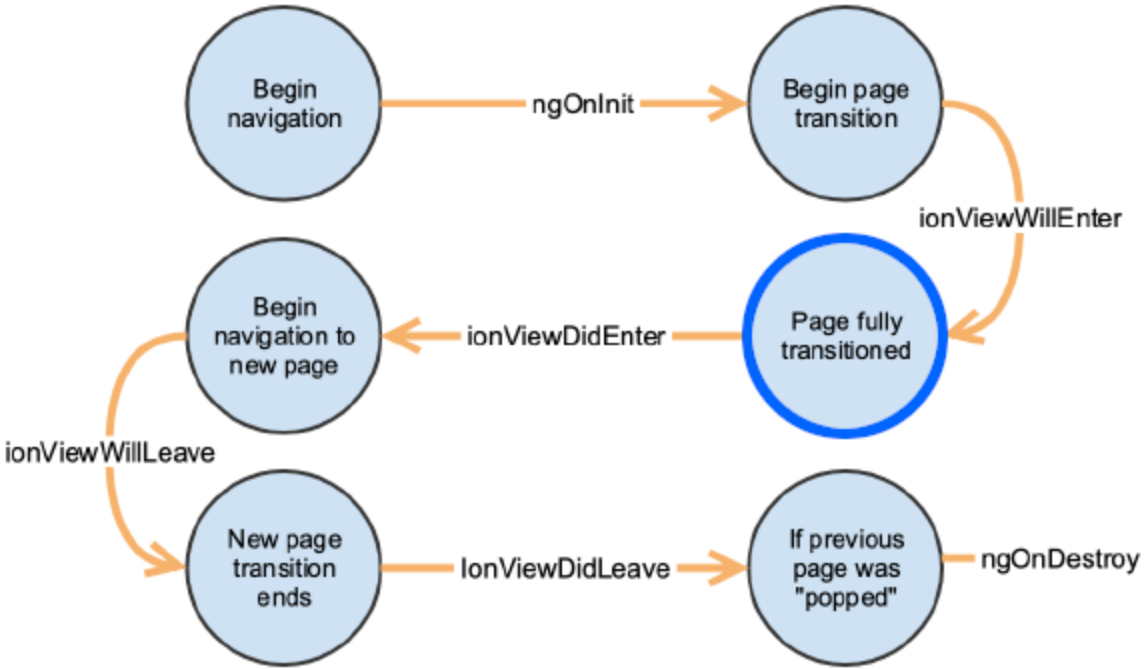
- El atributo "`defaultHref`" nos permite indicar una página por defecto a la que volver. Esta opción es útil cuando la página se abre directamente, sin pasar por las páginas anteriores, por lo que la pila estaría vacía.

CICLO DE VIDA
DE UNA PÁGINA

CICLO DE VIDA DE LAS PÁGINAS

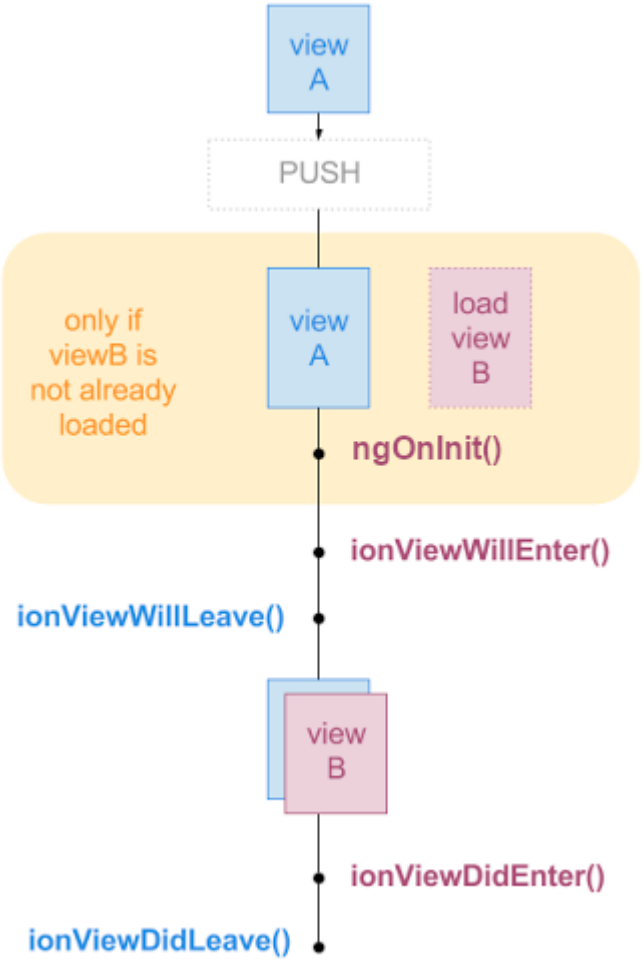
- Al cargar una página se producen los siguientes eventos:
  - `ngOnInit` → La página ya está cargada.
  - `ionViewWillEnter` → La página se va a mostrar.
  - `ionViewDidEnter` → La página ya es visible.
- Y al salir de una página:
  - `ionViewWillLeave` → La página va a salir.
  - `ionViewDidLeave` → La página ya no es visible.
  - `ngOnDestroy` → Se va a descargar la página.

CICLO DE VIDA

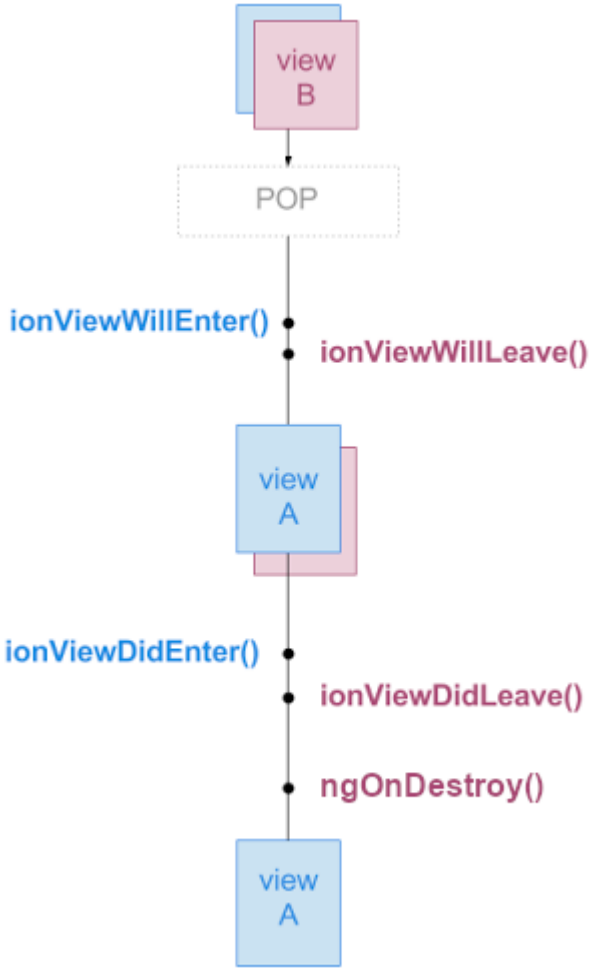


CICLO DE VIDA

Al abrir una nueva página



Al cerrar una página



CICLO DE VIDA

- Para sobrescribir el método `ngOnDestroy` tendremos que importarlo y hacer que la clase lo implemente:

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

@Component({
   …
})
export class HomePage implements OnInit, OnDestroy {
  constructor() {}
  ngOnInit() {
     console.log('HomePage ngOnInit');
  }
  ionViewDidEnter() {
     console.log('HomePage ionViewDidEnter');
  }
  ngOnDestroy() {
     console.log('HomePage ngOnDestroy');
  }
}
```

¿PREGUNTAS?