Componentes nativos de 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
- Introducir el concepto de decoración de clases (TS)
- Distinguir los conceptos de módulo y componente (Angular)

Componentes de Ionic

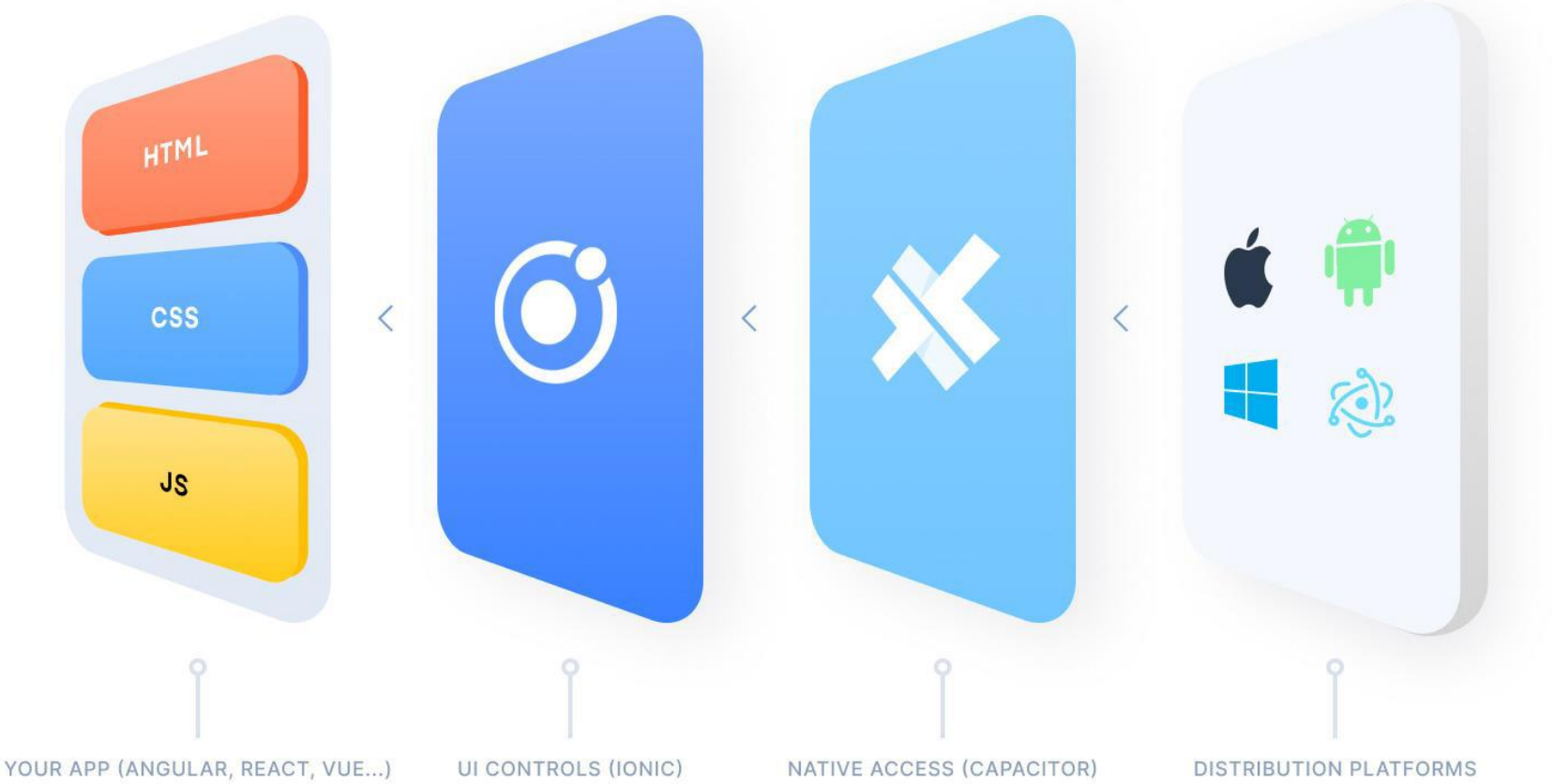
- Ionic proporciona una amplia variedad de componentes prediseñados que podemos usar para desarrollar nuestra App.
- Son muy parecidos a los ofrecidos por las interfaces nativas
- Los componentes son etiquetas HTML que podemos usar en cualquier parte de nuestra aplicación.

Capacitor

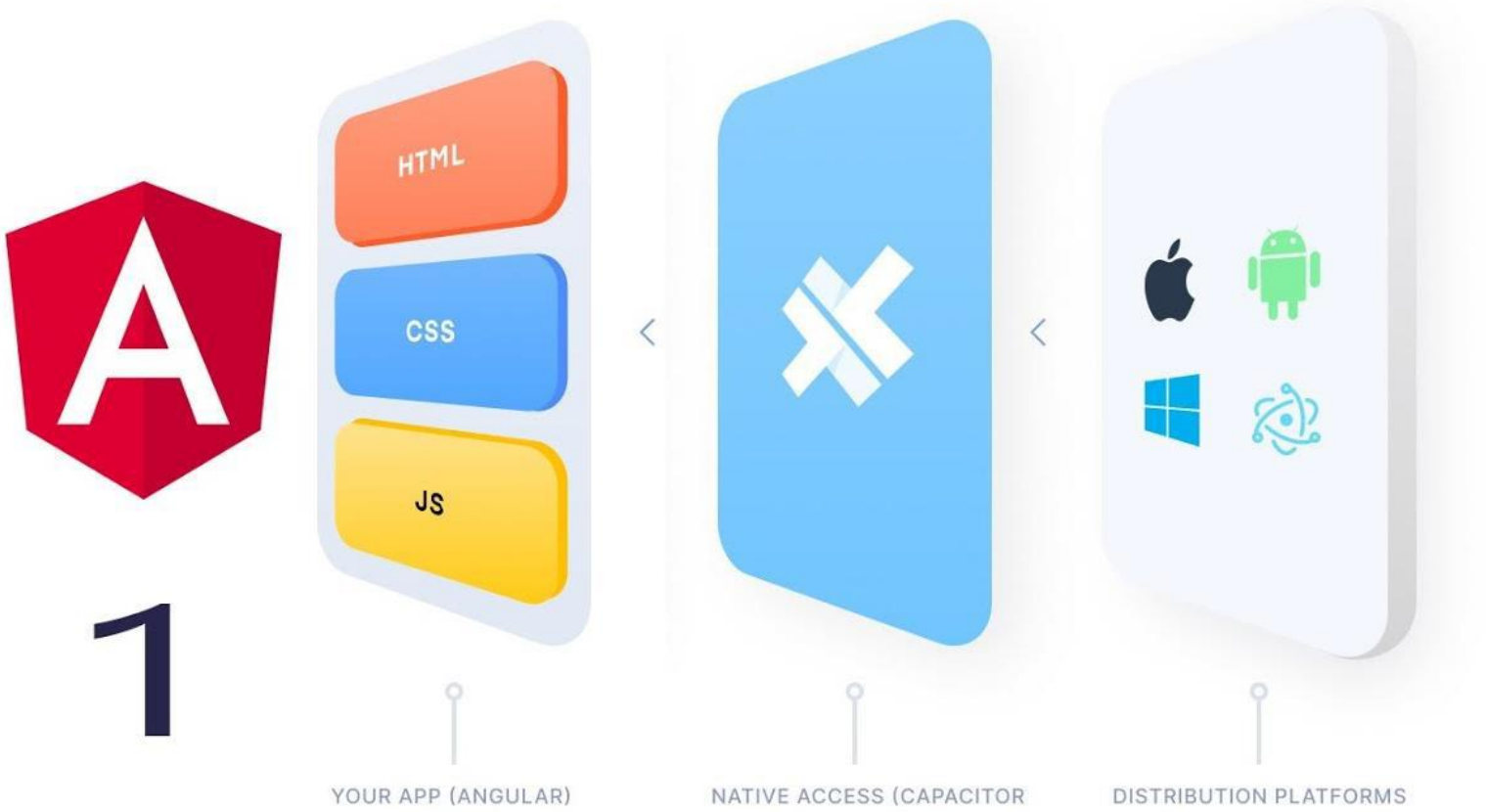
- Es la herramienta usada por Ionic para gestionar el acceso a las funciones nativas del dispositivo (cámara, GPS, giroscopio, notificaciones push, etc..)

Es una evolución de Cordova

Capas en el desarrollo



Capas en el desarrollo (afinando más)



Preparación del entorno

Comenzando con Ionic

Preparando el entorno

- Node / npm  
- VSCode + extensiones  
- Ionic CLI 

Node.js y npm

- **Node.js** es nuestro entorno de ejecución de JS
  - Simplificando mucho, me permite ejecutar código JS fuera del navegador Web (por ejemplo, en un servidor Web).
- **npm** (Node Package Manager) es el gestor de paquetes usado por Node.js.

Es una idea similar a apt-get en Debian, que permite tanto descargar los paquetes de una fuente fiable como actualizarlos fácilmente

npm viene incluido en la instalación de Node.js

Instalando Node.js

- Descargamos el instalador desde la página de descargas de Node
  - Elegimos versiones LTS (más estables) frente a las actuales

- Comprobamos si ya tenemos Node instalado

```
node --version
```

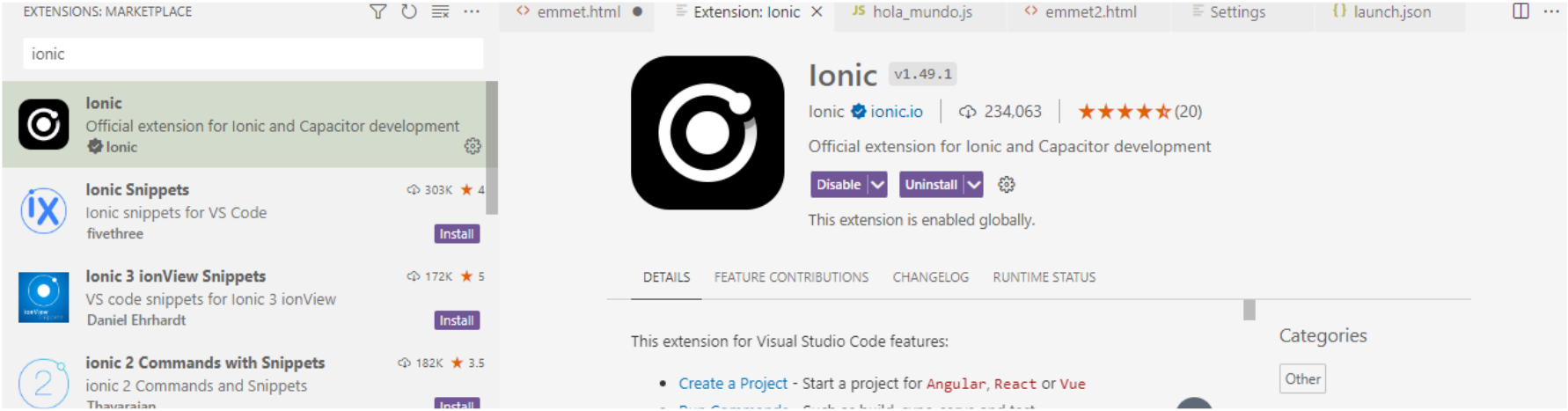
npm (Node Package Manager)

- Permite obtener librerías/paquetes de manera simple.
  - A las librerías que utilizo en mi aplicación las llamamos dependencias
- Por defecto, las dependencias se instalan en una carpeta local (`node_modules`) dentro de nuestro proyecto
  - También podemos indicar que queremos instalarlos de manera global para todos los proyectos
- Comprobamos si ya tenemos npm instalado

```
npm --version
```

Extensiones de VSCode

- Extensión oficial de Ionic



Ionic CLI

- Mediante comandos agilizamos las tareas más repetitivas del desarrollo
- Instalación de Ionic CLI desde un terminal

```
npm install -g @ionic/cli
```

-g → Instalación global (lo puedo usar desde cualquier sitio)

- Compruebo

```
ionic --version
```

Versión del CLI, NO versión del framework

Algunas versiones de Ionic

Versión de Ionic	Fecha de lanzamiento	Versiones de Angular compatibles
Ionic 8	Abril 2024	16+
Ionic 7	Mayo 2023	15, 16
Ionic 6	2022	13, 14

Podemos utilizar una versión de Angular en nuestro proyecto Ionic aunque tengamos otra versión global distinta.

Primera aplicación

Comenzando con Ionic

Creación de un proyecto Ionic

```
ionic start nombre_proyecto
```

- Lanza la creación de un nuevo proyecto Ionic
- Además, descarga todas las dependencias
  - Al trabajar con npm descarga todos los paquetes, tanto de Ionic como de terceros
- La utilidad nos pregunta una serie de aspectos de configuración

Creación de un proyecto Ionic (II)

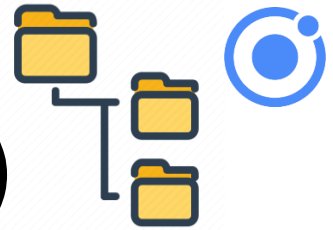
Aspecto	Explicación	Escogemos
Creation Wizard	Herramienta de creación vía Web	NO queremos usarlo
Framework	Framework en el cual basamos nuestra aplicación (Angular, Vue, React	`Angular`
Nombre del proyecto	Nombre que le daremos al proyecto Notación `PascalCase`	`PrimeraApp`
Starter template (plantilla de inicio)	Aplicaciones tipo con estructuras prediseñadas	`Blank`
Tipo de componente	Modelo de creación de componentes `NgModules` (clásico)| `Standalone` (nuevo)	`Standalone`
Cuenta de Ionic	Creación de una cuenta de desarrollador	Como queramos

Inspeccionando el código

Archivo/carpeta	Explicación
`package.json`	Gestión de las dependencias de proyecto. Debería incluir las librerías de Angular (20) + Ionic/Capacitator y otras dependencias
`node_modules`	Carpeta con las dependencias
`angular.json`	Configuración de la aplicación Angular

`tsconfig.json`
`tsconfig.app.json`
`tsconfig.spec.json`
	Configuración de TypeScript

Inspeccionando el código (II)



Archivo/carpeta	Explicación
`src`	Carpeta donde se encuentra el código de nuestra aplicación
`src/index.html`	Base de nuestra aplicación
`src/main.ts`	Arranque de nuestra aplicación (punto de salida)

Modelos de componentes en Angular

- Angular maneja dos modelos de componentes:

ngModules	*Standalone* components
Los módulos organizan la estructura de la aplicación agrupando componentes, directivas, pipes, servicios…	Se definen con la propiedad `standalone:true` en el decorador `@Component` Se importan otros componentes, directivas y módulos directamente en la configuración, sin depender de módulos Se enfoca en la modularidad a nivel de componente

Los componentes *standalone* se introducen a partir de Angular 14.

A partir de Angular 17 los componentes *standalone* se convierten en el modelo por defecto para la creación de componentes

Ejemplo de componente *standalone*

```ts
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-standalone-example',
  standalone: true,

  imports: [CommonModule],
  template: `<h1>¡Hola desde un componente standalone!</h1>`,
})
export class EjemploStandalone {}
```

Declaramos el componente standalone

Importación de elementos

Creación de componentes (Angular 17+)

- Creación de un componente *standalone* (por defecto)

```
ng generate component nombre-del-componente
```

- Creación de un componente asociado a un módulo

```
ng generate component nombre-del-componente
   --module=nombre-del-modulo
```

- Creación de un módulo

```
ng generate module nombre-del-modulo
```

Estructura de una aplicación Ionic

Comenzando con Ionic

Arranque de nuestra aplicación

```ts
import { bootstrapApplication } from '@angular/platform-browser';
import { RouteReuseStrategy, provideRouter, withPreloading, PreloadAllModules } from '@angular/router';
import { IonicRouteStrategy, provideIonicAngular } from '@ionic/angular/standalone';

import { routes } from './app/app.routes';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent, {
  providers: [
    { provide: RouteReuseStrategy, useClass: IonicRouteStrategy },
    provideIonicAngular(),
    provideRouter(routes, withPreloading(PreloadAllModules)),
  ],
});
```

Establecemos **AppComponent** como el componente de arranque (bootstrap) de nuestra aplicación

main.ts

El componente principal

```ts
import { Component } from '@angular/core';
import { IonApp, IonRouterOutlet } from '@ionic/angular/standalone';

@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html',
  styleUrls: ['app.component.scss'],

  imports: [IonApp, IonRouterOutlet],
})

export class AppComponent {
  constructor() {}
}
```

Decoramos clase AppComponent con Component

Especificación de selector, plantilla y estilos

Herramientas externas que queremos incluir en este componente

Declaramos y exportamos la clase AppComponent

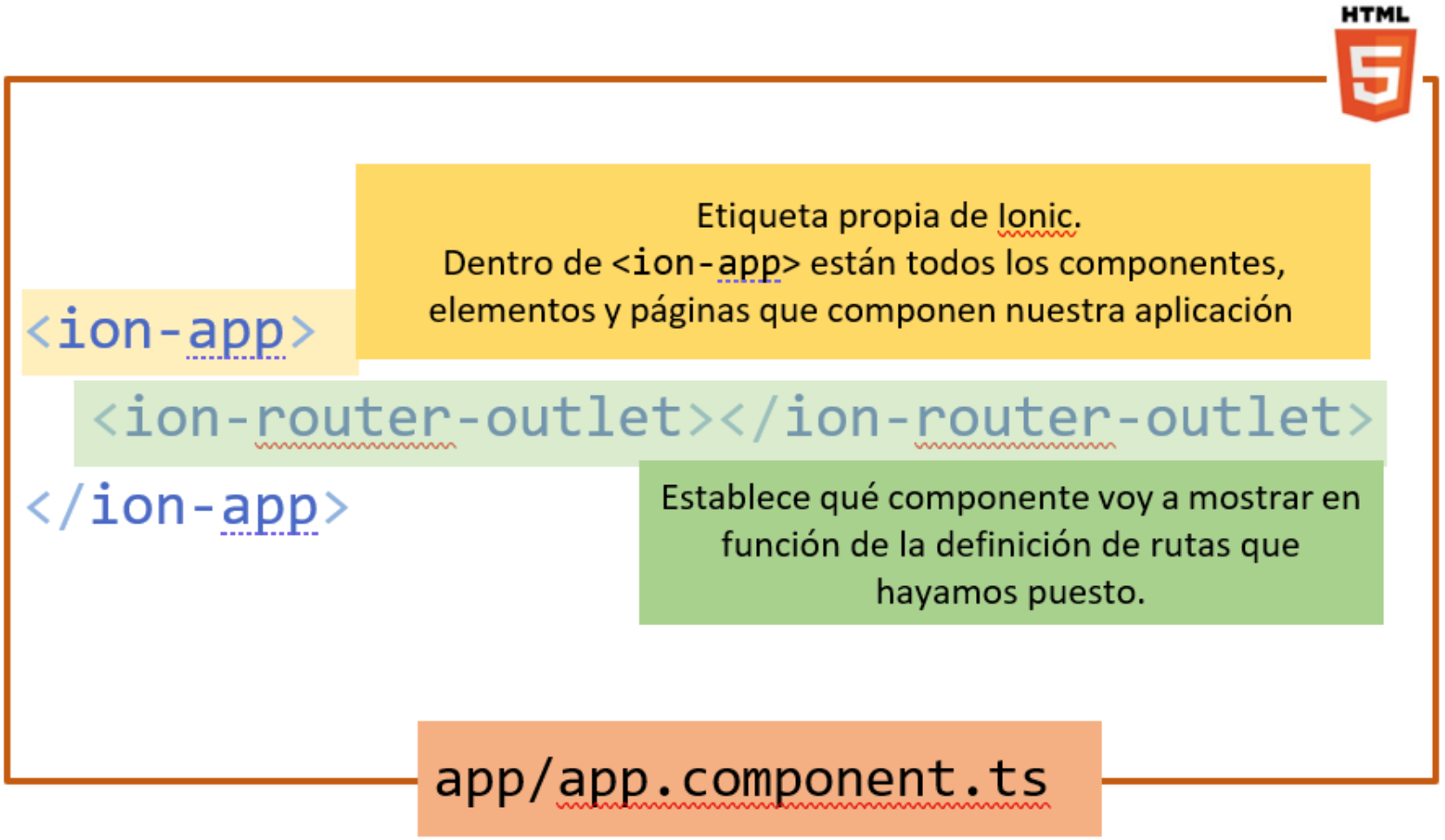
app/app.component.ts

Componentes de Angular

- Un componente controla una zona de pantalla que denominamos vista
- Los componentes de Angular están compuestos por:
  - Un archivo HTML donde se visualiza la interfaz (vista)
  - Un archivo CSS (o Sass/Less) donde incluimos los estilos
  - Un archivo de lógica TypeScript que define las interacciones

Las aplicaciones de Angular utilizan componentes para definir y controlar cada aspecto separado de la misma.

El componente principal - Vista



Definición de rutas

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

const routes: Routes = [
  {
    path: 'home',
    loadComponent: () => import('./home/home.page').
                then( m => m.HomePage)
  },
  {
    path: '',
    redirectTo: 'home',
    pathMatch: 'full'
  },
];
```

Si en la URL ponemos la ruta http://localhost:puerto/**home**, cargaremos el componente HomePage (con su propio enrutado)

Si en la URL ponemos la ruta raíz http://localhost:puerto/, redireccionamos a HomePage

app/app.routes.ts

Páginas en Ionic

- En Ionic, una página es un componente que representa una vista separada dentro de nuestra aplicación.

- Cada página suele asociarse a una ruta específica y está diseñada para ocupar toda la plantilla

Lazy loading

- `loadComponent()` permite la carga de componentes bajo demanda (*lazy loading*).
- Mejora el rendimiento al cargar los componentes sólo cuando se piden por el usuario, en lugar de cargarlas todas inicialmente.

Página HomePage

- Cada página de la App se mete en una carpeta separada (en este caso la carpeta home)
- Desarrollaremos nuestra app a base de ir creando las páginas asociadas a las distintas vistas que queramos crear
- La carpeta consta de los siguientes archivos

Archivo dentro de carpeta home	Explicación
`home.page.ts`	Definición del componente HomePage
`home.page.html`	Vista del componente
`home.page.scss`	Estilos del componente
`home.page.spec.ts`	Prueba del componente

Páginas y módulos

- En versiones anteriores de Ionic/Angular:
  - Cada página iba dentro de un módulo
  - Cada módulo tenía su propio enrutado, al cual se redirigía desde el módulo raíz.
- Esto hacía un poco más farragosa la gestión del enrutado.

El componente HomePage

```ts
@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],

  imports: [IonHeader, IonToolbar, IonTitle, IonContent],
export class HomePage {

  constructor() {}

}
```

No se utiliza porque hemos accedido a HomePage redireccionando a través de la ruta

Vista

Presentación

Componentes importados

app/home/home.page.ts

La vista de HomePage

```
<ion-header [translucent]="true">
  <ion-toolbar>
    <ion-title>  Blank </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content [fullscreen]="true">
  <ion-header collapse="condense">
    <ion-toolbar>
      <ion-title size="large">Blank</ion-title>
    </ion-toolbar>
  </ion-header>

  <div id="container">
    <strong>Ready to create an app?</strong>
    <p>Start with Ionic <a target="_blank" rel="noopener noreferrer"
         href="https://ionicframework.com/docs/components">UI Components</a></p>
  </div>
</ion-content>
```

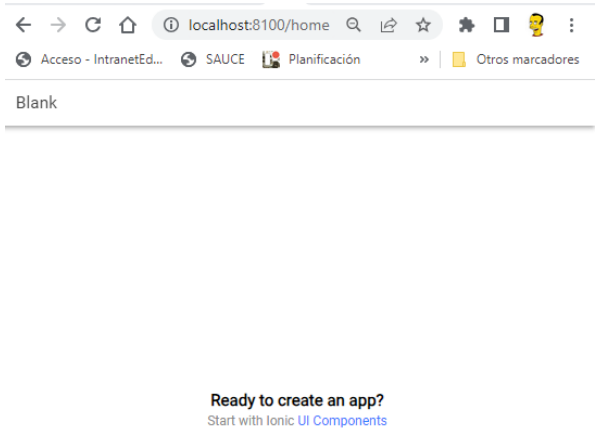
Ionic define un gran número de etiquetas específicas que traduce a HTML

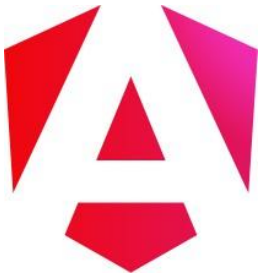
app/home/home.page.html

La vista de HomePage

```
ionic serve
```

- Lanza un navegador en un servidor local que nos permite ver la aplicación





Algunas novedades de Angular

Comenzando con Ionic

Nueva sintaxis para directivas

- A partir de Angular 17 de incluye una nueva sintaxis para plantillas llamada @-sintax
  - Simplifica y moderniza las tradicionales `*ngIf`, `*ngFor`, `*ngSwitch`
  - Evita necesidad de elementos “de relleno” `<ng-template>` y `ng-container>`

Traen mejoras en rendimiento (30-90%)

Estructura condicional @if

```
@if (show) {
  <span>Inside if</span>
} @else if (showAnotherIf) {
  <span>Inside else if</span>
} @else {
  <span>Inside else</span>
}
```

@if

```
<ng-container *ngIf="show; else elseTemplate">
  <span>Inside if</span>
</ng-container>
<ng-template #elseTemplate>
  <ng-container *ngIf="show; else anotherElseTemplate">
    <span>Inside else if</span>
  </ng-container>
  <ng-template #anotherElseTemplate>
    <span>Inside else</span>
  </ng-template>
</ng-template>
```

*ngIf

Estructura condicional @for

El bloque track debería llevar un ID único

```
<ul>
  @for (user of userList; track user.id; let i = $index){
    <li>{{ i }}. {{ user.name }}</li>
  } @empty {
    <li>Empty array</li>
  }
</ul>
```

Tratamiento especial en caso de que la lista esté vacía

@for

```
<ul>
  <ng-container *ngIf="userList.length; else emptyArray">
    <li *ngFor="let user of userList; index as i">{{i}}. {{ user.name }}</li>
  </ng-container>
  <ng-template #emptyArray>
    <li>Empty array</li>
  </ng-template>
</ul>
```

*ngFor

Estructura condicional @for

- El atributo track en un bucle @for sirve para mejorar el rendimiento del bucle.
- En track deberíamos indicar un atributo único dentro de los objetos de la lista que recorremos.
  - Angular lo usa para rastrear cada elemento
  - No usarlo puede penalizar el rendimiento
  - En casos excepcionales podemos:
    - No usar **track**
    - Tracker el **objeto completo**
    - **Usar el índice en el track** *(funciona bien si los objetos no cambian de posición)*

Estructura repetitiva @for

- Además, los bucles tienen variables implícitas al igual que en versiones anteriores de Angular

Variable	Descripción
`$index`	Proporciona el índice actual del elemento dentro de la colección
`$first`	Devuelve true si el elemento actual es el primero de la colección
`$last`	Devuelve true si el elemento actual es el último de la colección
`$even`	Devuelve true si el índice del elemento es par
`$odd`	Devuelve true si el índice del elemento es impar

Estructura repetitiva @switch

```
@switch (page) {
  @case (1) {
    <p>Viewing content of first page</p>
  }
  @case (2) {
    <p>Viewing content of second page</p>
  }
  @case (3) {
    <p>Viewing content of third page</p>
  }
  @default {
    <p>No page selected</p>
  }
}
```

@switch