



DECODED

ACADEMY

FULL STACK DEVELOPER BOOTCAMP

Desarrollamos
{ talento }

ANGULAR 2

Recursos

- [Awesome angular2](#)
- [cheatSheet](#)
- [Página oficial](#)
- [Documentación oficial](#)
- [Performance checklist](#)

Historia de Angular



Es un framework para aplicaciones web desarrollado en TypeScript de código abierto, mantenido por Google, y donde su objetivo es mantener una estructura con el paradigma M.V.C (Modelo Vista Controlador)

Angular en la actualidad se divide en dos proyectos.

1. Angular.js -> Basado en la versión 1 de angular actualmente se encuentra en la versión 1.6.8
2. Angular-> Basado en la versión 2

Existen grandes diferencias entre la versión 1 y 2 que cabe destacar la 2 versión esta enfocada al dearrollo de webcomponents, y utiliza typescript como lenguaje

Actualmente es mantenida por Google y podemos encontrar toda la información necesaria en su página web angular.io

Instalación

Para instalar el mismo sólo necesitamos instalar el paquete oficial denominado [Angular Cli](#)

```
$ npm install -g @angular/cli
```

Podemos comprobar si se ha instalado actualmente con:

```
$ ng version
```

Uso

Nuestra primera aplicación

Ejecutamos el siguiente comando para crear nuestra primera app en Angular

```
$ ng new [nombre-app]
```

Ahora para lanzarla sólo debemos ejecutar el comando.

```
$ ng serve --open
```

Vemos que nos arranca nuestra aplicación en nuestro localhost:4200.

Ahora vamos a crear nuestro primer componente

```
$ ng g component Home
```

Vemos que dentro del directorio src\app nos habrá creado el directorio prueba y dentro nos habrá creado

Ahora vemos que nos generan los siguientes archivos

- Home.component.css -> Aquí podemos disponer el css del componente.
- Home.component.html-> Aquí esta la vista del componente.
- Home.component.spec.ts -> Aquí se encuentra el archivo de pruebas.
- Home.component.ts->Este archivo dispondrá la lógica de nuestro componente

Ahora podemos añadir el nuevo componente sobre el componente base denominado app-component.html

```
<app-home></app-home>
```

Podemos pasar información desde la lógica del componente a la vista, para ello modificamos la lógica del componente

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-prueba',
  templateUrl: './prueba.component.html',
  styleUrls: ['./prueba.component.css']
})
export class PruebaComponent implements OnInit {
  nombre: string;
  constructor() {
    this.nombre = 'Xavi2';
  }
  ngOnInit() {
  }
}
```

Ahora modificamos la vista del componente para recibir la información.

```
<p>
  prueba works! {{ nombre }}
</p>
```

La información se interpola con la plantilla mediante {{ }}

Podemos instalar componentes para poder usarlos en nuestra aplicación, vamos a instalar **angular material**

```
$ npm install --save @angular/material @angular/cdk  
$ npm install --save @angular/animations
```

Ahora modificamos el archivo angular-cli.json

```
"styles": [  
  "styles.scss"  
]
```

Ahora debemos renombrar el archivo style.css a style.scss y importar el tema a utilizar.

```
@import '~@angular/material/prebuilt-themes/indigo-pink.css';

html, body {
  margin: 0px;
  padding: 0px;
}
body{
  background-color: #f0f0f0;
  font-family: Roboto, 'Helvetica New', 'Arial', sans-serif
}
```

Ahora debemos incorporar los componentes en nuestra aplicación para ello debemos modificar el archivo app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatCardModule } from '@angular/material/card';
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule,
    MatToolbarModule,
    MatCardModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Por último modificamos el archivo principal app.component.html

```
<mat-toolbar color="primary">
  <span></span>
</mat-toolbar>
<mat-card>
  <app-home></app-home>
</mat-card>
```

Creando una aplicación real

Vamos a crear una aplicación que conecte a la api de [The Movie DB](#)

Primero debemos crear nuestra aplicación.

```
$ ng new movieApp
```

Ahora vamos a crear nuestro primer componente que será un listado de películas.

```
$ ng g component films-list
```

Modificamos el archivo films-list.component.ts

```
ngOnInit() {  
    this.titulo="Listado de Películas:";  
    console.log("Listado de Peliculas cargado");  
}
```

Ahora modificamos film-list.component.html

```
<p>  
  {{titulo}}  
</p>
```

Ahora sólo modificamos el html de app.component.html

```
<div>
  <app-films-list></app-films-list>
</div>
```

Vamos a instalar BootStrap y ng Bootstrap para darle estilo a nuestra aplicación, para ello ejecutamos

```
$ npm install --save @ng-bootstrap/ng-bootstrap
```

Ahora modificamos el archivo .angular-cli.json para añadir las dependencias

```
"styles": [  
  "../node_modules/bootstrap/dist/css/bootstrap.min.css",  
  "styles.css"  
]
```

Ahora debemos incluir en el app.module.ts

```
import { NgbModule } from '@ng-bootstrap/ng-bootstrap';
@NgModule({
  declarations: [
    AppComponent,
    FilmsListComponent
  ],
  imports: [
    BrowserModule,
    NgbModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

Ahora generamos el routing para nuestra aplicación, para ello utilizaremos ng-cli como un módulo

```
$ ng g module app-routing
```

El cual nos genera la carpeta y archivo app-routing el cual modificamos

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { CommonModule } from '@angular/common';
import { FilmsListComponent } from '../films-list/films-list.component';

const routes: Routes=[  
  {  
    path :'',  
    component: FilmsListComponent,  
  },  
];  
  
@NgModule({  
  imports: [  
    RouterModule.forRoot(routes)  
  ],  
  exports:[  
    RouterModule  
  ],  
  declarations: []  
})  
export class AppRoutingModule { }>
```

Ahora modificamos el archivo app.module.ts

```
import { AppRoutingModule } from './app-routing/app-routing.module';
imports: [
  BrowserModule,
  NgbModule,
  AppRoutingModule,
]
```

En el componente principal debemos llamar al componente del routin dentro de app.component.html

```
<router-outlet></router-outlet>
```

Ahora sólo debemos realiza el linkado con el componente de Angular, para ello modificamos el componente nav

```
<a class="nav-link active" routerLink="/list">Active</a>
```

Un servicio es un proveedor de datos, que mantiene lógica de acceso a ellos, así como la operativa de la lógica de negocio. Los servicios son consumidos por los componentes, que delegarán en ellos la responsabilidad de acceder a la información y la realización de operaciones con los datos.

Para crear un servicio sólo debemos utilizar ng-cli

```
$ ng g service [nombre del servicio]  
// Si lo queremos añadir en un modulo determinado.  
$ ng g service [nombre del módulo]/[nombre-del-servicio]
```

Vamos a crear nuestro primer servicio

```
$ ng g service peliculas
```

Para poder utilizar nuestro Servicio debemos llamarlo en el módulo principal de nuestra App, primero importarlo y definirlo en el servicio

```
import { FilmListComponent } from './film-list/film-list.component';
providers: [
  FilmsService,
],
```

Para poder realizar la petición vamos a utilizar la librería [HttpClient](#)

Esta librería mapea la información de manera automática, para ello debemos llamarla en el módulo principal

```
import { HttpClientModule } from '@angular/common/http';
imports: [
  HttpClientModule,
]
```

La manera de poder utilizar la petición y que esta se quede desvinculada del hilo principal y no bloquee el navegador es mediante el uso de Observable

Esta clase es parte del proyecto [Reactive Extension o RxJS](#)

Esta clase implementa el patrón observador aplicado a streams de datos

Para poder utilizar la librería rxjs sólo debemos llamarlo dentro del servicio, así como el cliente HttpClient

```
import { Observable } from 'rxjs/Observable';
import { HttpClient } from '@angular/common/http';
```

Para insertar HttpClient debemos injectarlo por el constructor del servicio

```
constructor(private http: HttpClient) { }
```

Ahora vamos a realizar la petición desde el servicio

```
//petición api que devuelve el listado de películas más populares.  
public getFilms(): Observable[any[]]{  
    return this.http.get[any[]]  
(https://api.themoviedb.org/3/movie/popular?api\_key=\[api-key\]);  
}
```

Ahora vamos a utilizar nuestro servicio en nuestro componente, para ello debemos llamar al mismo

```
import { FilmsService } from './films.service';
```

Ahora debemos injectarlo utilizando el constructor

```
//Definimos un array donde guardaremos el resultado de la petición.  
peliculas: Array[any];  
constructor( private filmService : FilmsService) { }
```

Generamos un método para llamar al servicio en el mismo componente

```
getFilms () {  
    this.filmService.getFilms () .subscribe (  
        result => {  
            this.peliculas = result.results;  
        },  
        error => {  
            console.log (error);  
        }  
    )  
}
```

Ahora vamos a modificar el template para poder presentar las películas por pantalla

```
<h1>{{titulo}}</h1>
<ul>
  <li *ngFor = "let pelicula of peliculas">
    <div>
      <h4>{{pelicula.title}}</h4>
    </div>
  </li>
</ul>
```

Utilizamos `*ngFor` para iterar sobre un array con los datos y presentar en la vista.

La naturaleza asíncrona de las comunicaciones entre máquinas se implementó mediante callbacks en JavaScript. Esta forma de programar degenera en código difícil de mantener. Con el tiempo el patrón promesa se impuso, y en AngularJS 1.x es la manera recomendada de programar. Pero las promesas también tienen sus limitaciones, y ahí aparecen los observables.

Components

El decorador `@Component` identifica la clase inmediatamente debajo de ella como una clase de componente y especifica sus metadatos. En el código de ejemplo siguiente, puede ver que `HeroListComponent` es solo una clase, sin notación especial de Angular. No es un componente hasta que lo marques como uno con el decorador `@Component`.

```
@Component({
  selector: 'app-hero-list',
  templateUrl: './hero-list.component.html',
  providers: [ HeroService ]
})
export class HeroListComponent implements OnInit {
  /* . . . */
}
```

- **selector:** un selector de CSS que le dice a Angular que cree e inserte una instancia de este componente donde encuentre la etiqueta correspondiente en la plantilla HTML. Por ejemplo, si el HTML de una aplicación contiene </ app-hero-list>, Angular inserta una instancia de la vista HeroListComponent entre esas etiquetas.
- **templateUrl:** la dirección relativa al módulo de la plantilla HTML de este componente. Alternativamente, puede proporcionar la plantilla HTML en línea, como el valor de la propiedad de la plantilla. Esta plantilla define la vista de host del componente.
- **template:** Una plantilla para un componente angular. Si se proporciona, no proporcione un archivo de plantilla usando templateUrl.

- **styleUrls**

Una o más URL para archivos que contienen hojas de estilo CSS para usar en este componente.

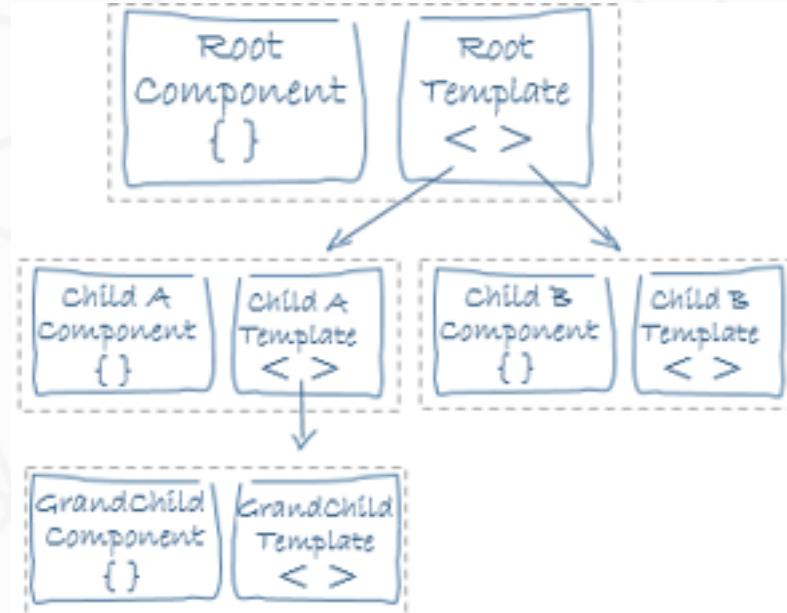
- **styles**

Una o más hojas de estilo CSS en línea para usar en este componente.

- **providers** una variedad de proveedores de inyección de dependencia para los servicios que requiere el componente. En el ejemplo, esto le dice a Angular que el constructor del componente requiere una instancia de HeroService para mostrar la lista de héroes.

Se define la vista de un componente con su plantilla complementaria. Una plantilla es una forma de HTML que le dice a Angular cómo renderizar el componente.

Las vistas generalmente se organizan jerárquicamente, lo que le permite modificar o mostrar y ocultar secciones o páginas de IU enteras como una unidad. La plantilla asociada inmediatamente con un componente define la vista de host de ese componente. El componente también puede definir una jerarquía de vista, que contiene vistas incrustadas, alojadas por otros componentes.



Modules

Las aplicaciones de angular son modulares y Angular tiene su propio sistema de modularidad llamado NgModules.

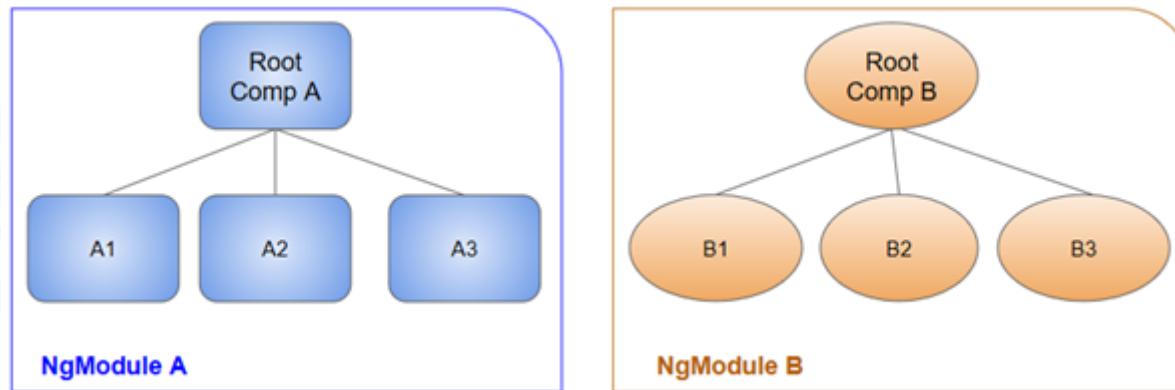
Un NgModule es un contenedor para un bloque de código dedicado a un dominio de aplicación, un flujo de trabajo o un conjunto de capacidades estrechamente relacionadas.

Puede contener componentes, proveedores de servicios y otros archivos de código cuyo alcance está definido por el NgModule que lo contiene. Puede importar funcionalidades que se exportan desde otros NgModules, y exportar funcionalidades seleccionadas para su uso por otros NgModules.

```
import { NgModule }           from '@angular/core';
import { BrowserModule }    from '@angular/platform-browser';
import { AppComponent }      from './components/app.component.ts';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

- **declarations**: los componentes, las directivas y las pipes que pertenecen a este NgModule.
- **exports**: el subconjunto de declaraciones que deberían ser visibles y utilizables en las plantillas de componentes de otros NgModules.
- **imports**: otros módulos cuyas clases exportadas son necesarias para las plantillas de componentes declaradas en este NgModule.
- **providers**: creadores de servicios que este NgModule contribuye a la recopilación global de servicios; se vuelven accesibles en todas las partes de la aplicación. (También puede especificar proveedores en el nivel del componente, que a menudo se prefiere).
- **bootstrap**: la vista principal de la aplicación, llamada componente raíz, que aloja todas las demás vistas de la aplicación. Solo el NgModule raíz debe establecer esta propiedad de arranque.

- NgModules y components



Template Syntax

Interpolation (Template expressions)

- {{ // sentencia js}}

Puedes utilizar la interpolación para injectar texto entre las etiquetas de elementos HTML y/o dentro de las asignaciones de atributos.

```
<h3>
  {{title}}
  
</h3>
```

Angular evalúa todas las expresiones en llaves dobles, convierte los resultados de expresión en cadenas y los vincula con cadenas literales contiguas. Finalmente, asigna este resultado interpolado compuesto a un elemento o propiedad directiva.

```
<!-- "The sum of 1 + 1 is 2" -->
<p>The sum of 1 + 1 is {{1 + 1}}</p>
```

Se prohíben las expresiones de JavaScript que tienen o promueven efectos secundarios, incluyendo: - asignaciones (`=`, `+ =`, `- =`, ...)

- `new`
- expresiones de encadenamiento con `;` o `,`
- operadores de incremento y decremento (`++` y `--`).

Otras diferencias notables de la sintaxis de JavaScript incluyen:

- no soporte para los operadores bit a bit `|` y `&`
- nuevos operadores de expresión de plantilla, como `!?,.` y `!`

Expression context

El contexto de expresión es típicamente la instancia del componente. En los siguientes fragmentos, el título dentro de llaves dobles se refieren a las propiedades del componente de aplicación.

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

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    <h2>My favorite hero is: {{myHero}}</h2>
  `
})
export class AppComponent {
  title = 'Tour of Heroes';
  myHero = 'Windstorm';
}
```

Property binding [attribute]

Escriba un *property binding* de plantilla para establecer una propiedad de un elemento de vista. El enlace establece la propiedad al valor de una expresión de plantilla.

```
<img [src]="heroImageUrl">
```

Los siguientes lineas hacen lo mismo:

```
<p> is the <i>property bound</i> image.</p>
```

La interpolación es una alternativa conveniente al *property binding* en muchos casos.

Al representar valores de datos como cadenas, no hay ninguna razón técnica para preferir una forma a la otra. Te inclinas hacia la legibilidad, lo que tiende a favorecer la interpolación.

Al establecer una propiedad de un elemento a un valor no string, debe usar el *property binding* [].

The pipe operator |

El resultado de una expresión puede requerir alguna transformación antes de que esté listo para usarlo en una vinculación. Por ejemplo, puede mostrar un número como moneda, forzar texto a mayúsculas o filtrar una lista y ordenarla.

```
<div>Title through uppercase pipe: {{title | uppercase}}</div>
```

El operador de navegación segura (?.)

El operador de navegación segura angular (?.) Es una forma fluida y conveniente de protegerse contra los valores nulos e indefinidos en las rutas de las propiedades. Aquí está, protegiendo contra una vista, render failure si el actualHero es nulo.

```
<div>The current hero's name is {{currentHero?.name}}</div>
```

One-way binding

Las personas a menudo describen *property binding* o la *interpolation* como *one-way data binding* porque fluye un valor en una dirección, desde la propiedad de datos de un componente hasta la propiedad de un elemento de destino.

No puede usar el enlace de propiedad para extraer valores del elemento de destino. No puede vincularse a una propiedad del elemento objetivo para leerlo. Solo puedes configurarlo.

No puede usar el enlace de propiedad para llamar a un método en el elemento de destino.

Event binding (event)

La única forma de conocer una acción del usuario es escuchar ciertos eventos como pulsaciones de teclas, movimientos del mouse, clics , etc...

El nombre entre paréntesis, (clic), identifica el evento.

```
<button (click)="onSave () ">Save</button>  
  
<!-- hace lo mismo que el prefijo on-* -->  
<button on-click="onSave () ">On Save</button>
```

En el *property context* se expone un variable llamada **\$event** que sirve para acceder al evento nativo generado:

```
<button (click)="onSave($event)">On Save</button>
```

Este código establece la propiedad del valor del cuadro de entrada vinculándose a la propiedad del nombre. Para escuchar los cambios en el valor, el código se une al evento de entrada del cuadro de entrada. Cuando el usuario realiza cambios, se genera el evento de entrada (input) y el enlace ejecuta la declaración dentro de un contexto que incluye el objeto de evento DOM \$event.

```
<input [value]="currentHero.name"  
       (input)="currentHero.name=$event.target.value" >
```

Template reference variables #var

Una template reference a menudo es una referencia a un elemento DOM dentro de una plantilla. También puede ser una referencia a un componente angular o directiva o un componente web.

```
<input #phone placeholder="phone number">
```

```
<button (click)="callPhone(phone.value)">Call</button>
```

Two-way binding [(...)]

Angular ofrece una sintaxis de enlace de datos bidireccional especial para este propósito, [(x)]. La sintaxis [(x)] combina los corchetes del enlace de propiedad, [x], con los paréntesis del enlace de evento, (x).

Al desarrollar formularios de entrada de datos, a menudo ambos muestran una propiedad de datos y actualizan esa propiedad cuando el usuario realiza cambios.

El enlace de datos bidireccional con la directiva NgModel lo hace fácil.

```
<input [(ngModel)]="currentHero.name">
```

Antes de utilizar la directiva ngModel en un enlace de datos bidireccional, debe importar el FormsModule y agregarlo a la lista de importaciones del NgModule.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms'; // <--- JavaScript import
from Angular

/* Other imports */

@NgModule({
  imports: [
    BrowserModule,
    FormsModule // <--- import into the NgModule
  ],
  /* Other module metadata */
})
export class AppModule { }
```

Resumen

Data direction	Syntax	Type
One-way from data source to view target	<code>{{expression}}</code> <code>[target]="expression"</code> <code>bind-target= "exp"</code>	Interpolation Property Attribute Class Style
One-way from view target to data source	<code>(target)="statement"</code> <code>on-target="statement"</code>	Event
Two-way	<code>[(target)]="expression"</code> <code>bindon-target="expr"</code>	Two-way

Directivas

Hay tres tipos de directivas en Angular:

- Componentes: directivas con una plantilla.
- Directivas estructurales: cambia el diseño DOM añadiendo y eliminando elementos DOM.
- Directivas de atributo: cambian la apariencia o el comportamiento de un elemento, componente u otra directiva.

Custom directive

Vamos a crear un directiva de atributo:

```
<p appHighlight>Highlight me!</p>
```

Crea el archivo de clase directiva en una ventana de terminal con este comando CLI:

```
$ ng generate directive highlight
```

Se crea:

```
// src/app/highlight.directive.ts
import { Directive } from '@angular/core';

@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
  constructor() { }
}
```

Así podemos poner el background amarillo en los elementos que utilicen la directiva appHighlight:

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
  constructor(el: ElementRef) {
    el.nativeElement.style.backgroundColor = 'yellow';
  }
}
```

Built-in attribute directives

- NgClass: agrega y elimina un conjunto de clases CSS
- NgStyle: agrega y elimina un conjunto de estilos HTML
- NgModel: enlace de datos bidireccional a un elemento de formulario HTML

NgClass

Sin ngClass:

```
<div class="{{isSaveable ? 'saveable': ''}} special">This div is  
initially saveable</div>
```

Con ngClass:

```
<div [ngClass]="currentClasses">This div is initially saveable,  
unchanged, and special</div>  
  
currentClasses = {  
  'saveable': this.isSaveable,  
};
```

Un *class binding* es una buena forma de agregar o eliminar una sola clase.

```
<!-- reset/override all class names with a binding -->
<div class="bad curly special" [class]="badCurly">Bad curly</div>
```

```
<!-- toggle the "special" class on/off with a property -->
<div [class.special]="isSpecial">The class binding is special</div>
```

```
<!-- binding to `class.special` trumps the class attribute -->
<div class="special"
      [class.special]!="!isSpecial">This one is not so special</div>
```

NgStyle

```
<div [ngStyle]="currentStyles">  
  This div is initially italic, normal weight, and extra large (24px).  
</div>  
  
currentStyles: {};  
setCurrentStyles() {  
  // CSS styles: set per current state of component properties  
  this.currentStyles = {  
    'font-style': this.canSave ? 'italic' : 'normal',  
    'font-weight': !this.isUnchanged ? 'bold' : 'normal',  
    'font-size': this.isSpecial ? '24px' : '12px'  
  };  
}
```

Si bien ésta es una buena forma de establecer un solo estilo, generalmente se prefiere la directiva NgStyle al configurar varios estilos en línea al mismo tiempo.

```
<button [style.color]="isSpecial ? 'red': 'green'">Red</button>
<button [style.background-color]="canSave ? 'cyan': 'grey'">Save</button>
```

```
<button [style.fontSize.em]="isSpecial ? 3 : 1" >Big</button>
<button [style.fontSize.%]!="!isSpecial ? 150 : 50" >Small</button>
```

Built-in structural directives

Las directivas estructurales son responsables del layout. Forman o remodelan la estructura del DOM, generalmente agregando, eliminando o manipulando elementos.

Un asterisco (*) precede al nombre del atributo de directiva como en este ejemplo.

```
<app-hero-detail *ngIf="isActive"></app-hero-detail>
```

- Nglf: agrega o elimina condicionalmente un elemento del DOM
- NgSwitch: un conjunto de directivas que cambian entre vistas alternativas
- NgForOf: repite una plantilla para cada elemento en una lista

ngIf

```
<app-hero-detail *ngIf="isActive"></app-hero-detail>
```

Cuando la expresión isActive devuelve un valor *truthy*, NgIf agrega HeroDetailComponent al DOM. Cuando la expresión es *falsy*, NgIf elimina HeroDetailComponent del DOM, destruyendo ese componente y todos sus subcomponentes.

La directiva `ngIf` a menudo se usa para evitar acceder a null/undefined. Angular emitirá un error si una expresión anidada intenta acceder a una propiedad de null.

```
<div *ngIf="currentHero">Hello, {{currentHero.name}}</div>
```

ngForOf / ngFor

Define cómo se debe mostrar un solo elemento y le dices a Angular que use ese bloque como plantilla para representar cada elemento en la lista.

```
<div *ngFor="let hero of heroes">{{hero.name}}</div>
```

*ngFor con índice:

```
<div *ngFor="let hero of heroes; let i=index">{{i + 1}} - {{hero.name}}</div>
```

*ngFor con trackBy:

La directiva NgForOf puede funcionar mal, especialmente con listas grandes. Un pequeño cambio en un elemento, un elemento eliminado o un elemento agregado puede desencadenar una cascada de manipulaciones de DOM.

```
// src/app/app.component.ts
trackByHeroes(index: number, hero: Hero): number { return hero.id; }

<!-- src/app/app.component.html -->
<div *ngFor="let hero of heroes; trackBy: trackByHeroes">
  {{hero.id}} {{hero.name}}
</div>
```

ngSwitch

NgSwitch es como la declaración de switch de JavaScript. Puede mostrar un elemento entre varios elementos posibles, en función de una condición de cambio. Angular pone solo el elemento seleccionado en el DOM.

NgSwitch es en realidad un conjunto de tres directivas que cooperan: NgSwitch, NgSwitchCase y NgSwitchDefault como se ve en este ejemplo.

```
<div [ngSwitch]="currentHero.emotion">
  <app-happy-hero *ngSwitchCase="'happy'" [hero]="currentHero">
  </app-happy-hero>
  <app-sad-hero *ngSwitchCase="'sad'" [hero]="currentHero">
  </app-sad-hero>
  <app-confused-hero *ngSwitchCase="'confused'" [hero]="currentHero">
  </app-confused-hero>
  <app-unknown-hero *ngSwitchDefault [hero]="currentHero">
  </app-unknown-hero>
</div>
```

Lifecycle Hooks

Las instancias de directivas y componentes tienen un ciclo de vida ya que Angular las crea, las actualiza y las destruye. Los desarrolladores pueden aprovechar los momentos clave en ese ciclo de vida.

```
import { Component, OnInit } from '@angular/core';
@Component ({
  selector: 'peekABoo'
})
export class PeekABoo implements OnInit {
  constructor(private logger: LoggerService) { }

  ngOnInit() { this.logIt(`OnInit`); }

  logIt(msg: string) {
    this.logger.log(`#${nextId++} ${msg}`);
  }
}
```


- **ngOnChanges()** Se invoca cuando Angular (re) establece propiedades de entrada enlazadas a datos. El método recibe un objeto SimpleChanges de valores de propiedad actuales y anteriores. Se invoca antes de ngOnInit () y cada vez que cambian una o más propiedades de entrada enlazadas a datos.
- **ngOnInit ()** Inicialice la directiva / componente después de que Angular primero muestre las propiedades de datos enlazados y establezca las propiedades de entrada de la directiva / componente. Se llama una vez, después de la primera ngOnChanges ().

- **ngDoCheck ()** Detecta y actúa sobre los cambios que Angular no puede o no detectará por sí mismo. Se invoca durante cada ejecución de detección de cambios, inmediatamente después de ngOnChanges () y ngOnInit ().
- **ngAfterContentInit ()** Responda después de que los proyectos externos tengan contenido externo en la vista del componente / la vista en la que se encuentra una directiva. Se llama una vez después de la primera ngDoCheck () .

- **ngAfterContentChecked ()** Responda después de que Angular verifique el contenido proyectado en la directiva / componente. Se invoca después de ngAfterContentInit () y de cada ngDoCheck posterior ().
- **ngAfterViewInit ()** Responda después de que Angular inicialice las vistas del componente y las vistas secundarias / la vista en la que se encuentra una directiva. Se invoca una vez después de la primera ngAfterContentChecked () .

- **ngAfterViewChecked ()** Responda después de que Angular verifique las vistas del componente y las vistas secundarias / la vista en la que se encuentra una directiva. Se invoca después de ngAfterViewInit y de cada ngAfterContentChecked posterior ().
- **ngOnDestroy ()** Limpieza justo antes de que Angular destruya la directiva / componente. Anule la suscripción de Observables y separe los controladores de eventos para evitar fugas de memoria. Llamado justo antes de que Angular destruya la directiva / componente.

Ejemplos prácticos

Obtener la entrada del usuario desde una variable de referencia:

keyup loop-back component

```
@Component({  
  selector: 'app-loop-back',  
  template: `  
    <input (keyup)="text = $event.target.value" >  
    <p>{{text}}</p>  
  `,  
})  
export class LoopbackComponent { }
```

```
@Component({  
  selector: 'app-loop-back',  
  template: `  
    <input [(ngModel)]="text" >  
    <p>{{text}}</p>  
  `,  
})  
export class LoopbackComponent {  
}
```

```
@Component({  
  selector: 'app-loop-back',  
  template: `  
    <input #box (keyup)="0">  
    <p>{ {box.value} }</p>  
  `,  
})  
export class LoopbackComponent {}
```

Esto no funcionará a menos que te escuches a un evento.

Angular actualiza los enlaces (y por lo tanto la pantalla) solo si la aplicación hace algo en respuesta a eventos asíncronos, como las pulsaciones de teclas.

Filtrado de eventos key con \$event:

```
@Component({
  selector: 'app-key-up3',
  template: `

    <input (keyup.enter)="onEnter($event.target.value)">

    <p>{ value }</p>
  `

})
export class KeyUpComponent_v3 {
  value = '';
  onEnter(text: string) { this.value = text; }
}
```

Filtrado de eventos key con #ref:

```
@Component({
  selector: 'app-key-up3',
  template: `
    <input #box>
    <button (click)="onEnter(box.value)">
      <p>{{value}}</p>
    `
})
export class KeyUpComponent_v3 {
  value = '';
  onEnter(text: string) { this.value = text; }
}
```

Validación de formularios:

```
<input id="name"
       name="name"
       class="form-control"
       pattern="^[^A-Za-z0-9]@"
       required
       minlength="4"
       appForbiddenName="bob"
       [(ngModel)]="hero.name"
       #name="ngModel">
<div *ngIf="name.invalid && (name.dirty || name.touched)"
      class="alert alert-danger">
  <div *ngIf="name.errors.required">
    Name is required.
  </div>
  <div *ngIf="name.errors.minLength">
    Name must be at least 4 characters long.
  </div>
  <div *ngIf="name.errors.forbiddenName">
    Name cannot be Bob.
  </div>
</div>
```

Built in validators

[docs](#)

Validation CSS classes

Angular refleja automáticamente muchas propiedades de control en el elemento de control de formulario como clases de CSS. Puede usar estas clases para diseñar elementos de control de formulario de acuerdo con el estado del formulario. Las siguientes clases son actualmente compatibles:

.ng-valid .ng-invalid .ng-pending .ng-pristine .ng-dirty .ng-untouched .ng-touched

Interacción entre componentes

Pasando datos de padres a hijos



```
import { Component } from '@angular/core';
@Component({
  selector: 'app-hero',
  template:
`<div>
  <app-hero-child *ngFor="let hero of heroes" [hero]="hero"></app-
hero-child>
</div>
`})
export class HeroComponent {
  heroes: Array<object> = [
    {
      name: 'Superman',
      age: 34, superpower: 'all',
      img: './assets/superman.jpg'
    },
    {
      name: 'Batman',
      age: 34, superpower: 'money',
      img: './assets/Batman.jpg'
    },
    {
      name: 'super Lopez',
      age: 34, superpower: 'ninguno',
      img: './assets/Lopez.jpg'
    }
  ]}
```

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'app-hero-child',
  template: `
    <div>
      
      <h3>{{hero.name}} says:</h3>
      <p>I, {{hero.name}}, am at your service.</p>
      <p> SUPERPOWERS: {{ hero.superpower }}</p>
    </div>`})
export class HeroChildComponent {
  @Input() hero: object;
}
```

Escuchando eventos del hijo



```
import { Component }      from '@angular/core';
@Component({
  selector: 'app-vote-taker',
  template: `
    <h2>Should mankind colonize the Universe?</h2>
    <h3>Agree: {{agreed}}, Disagree: {{disagreed}}</h3>
    <app-voter *ngFor="let voter of voters"
      [name]="voter"
      (voted)="onVoted($event)">
    </app-voter>
  `
})
export class VoteTakerComponent {
  agreed = 0;
  disagreed = 0;
  voters = ['Mr. IQ', 'Ms. Universe', 'Bombasto'];

  onVoted(agreed: boolean) {
    agreed ? this.agreed++ : this.disagreed++;
  }
}
```

```
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
  selector: 'app-voter',
  template: `
    <h4>{ name }</h4>
    <button (click)="vote(true)" [disabled]="didVote">Agree</button>
    <button (click)="vote(false)" [disabled]="didVote">Disagree</button>
  `
})
export class VoterComponent {
  @Input() name: string;
  @Output() voted = new EventEmitter<boolean>();
  didVote = false;

  vote(agreed: boolean) {
    this.voted.emit(agreed);
    this.didVote = true;
  }
}
```

Services

Un servicio es una categoría que abarca cualquier valor, función o característica que una aplicación necesite.

Angular distingue los componentes de los servicios para aumentar la modularidad y la reutilización.

- Al separar la funcionalidad relacionada con la vista de un componente de otros tipos de procesamiento, puede hacer que sus clases de componentes sean ligeras y eficientes. Idealmente, el trabajo de un componente es habilitar la experiencia del usuario y nada más.
- Un componente no debería necesitar definir cosas como cómo recuperar datos del servidor, validar la entrada del usuario o iniciar sesión directamente en la consola. En cambio, puede delegar tales tareas a los servicios.

```
// src/app/logger.service.ts (class)
@Injectable()
export class Logger {
  log(msg: any) { console.log(msg); }
  error(msg: any) { console.error(msg); }
  warn(msg: any) { console.warn(msg); }
}
```

Debe registrar al menos un proveedor de cualquier servicio que va a utilizar. Puede registrar proveedores en módulos o en componentes.



```
// src/app/app.module.ts (module providers)
@NgModule({
  imports: [
    BrowserModule,
  ],
  providers: [
    BackendService,
    Logger
  ],
  declarations: [
    AppComponent,
  ],
  bootstrap: [ AppComponent ]
```

o en un componente localmente:

```
// src/app/hero-list.component.ts (component providers)
@Component({
  selector: 'app-hero-list',
  templateUrl: './hero-list.component.html',
  providers: [ Logger ]
})
```


HttpClient

La mayoría de las aplicaciones de aplicaciones para el usuario se comunican con servicios de back-end a través del protocolo HTTP. Los navegadores modernos admiten dos API diferentes para realizar solicitudes HTTP: la interfaz XMLHttpRequest y la API fetch () .

El HttpClient en @angular/common/http ofrece una API HTTP cliente simplificada, que se basa en la interfaz XMLHttpRequest expuesta por los navegadores.

Los beneficios de HttpClient incluyen: características de testing, objetos de solicitud y respuesta tipeados, intercepción de solicitud y respuesta, API observables y manejo simplificado de errores.

Antes de poder usar el HttpClient se debe importar el HttpClientModule:

```
import { NgModule }           from '@angular/core';
import { BrowserModule }    from '@angular/platform-browser';
import { HttpClientModule }  from '@angular/common/http';

@NgModule({
  imports: [
    BrowserModule,
    // import HttpClientModule after BrowserModule.
    HttpClientModule,
  ],
  declarations: [
    AppComponent,
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule {}
```

Después de haber importado HttpClientModule en el AppModule, puede injectar el HttpClient en una clase de aplicación como se muestra en el siguiente ejemplo de ConfigService.

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

@Injectable()
export class ConfigService {
  constructor(private http: HttpClient) { }
}
```

Router

El Router permite la navegación de una vista a la siguiente al cambiar la ruta del navegador sin recargar la página.

- Acceder a una URL en la barra de direcciones y el navegador navega a la página correspondiente.
- Hacer clic en los enlaces en la página y el navegador navega a una nueva página.
- Hacer clic en los botones de avance y retroceso del navegador y el navegador navega hacia atrás y hacia adelante a través del historial de las páginas que ha visto.

- Puede interpretar una URL del navegador como una instrucción para navegar a una vista generada por el cliente.
- Puede pasar parámetros opcionales junto con el componente de vista de soporte que lo ayuda a decidir qué contenido específico presentar.
- Puede enlazar el enrutador a los enlaces en una página y navegará a la vista de aplicación adecuada cuando el usuario haga clic en un enlace.
- Puede navegar de forma imperativa cuando el usuario hace clic en un botón, selecciona de un cuadro desplegable o en respuesta a algún otro estímulo de cualquier fuente.
- Y el enrutador registra la actividad en el diario de historial del navegador para que los botones de avance y retroceso también funcionen.

Router outlet

```
<router-outlet></router-outlet>
```

El RouterOutlet es una directiva de la biblioteca del enrutador que se utiliza como un componente.

Actúa como un marcador de posición que marca el lugar en la plantilla donde el enrutador debe mostrar los componentes.

Configuración

Cuando cambia la URL del navegador, el enrutador busca una Ruta correspondiente desde la cual puede determinar el componente que se mostrará en el router-outlet

```
import { CrisisListComponent, HeroDetailComponent, HeroListComponent, PageNotFoundComponent }  
from './components';  
const appRoutes: Routes = [  
  { path: 'crisis-center', component: CrisisListComponent },  
  { path: 'hero/:id', component: HeroDetailComponent },  
  {  
    path: 'heroes',  
    component: HeroListComponent,  
    data: { title: 'Heroes List' }  
  },  
  { path: '',  
    redirectTo: '/heroes',  
    pathMatch: 'full'  
  },  
  { path: '**', component: PageNotFoundComponent }  
];  
@NgModule({  
  imports: [  
    RouterModule.forRoot(  
      appRoutes,  
      { enableTracing: true } // <-- debugging purposes only  
    )  
    // other imports here  
  ],  
  ...  
})  
export class AppModule { }
```

Generación de un módulo que gestione las rutas:

```
$ ng generate module app-routing --module app --flat
```

```
import { NgModule }           from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { CrisisListComponent } from './crisis-list/crisis-
list.component';
import { HeroListComponent }   from './hero-list/hero-list.component';
import { PageNotFoundComponent } from './page-not-found/page-not-
found.component';
const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'heroes', component: HeroListComponent },
  { path: '', redirectTo: '/heroes', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent }
];
@NgModule({
  imports: [
    RouterModule.forRoot(appRoutes, { enableTracing: true }) // <-- debugging purposes only
  ],
  exports: [
    RouterModule
  ]
})
export class AppRoutingModule {}
```

```
//app.module.ts
import { NgModule }           from '@angular/core';
import { BrowserModule }      from '@angular/platform-browser';
import { FormsModule }         from '@angular/forms';
import { AppComponent }        from './app.component';
import { AppRoutingModule }    from './app-routing.module';
import { CrisisListComponent } from './crisis-list/crisis-
list.component';
import { HeroListComponent }   from './hero-list/hero-list.component';
import { PageNotFoundComponent }from './page-not-found/page-not-
found.component';
@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    AppRoutingModule
  ],
  declarations: [
    AppComponent,
    HeroListComponent,
    CrisisListComponent,
    PageNotFoundComponent
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Router links

A la URL se podría llegar directamente desde la barra de direcciones del navegador. Pero la mayoría de las veces navega como resultado de alguna acción del usuario, como el clic de una etiqueta de ancla.

```
<nav>
  <a routerLink="/crisis-center" routerLinkActive="active">Crisis
  Center</a>
  <a routerLink="/heroes" routerLinkActive="active">Heroes</a>
</nav>
<router-outlet></router-outlet>
```

Route parameters

Los parámetros se indican en la configuración precedidos de : y puedes ser opcionales indicandolo con el ?

```
const appRoutes: Routes = [
  // localhost:4200/hero/123
  { path: 'hero/:id', component: HeroDetailComponent },
  // localhost:4200/crisis-center/
  // or localhost:4200/crisis-center/nuclear
  { path: 'crisis-center/:why?', component: CrisisListComponent },
];

```

Router Services

El Servicio **ActivatedRoute** nos da acceso a los parámetros de la url

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
export class HeroComponent implements OnInit {
  id: number;
  constructor(
    private route: ActivatedRoute
  ) {}
  ngOnInit() {
    this.id = +this.route.snapshot.paramMap.get('id');
  }
}
```

También podemos suscribirnos por si el parámetro cambiara estando en la misma vista:

```
import { Component, OnInit, OnDestroy } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
export class HeroListComponent implements OnInit,OnDestroy {
  id: number;
  paramsSubscription: any;

  constructor(
    private route: ActivatedRoute
  ) {}

  ngOnInit() {
    this.paramsSubscription = this.route.paramMap.subscribe(params => {
      this.id = +params.get('id');
    });
  }
  ngOnDestroy() {
    this.paramsSubscription.unsubscribe();
  }
}
```

El Servicio Router proporciona las capacidades de navegación y manipulación de url.

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
export class HeroComponent implements OnInit, OnDestroy {
  constructor(
    private router: Router
  ) {}
  gotoHeroes() {
    this.router.navigate(['/heroes'])
      .then(success => console.log('navigation success?', success))
      .catch(console.error);
  }
}
```

Guia de estilo

Responsabilidad única

- [wikipedia](#)

Regla del uno:

- Defina una cosa, como un servicio o componente, por archivo.
- Considere limitar los archivos a 400 líneas de código.

Funciones pequeñas:

- Defina las funciones pequeñas
- Considere limitarlas a no más de 75 líneas.

Naming

- Siga un patrón que describa la característica del símbolo y luego su tipo. El patrón recomendado es feature.type.ts. (Por ejemplo: hero-list.component.ts)
- Use guiones para separar palabras en el nombre descriptivo.
- Use puntos para separar el nombre descriptivo del tipo.

Estructura de la Aplicación

- LIFT (Locate Identify Flat Try to be DRY):
 - Locate: Haga que la localización del código sea intuitiva, simple y rápida.
 - Identify: Asigne un nombre al archivo para que sepa al instante qué contiene y qué representa.
 - Flat: Mantenga una estructura de carpetas horizontal dentro de lo posible.
 - DRY: Don't Repeat Yourself