



Rubén Magallón



Casi 20 años de experiencia en el mundo de la programación

+11 años como formador en SEAS.

+15 años trabajando con Microsoft .Net.

+1 año con Angular 2/7

Durante mi vida profesional he ido avanzando en distintas empresas.

Actualmente en GFT (Ibercaja).

Angular es una tecnología estratégica en GFT.

rubenmagallon@estudiosabiertos.com



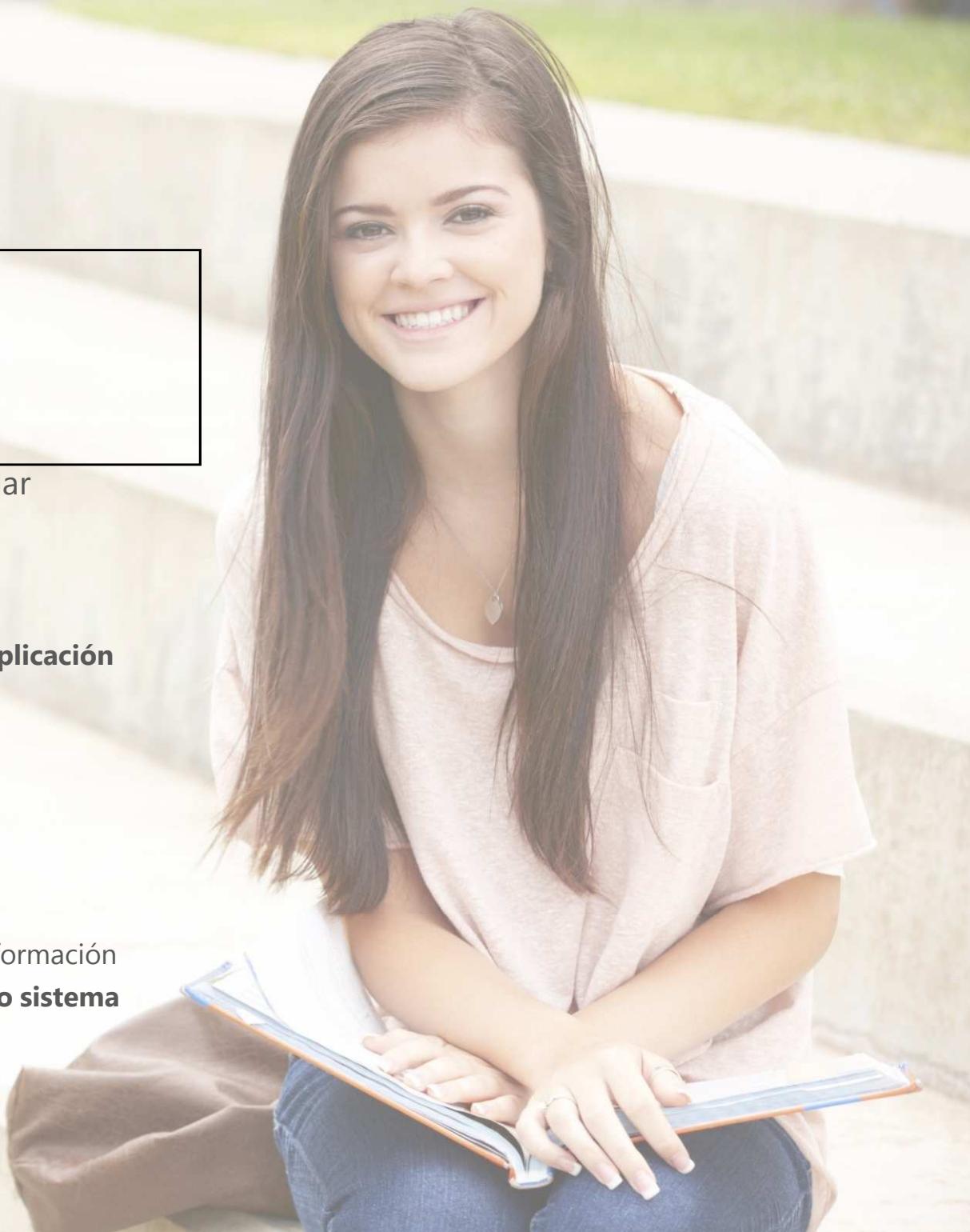
Índice

Primera sesión

1. Primeros pasos
 - a) Orígenes, historia y versiones.
 - b) Herramientas de trabajo
 - c) Instalando Angular
 - d) Mi primera aplicación**
2. Conociendo los recursos de Angular
 - a) ¿Con qué elementos contamos?
 - b) Trabajando con componentes
 - c) Implementando tuberías.
 - d) Agregando reviews a nuestra aplicación**

Segunda sesión

1. Conectando nuestras páginas
 - a) Motor de rutas
 - b) Desarrollando la arquitectura**
2. Accediendo a datos
 - a) Servicios para la obtención de información
 - b) Gestionando datos para nuestro sistema**





a | Orígenes, historia y versiones

- Angular es un “framework/plataforma” de desarrollo para JavaScript creado por Google. La finalidad de Angular es facilitarnos el desarrollo de aplicaciones web SPA y además darnos herramientas para trabajar con los elementos de una web de una manera más sencilla y óptima.
- Otro propósito que tiene Angular es la separación completa entre el front-end y el back-end en una aplicación web.
- Desde la versión 2, ha sido completamente reescrito, ahora la versión 1 se llama Angular JS, y a partir de la versión 2, se llama Angular.
- Cada versión que sacan mejoran la velocidad de compilación y el tamaño final de la aplicación. El objetivo es que un “Hola mundo”, se pueda ejecutar con 10K (actualmente estamos en 22K, y se comenzó con 170K)
- Se puede ampliar con otros frameworks, ensamblados o patrones actuales como:
 - Ionic (framework pensado para móviles)
 - RxJS (librería asíncrona y basados en eventos y observables).
 - PWA (Progresive Web Apps) con Angular es sencillo (server push notifications, service workers, etc.)
- “Web Components” a través de Angular Elements es otro objetivo de Angular, ser capaces de ejecutar un componente fuera del mundo Angular. (v6)
- Angular combina HTML, CSS y Javascript (TypeScript)



a Orígenes, historia y versiones

- **AngularJS** fue creado por [Misko Hevery](#) en 2009 en Google.
- No surgió como proyecto de Google, sino que fue desarrollado en el tiempo extra que proporciona Google a sus empleados para desarrollos personales.
- Se apostó unas cervezas a que podía reescribir la aplicación de feedback de Google en 2 semanas (el desarrollo original tiene 17.000 líneas de código y costó 6 meses)
- ¡Perdió!. Tuvo que pagar las cervezas. Le costó 3 semanas y tan solo 1500 líneas de código.
- Esto llamó la atención y el siguiente proyecto se realizó en paralelo con Angular y con la arquitectura anterior a ver quien acababa antes con los estándares de calidad de la compañía.
- Aproximadamente a los 3 meses, abandonaron el desarrollo con la antigua arquitectura, ya que llevaban desarrollado un porcentaje (%) muy bajo, frente a lo que ya estaba desarrollado con angular.
- A partir de aquí se doblaron esfuerzos en el equipo Angular, lo van mejorando cada día más, sacando nuevas versiones cada poco tiempo, reescribieron la versión original (JS) para hacerla más rápida.

**a**

Orígenes, historia y versiones

AngularJS

2010 – Angular 1

2013 – Angular 1.2

2014 – Angular 1.3

2015 Mayo – Angular 1.4

2016 Febrero – Angular 1.5

Componentes

2016 Diciembre – Angular 1.6

2018 – Angular 1.7

Angular

Se anuncia la versión 2

2016 Septiembre – Angular 2.0

2017 Marzo – Angular 4.0

2017 Noviembre – Angular 5.0

Mayo – Angular 6.0

2018 Octubre – Angular 7.0

Nuevas versiones cada 6 meses

Actualizaciones casi automáticas
[ng update](#)

Diapositiva 5

RMN7 <https://github.com/angular/angular/blob/master/CHANGELOG.md>

La versión 2 frente a la uno es radical, fue reescrita por completo.

<http://blog.enriqueoriol.com/2017/03/angular4.html#novedades>

Cambios de la 2 a la 4:

- if/else en templates
- Modulo de animación separado
- Uso de StrictNullChecks de TypeScript
- Integración de Angular Universal como módulo de Angular
- Mejoras de rendimiento gracias a FESM

<https://www.campusmvp.es/recursos/post/angular-5-todo-lo-que-necesitas-saber-en-10-minutos-o-menos.aspx>

Cambios de la 4 a la 5

- HttpClient
- Nuevos eventos del ciclo de vida del enrutador
- Mejoras en el compilador
- Build Optimizer
- CLI v1.5
- API Angular Universal State Transfer y soporte DOM
- Internacionalización de tuberías para números, fechas y monedas
- Se reemplaza ReflectiveInjector con StaticInjector
- Mejoras de velocidad de zonas
- exportAs
- Formularios
- RxJS 5.5

<http://blog.enriqueoriol.com/2018/05/novedades-angular-6.html>

Cambios de la 5 a la 6

- Comando de actualización ng-update
- Comando de añadir dependencias ng-add
- Angular Elements
- Angular Material y Material starters
- Angular Component Dev Kit (CDK)
- Nuevo archivo de configuración y CLI workspaces

Diapositiva 5 (continuación)

- Soporte de la CLI a librerías
- Providers tree-shakable
- Mejor rendimiento en animaciones
- RxJS v6
- Long Term Support (LTS)

<https://www.campusmvp.es/recursos/post/angular-7-ya-esta-aqui-y-estas-son-sus-novedades.aspx>

Cambios de la 6 a la 7

- Pequeñas mejoras de rendimiento
- Avisos en la CLI
- Diseño "Material" y el Component Development Kit
- Scroll virtual
- Proyección de contenidos en elementos
- Nuevos componentes y utilidades de la comunidad
- Nuevas dependencias

Rubén Magallón Nuño; 04/01/2019

b Herramientas de trabajo

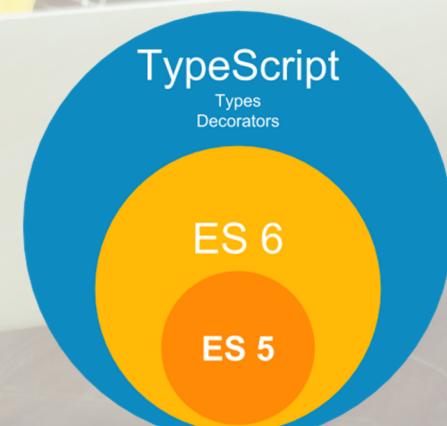
Entorno de desarrollo Local

- Entorno de ejecución [NodeJS](#)
- Editor [Visual Studio Code](#)

Entorno de desarrollo en la nube

- <https://stackblitz.com/>

TypeScript (Microsoft)



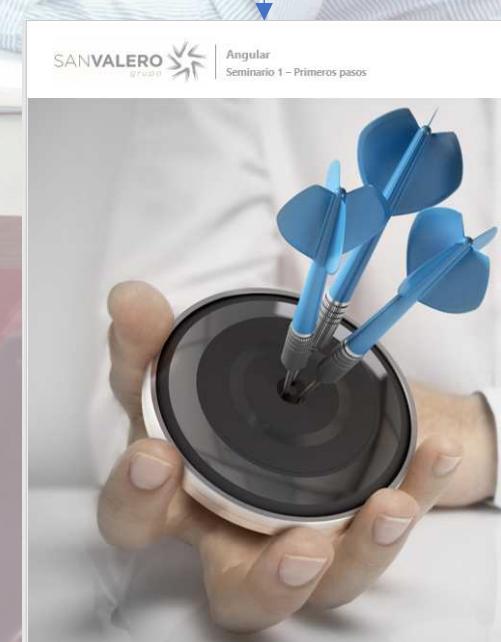
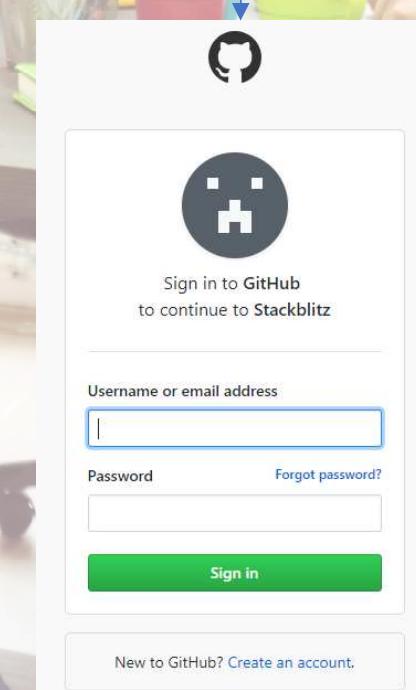
2 proyectos

Pruebas

Preferiblemente en stackblitz
Plugin botón derecho para crear

YouFilm

Preferiblemente en local
Comandos para crear



Diapositiva 6

RMN3

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipado estático y objetos basados en clases.

En principio no llevaba decoradores y Angular los necesitaba (estaban pensando en hacer otro subconjunto por encima de typeScript) pero microsoft les dijo que lo ponían ellos en TypeScript.

De esta forma llegó la colaboración entre google y microsoft en Angular

Rubén Magallón Nuño; 29/12/2018



C Instalando Angular

[npm install -g @angular/cli@latest](#) -> Instala a nivel global la última versión

[npm install -g @angular/cli@7.0.2](#) -> Instala a nivel global la versión indicada

Es posible tener instalado por cada aplicación Angular el angular cli, quitando el parámetro -g, pero son muchos archivos que tendríamos repetidos por cada aplicación, lo mejor es tener una versión instalada globalmente.

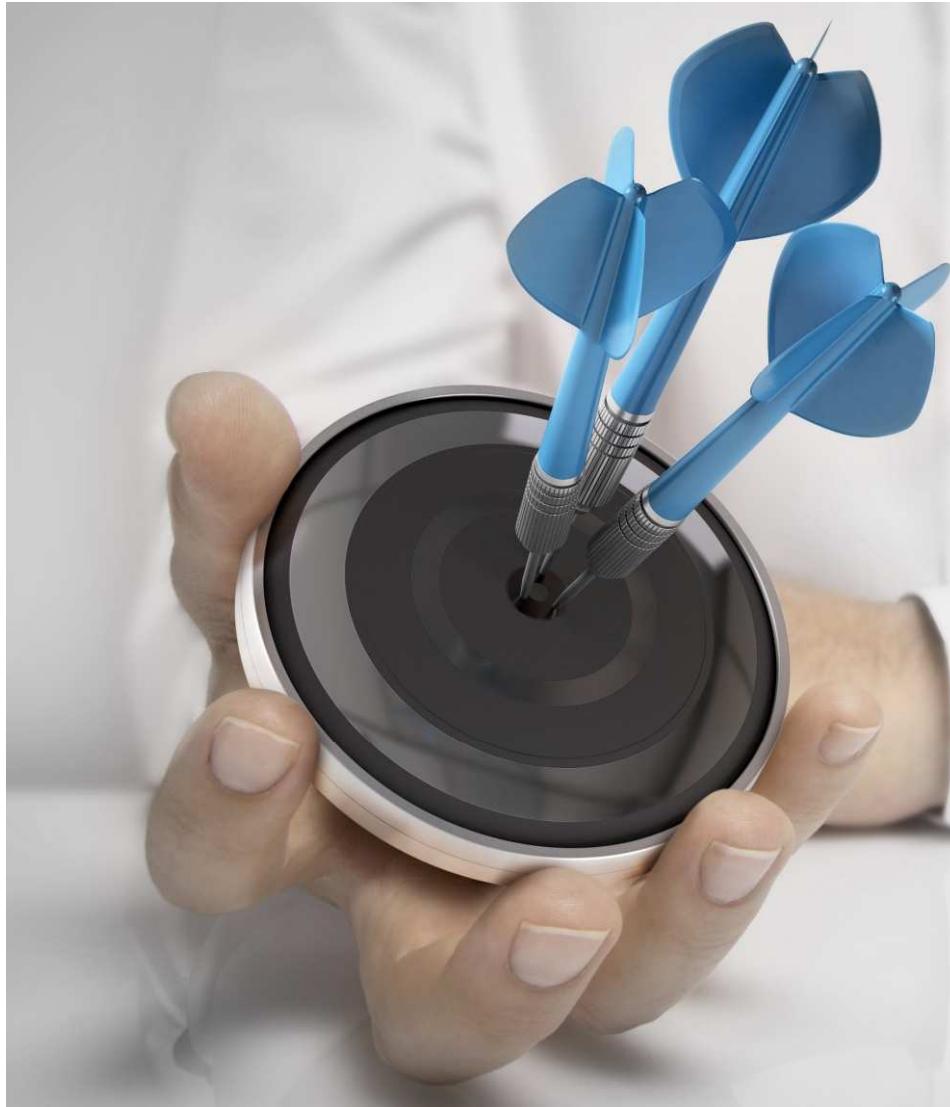
[npm uninstall -g @angular/cli](#) -> Desinstala la versión de angular cli instalada

[npm cache clean](#) -> Limpia la caché

[npm cache verify](#) -> Verifica la caché

[node --version](#)

[ng --version](#)

**d**

Mi primera aplicación

ng new YouFilms

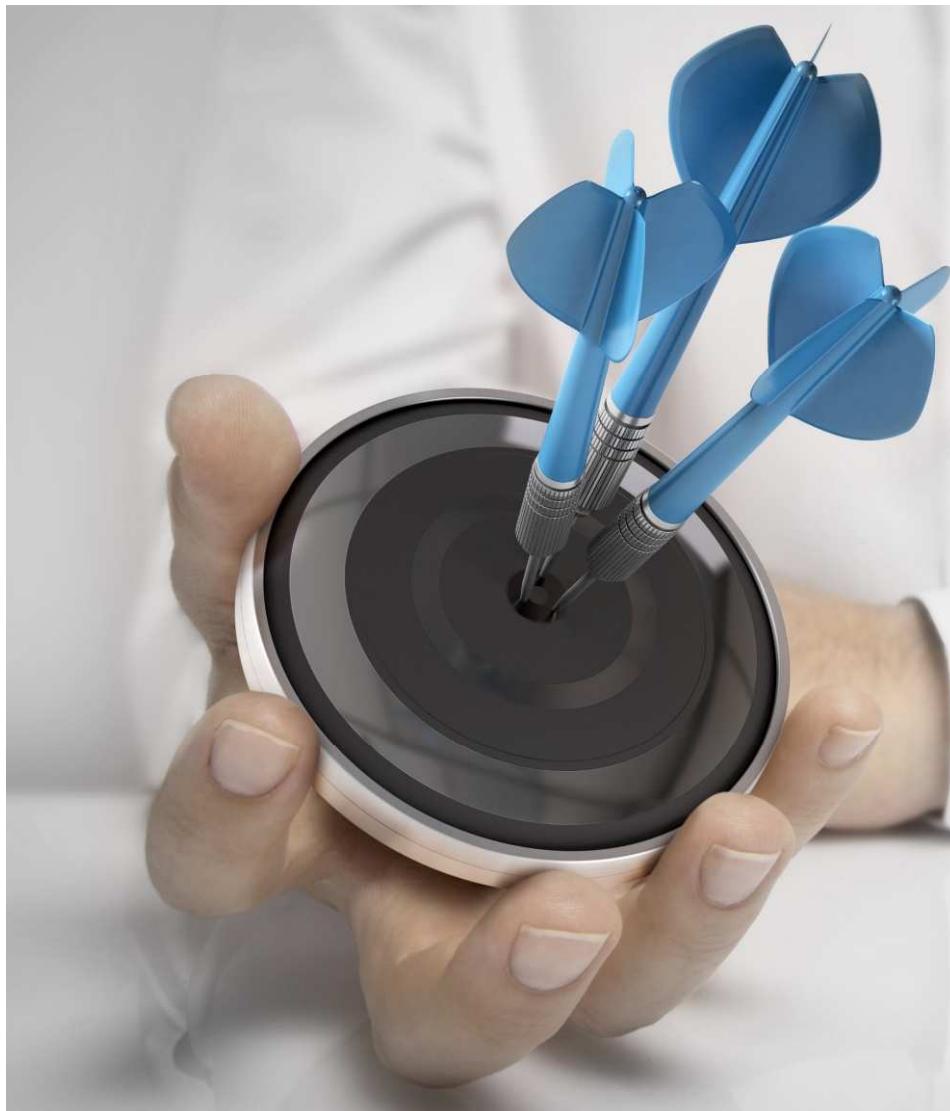
Pregunta si queremos rutas, para este primer ejemplo indicamos N

Pregunta por el formato de las hojas de estilo, de momento CSS

Entramos al directorio del proyecto

ng serve

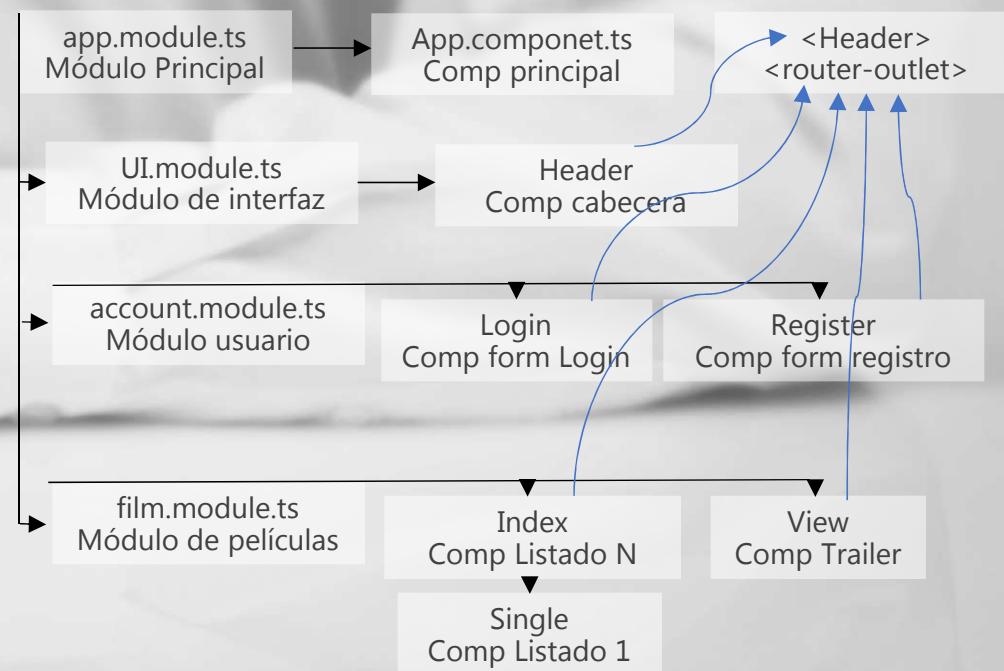
Navegador abrir <http://localhost:4200>



d

Mi primera aplicación

La aplicación va a mostrar una serie de películas (nombre, imagen, descripción y puntuación) donde podemos ver su tráiler solo si estamos autenticados, para ello tendremos la posibilidad de registrarnos y de acceder con nuestro usuario y contraseña.





Cambiamos de tercio.

¿Qué tal vamos de hora?

Muy pronto para el descanso ;)



centro
SAN VALERO
GRUPO SANVALERO



fundación dominicana
FUNDOSVA
GRUPO SANVALERO



formación
CPA SALDUIE
GRUPO SANVALERO



estudios abiertos
SEAS
GRUPO SANVALERO



universidad
SANJORGE
GRUPO SANVALERO

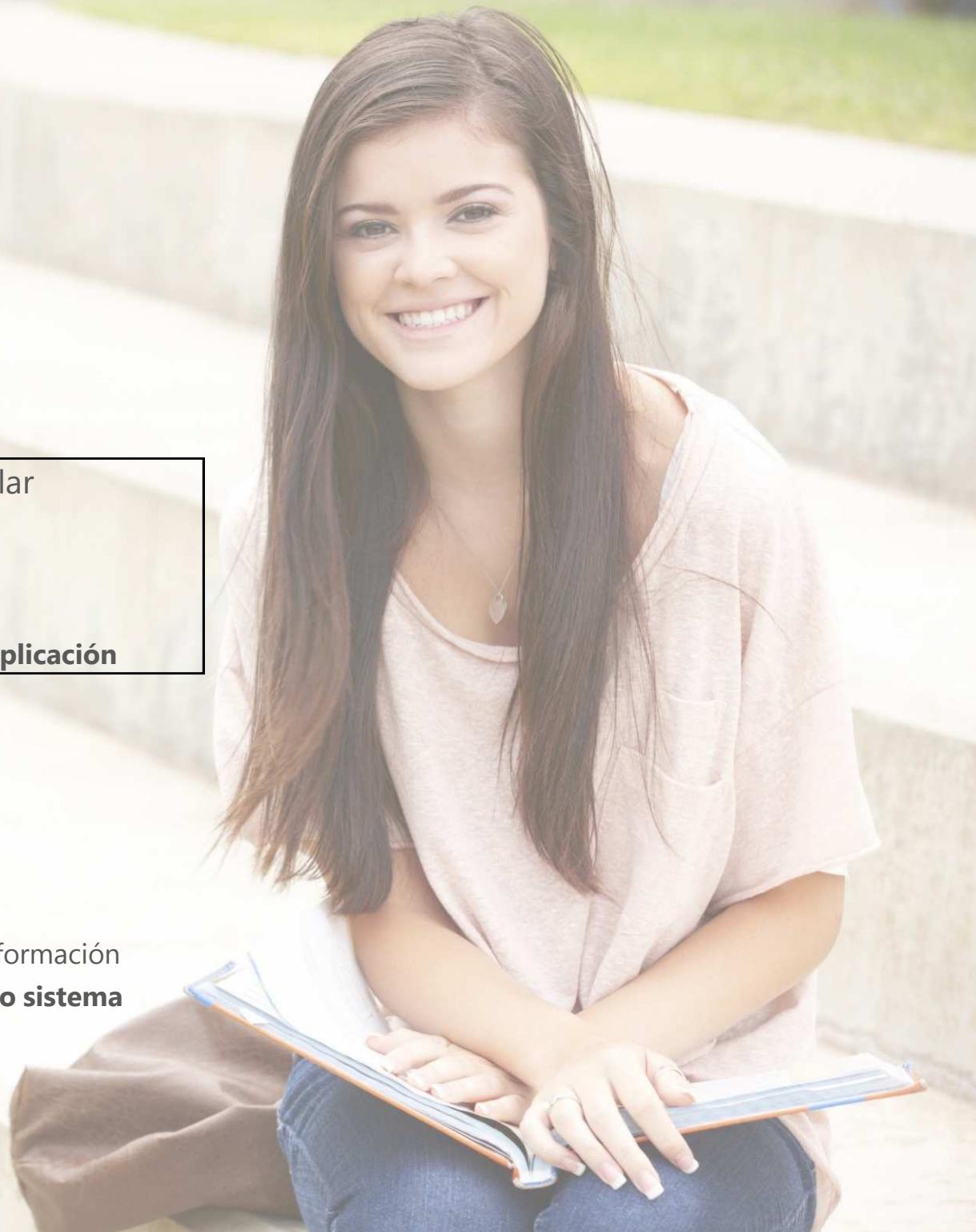
Índice

Primera sesión

1. Primeros pasos
 - a) Orígenes, historia y versiones.
 - b) Herramientas de trabajo
 - c) Instalando Angular
 - d) Mi primera aplicación**
2. Conociendo los recursos de Angular
 - a) ¿Con qué elementos contamos?
 - b) Trabajando con componentes
 - c) Implementando tuberías.
 - d) Agregando reviews a nuestra aplicación**

Segunda sesión

1. Conectando nuestras páginas
 - a) Motor de rutas
 - b) Desarrollando la arquitectura**
2. Accediendo a datos
 - a) Servicios para la obtención de información
 - b) Gestionando datos para nuestro sistema**



a

Con qué elementos contamos

[module](#)[component](#)[pipe](#)[service](#)[guard](#)[directive](#)

class

enum

...

[ng generate](#)

Diapositiva 12

M1

Modulo: Agrupación de elementos que nos permite dividir y organizar la aplicación.

Componente: controla una zona de espacio de la pantalla que denominada vista a través de un tag HTML

tuberia: Necesitamos formatear algo en una vista, un número, una fecha, o cualquier otra cosa, para eso usaremos las tuberías o pipes

servicios: Los servicios son una pieza fundamental, pues en ellos vamos a tener encapsulada toda la lógica de negocio, y las llamadas HTTP

class: Clase típica de programación donde podemos poner propiedades y métodos.

guard: Asegurar el acceso a una ruta en concreto en función de unas condiciones

directiva: decorador para añadir metadatos

enum: típico enumerador donde tenemos una lista de valores

Magallón; 28/12/2018

b

Trabajando con componentes

Antes de empezar con los componentes, veamos módulos.

Los módulos es un elemento principal, en el cual agrupamos componentes, directivas, tuberías, etc.

Al crear una aplicación, el módulo principal ya viene creado app.module.ts, y en base a él cuelgan todo lo demás.

El decorador que caracteriza a un módulo de Angular es [@NgModule](#)

Podemos crear tantos como queramos, pero con sentido común.

[ng generate module name](#)

@NgModule({

imports: [],	Importa otros módulos necesarios.
declarations: [],	Declara elementos que pertenecen al módulo.
exports: [],	Hace públicos elementos para que puedan ser usados por otros módulos.
providers:[]	Proporciona servicios para que los componentes del módulo puedan usarlos.

})

b

Trabajando con componentes

En nuestra aplicación YouFilms vamos a agregar tres módulos (debería crearnos las carpetas de los módulos, si no es así las creamos y movemos los ficheros para que quede de la siguiente forma):

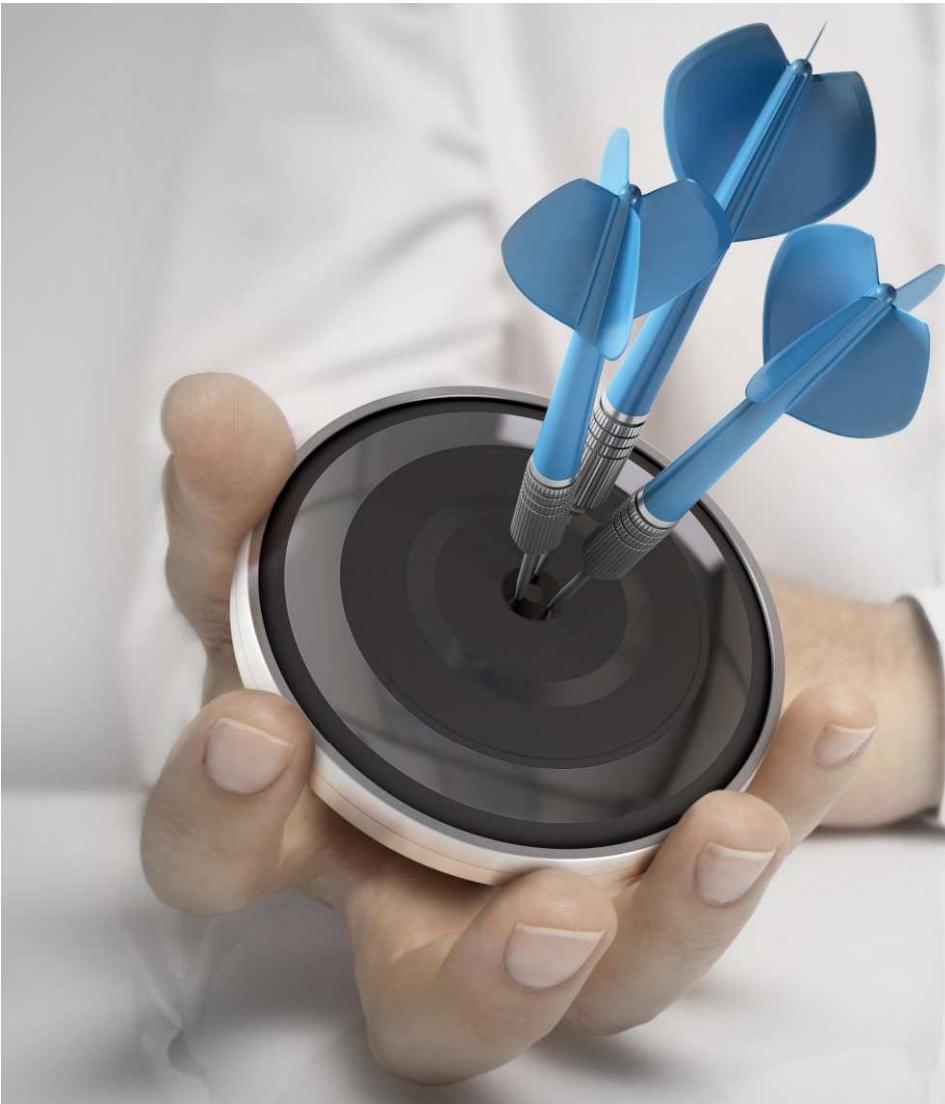
```
App
  Account
    account.module.ts
  Film
    film.module.ts
  UI
    ui.module.ts
  app.module.ts
```

Nos situamos en la carpeta src/app

UI -> [ng generate module](#) UI

Account -> [ng generate module](#) Account

Film -> [ng generate module](#) Film



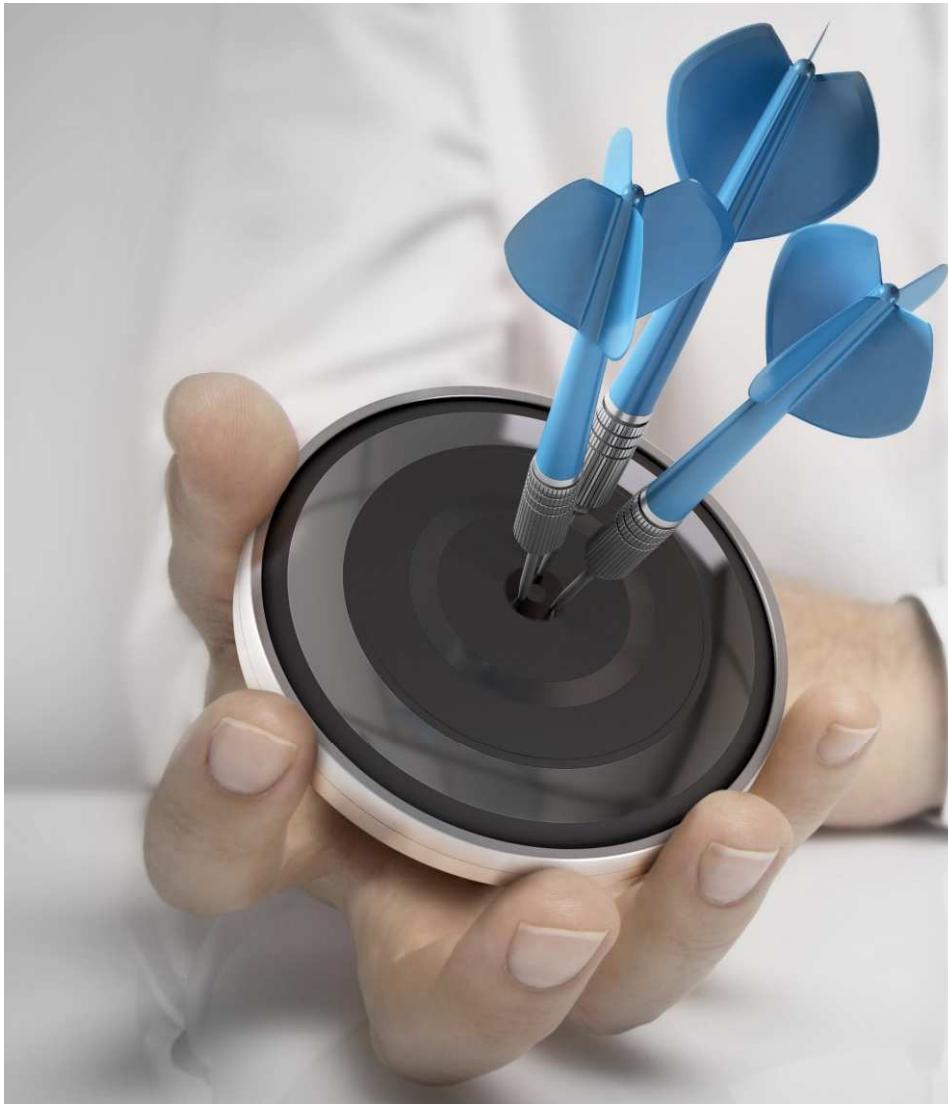
b

Trabajando con componentes

Importar los módulos recién creados en el módulo principal (app.module.ts) quedando de esta forma:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { UIModule } from './ui/ui.module';
import { AccountModule } from './account/account.module';
import { FilmModule } from './film/film.module';
import { AppComponent } from './app.component';
import { HelloComponent } from './hello.component';

@NgModule({
  imports: [ BrowserModule, FormsModule, UIModule,
AccountModule, FilmModule ],
  declarations: [ AppComponent, HelloComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```



b

Trabajando con componentes

```
ng generate component hello
```

```
hello.component.ts
```

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

```
@Component({
```

```
  selector: 'hello',
```

```
  templateUrl: './hello.component.html',
```

```
  styleUrls: [ './hello.component.css' ]
```

```
})
```

```
export class AppComponent {
```

```
}
```

```
hello.component.html
```

```
<H1>Hello! </H1>
```

```
hello.component.css
```

```
h1 { font-family: Lato; }
```

```
@Component({
```

```
  selector: 'hello',
```

```
  template: `<h1>Hello!</h1>`,
```

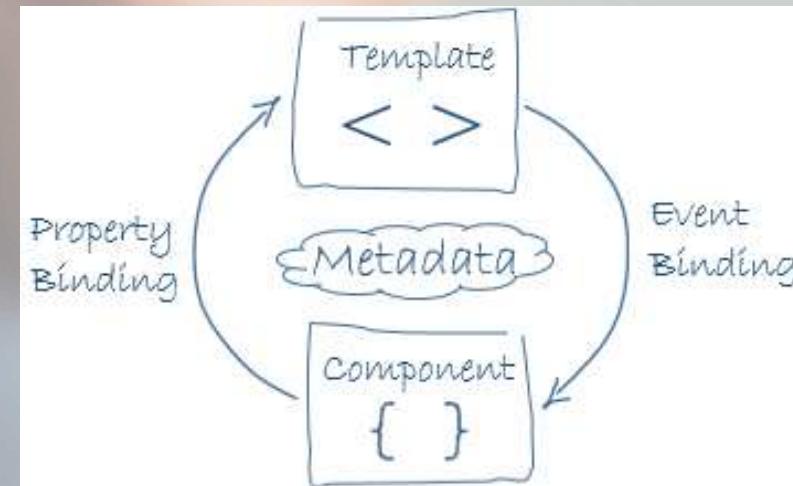
```
  styles: [ `h1 { font-family: Lato; }` ]
```

```
})
```

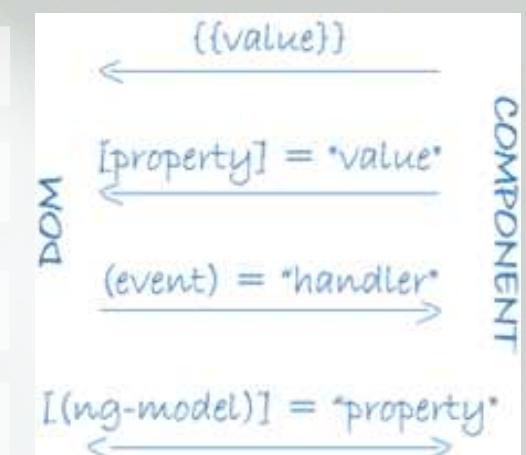
b

Trabajando con componentes

Comunicación entre componente y template



- [Interpolation](#)
- [Property Binding](#)
- [Event Binding](#)
- [Two-Way](#)



b

Trabajando con componentes

componente.ts

```
export class ComponentsComponent {  
    name:string = "Angular";  
}
```

- **Interpolation**

componente.html

```
<b>Interpolación:</b>{{name}}
```

- **Property Binding**

componente.html

```
<p [innerHTML]="name">
```

- **Event Binding**

componente.html

```
<button (click)="Set('Angular')">reset</button>
```

componente.ts

```
Set(value:string){ this.name = value }
```

- **Two-Way** (hace falta FormsModule de '@angular/forms')

componente.html

```
<input [(ngModel)]=name>
```

b

Trabajando con componentes

Vamos a practicar con Two-Way y Event Binding

En nuestra aplicación YouFilms vamos a agregar varios componentes, quedando de esta forma:

Account

login

login.component.css

login.component.html

login.component.ts

register

register.component.css

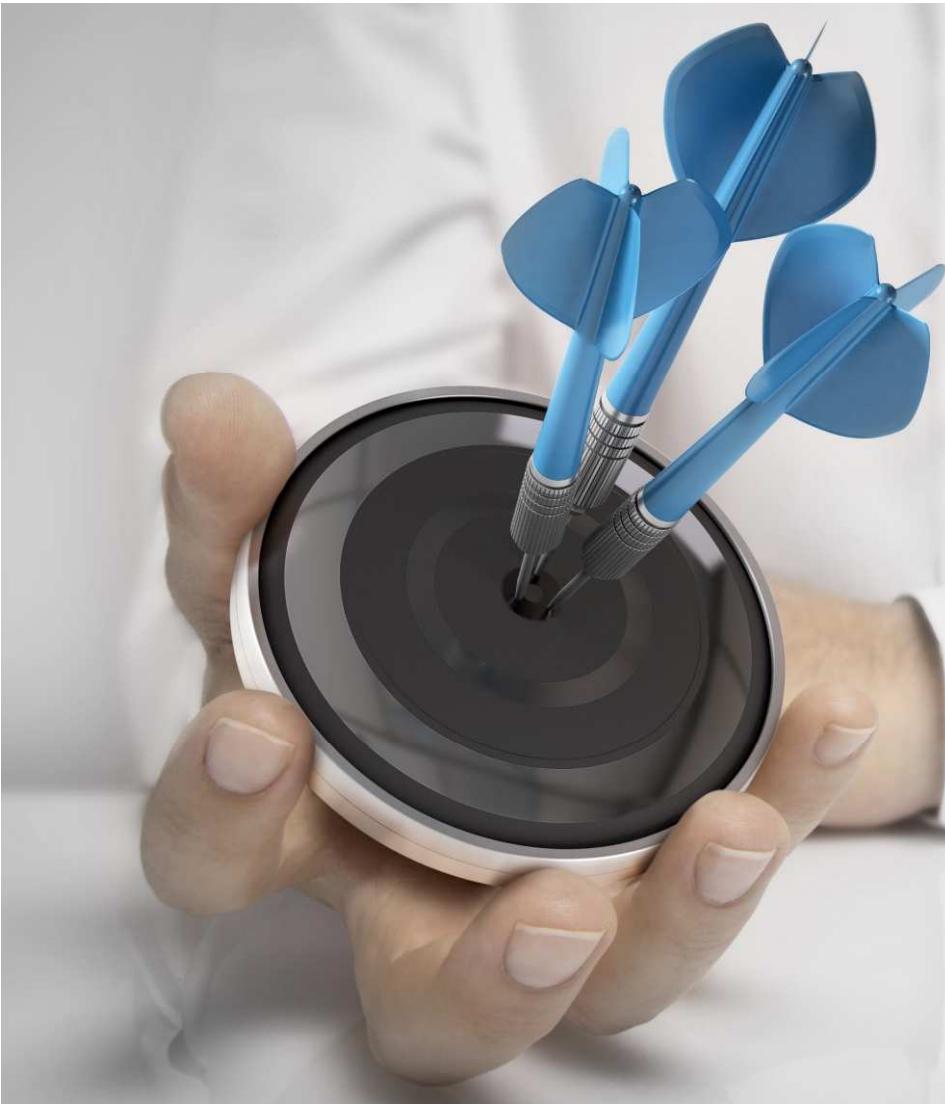
register.component.html

register.component.ts

Nos situamos dentro de la carpeta del módulo Account

[ng generate component Login](#)

[ng generate component Register](#)



b

Trabajando con componentes

login.component.ts

```
export class LoginComponent implements OnInit {  
    username:string;  
    password:string;  
    Login(){  
        console.log("Login de Usuario:" + this.username +  
        " Password:" + this.password);  
    }  
    constructor() {}  
    ngOnInit() {}  
}
```

login.component.html

```
<H1>Login</H1>  
<input [(ngModel)]=username type="text"><br/>  
<input [(ngModel)]=password type="password"><br/>  
<button (click)="Login()">Login</button>
```

Lo mismo con el componente register, cambiar el nombre del método de Login a Register, y los textos que aparece Login por Registro, además indicar debajo del botón algo así como: **Recuerde que para acceder deberá introducir como nombre lo de usuario: {{username}}**



b

Trabajando con componentes

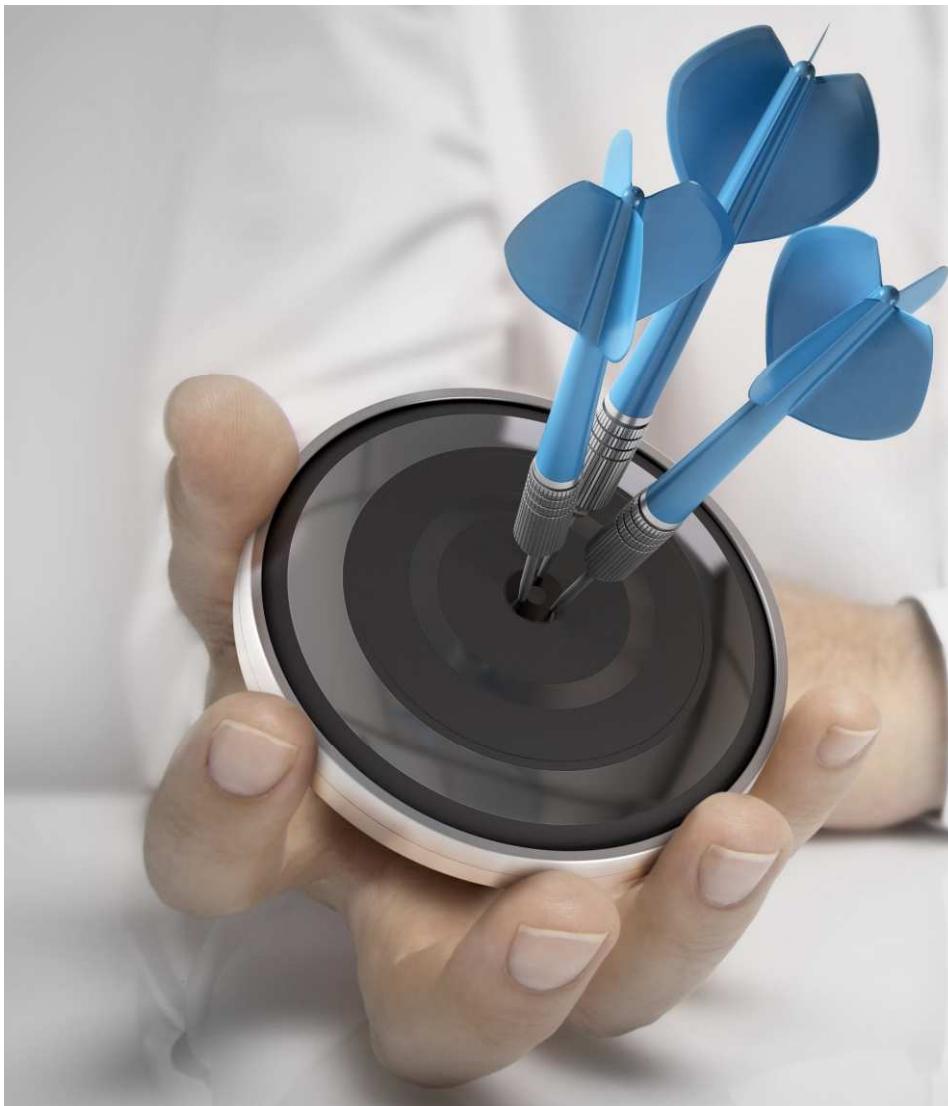
Modificar el módulo Account, para importar FormsModule que nos hará falta por el uso de ngModel en el template, y vamos a exportar para hacer públicos sus componentes.

account.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';

import { LoginComponent } from './login/login.component';
import { RegisterComponent } from './register/register.component';

@NgModule({
  imports: [ CommonModule, FormsModule ],
  declarations: [LoginComponent, RegisterComponent],
  exports : [ LoginComponent, RegisterComponent ]
})
export class AccountModule { }
```



b

Trabajando con componentes

Ahora que ya sabemos que un template es código HTML

Veamos algunas pequeñas utilidades para el template:

*ngIf -> Usado para mostrar u ocultar algo en base a una condición

```
<div *ngIf="(accountService.token==null)">  
    Para poder ver un video necesitas estar autenticado  
</div>
```

*ngFor -> Usado para iterar dentro de una colección

```
<p *ngFor="let name of names">  
    Hola {{name}}  
</p>
```

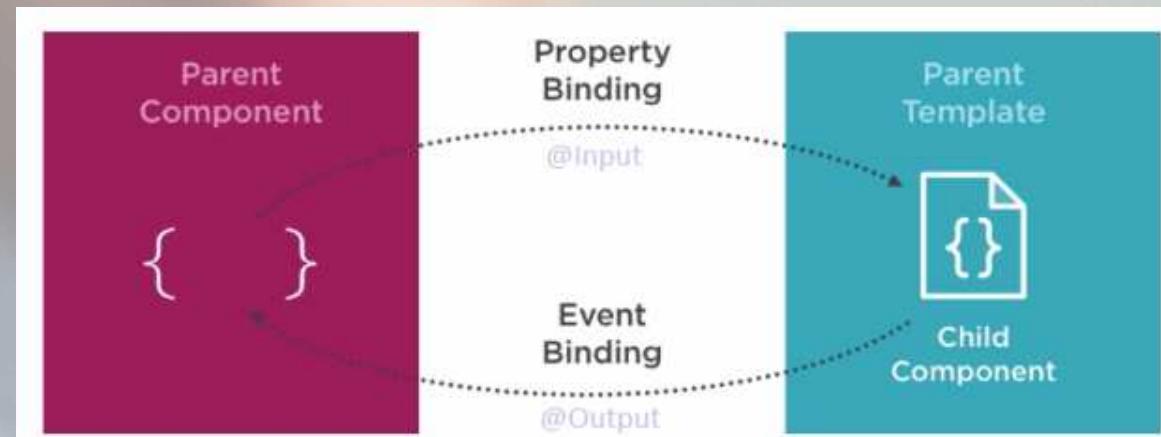
ngSwitch -> Usado para mostrar una cosa u otra en función del valor de una propiedad

```
<container-element [ngSwitch]="switch_expression">  
    <some-element *ngSwitchCase="match_expression_1">...</some-element>  
    <some-element *ngSwitchCase="match_expression_2">...</some-element>  
    <some-element *ngSwitchDefault>...</some-element>  
</container-element>
```

b

Trabajando con componentes

Comunicación entre distintos componentes



[@Input](#)

[@Output](#)

b

Trabajando con componentes

@Input

childComponent.ts

```
import { Component, OnInit, Input } from '@angular/core';
@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  @Input() name:string;
}
```

childComponent.html

```
Hola {{name}}
```

parentComponent.ts

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-parent',
  template: `<p *ngFor="let name of names">
    <app-child name={{name}}></app-child></p>`,
  styles: [`p{font-size:18px}`]
})
export class ParentComponent {
  names: Array<string> = ["Albert", "Jorge", "Luis"];
}
```

b

Trabajando con componentes

@Output

childComponent.ts

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';
@Component({
  selector: 'app-childCP',
  template: `<br/><button (click)=send()>Enviar</button>`
})
export class ChildComponentCP {
  @Output() event = new EventEmitter<string>();
  send = function(){
    this.event.emit("enviado");
  }
}
```

parentComponent.ts

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-parentCP',
  template: `<app-childCP (event)=aviso($event)></app-childCP>`
})
export class ParentComponentCP {
  aviso = function(message:string){ alert(message); }
}
```

b

Trabajando con componentes

Vamos a practicar con Input y Output

En nuestra aplicación YouFilms vamos a agregar varios componentes, quedando de esta forma:

Film

Index
index.component.css
index.component.html
index.component.ts

Single

single.component.css
single.component.html
single.component.ts

View

view.component.css
view.component.html
view.component.ts

Film.ts (clase)

id, points:number;
name, description, video:string;
images:Array<string>;

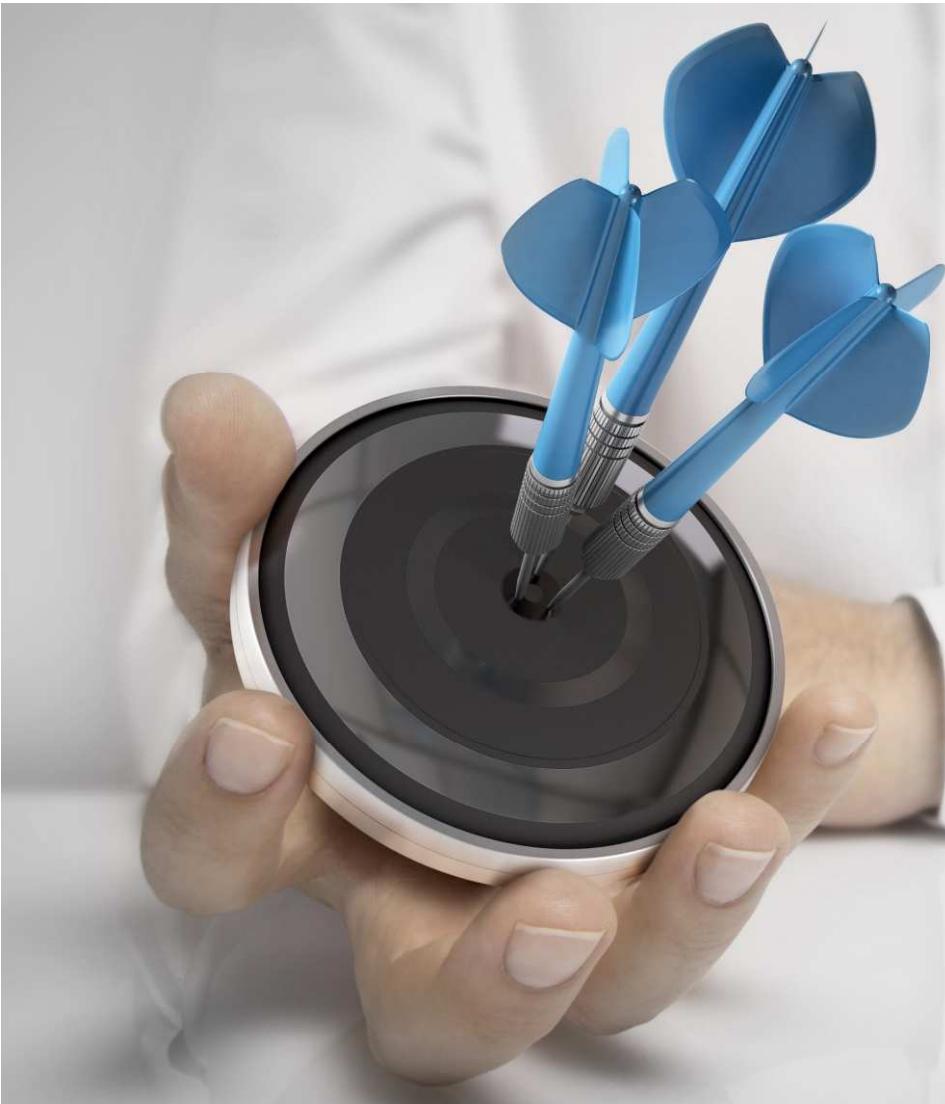
Nos situamos dentro de la carpeta del modulo Film

[ng generate component Index](#)

[ng generate component Single](#)

[ng generate component View](#)

[ng generate class Film](#)



b

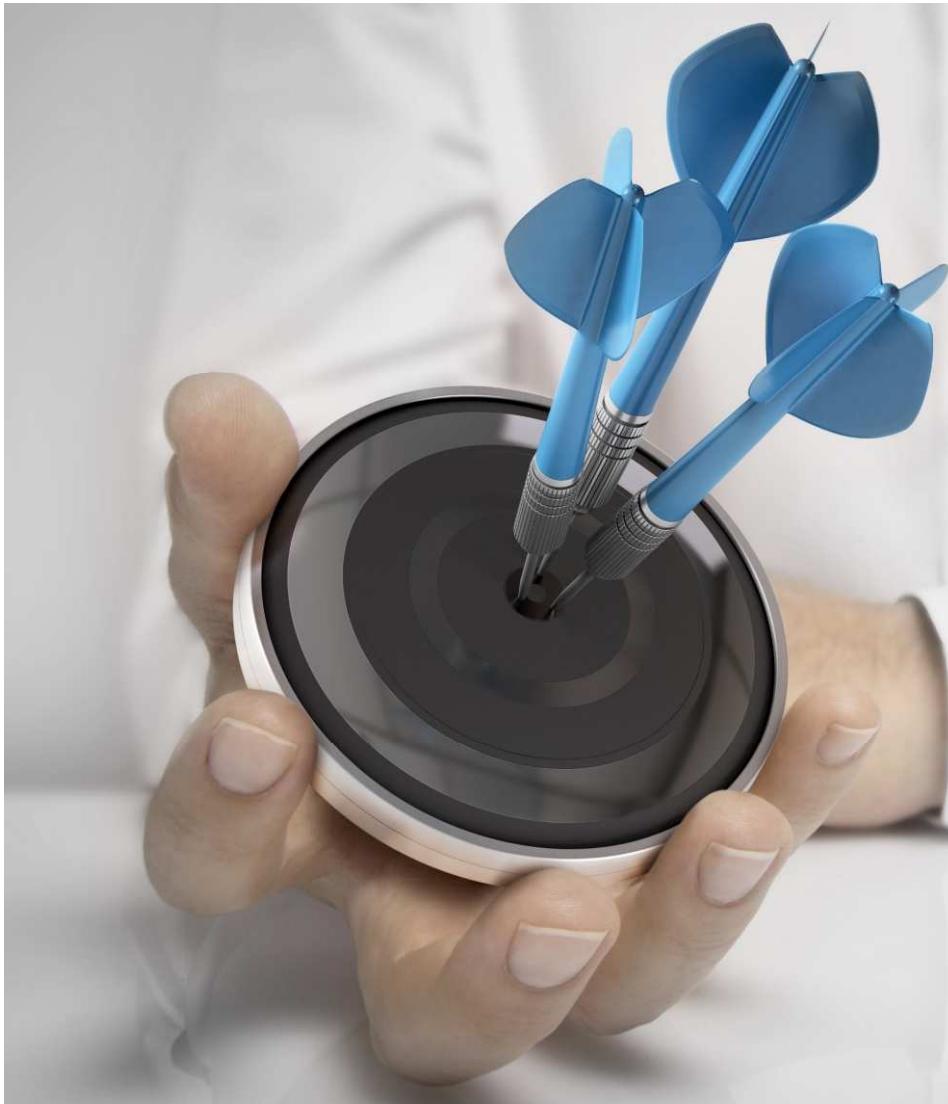
Trabajando con componentes

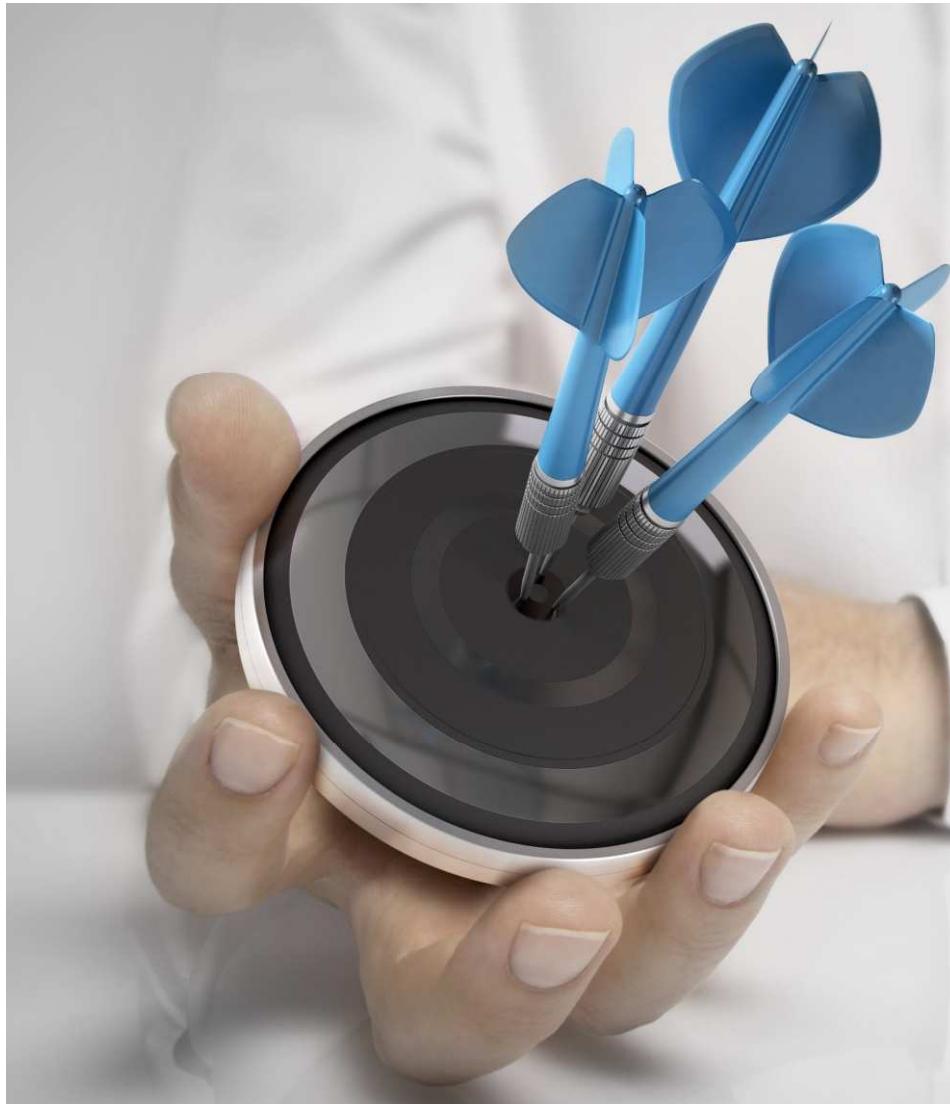
single.component.html

```
<table style="width:360px;display: inline-block;float;padding:5px;vertical-align: text-top">
<tr><td>
<h2>{{filmInput.name}} <button
(click)=view(filmInput)>Ver</button></h2>
</td></tr><tr><td>
<img [src] = "filmInput.images[0]" width="100%" />
</td></tr><tr><td>
{{filmInput.description}}
</td></tr></table>
```

single.component.ts

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
import { Film } from './film';
...
export class SingleComponent implements OnInit {
  @Input() filmInput: Film;
  @Output() viewEvent = new EventEmitter<Film>();
  constructor() { }
  ngOnInit() { console.log('mostrando:', this.filmInput.name); }
  view(film: Film) { this.viewEvent.emit(film); }
}
```



**b**

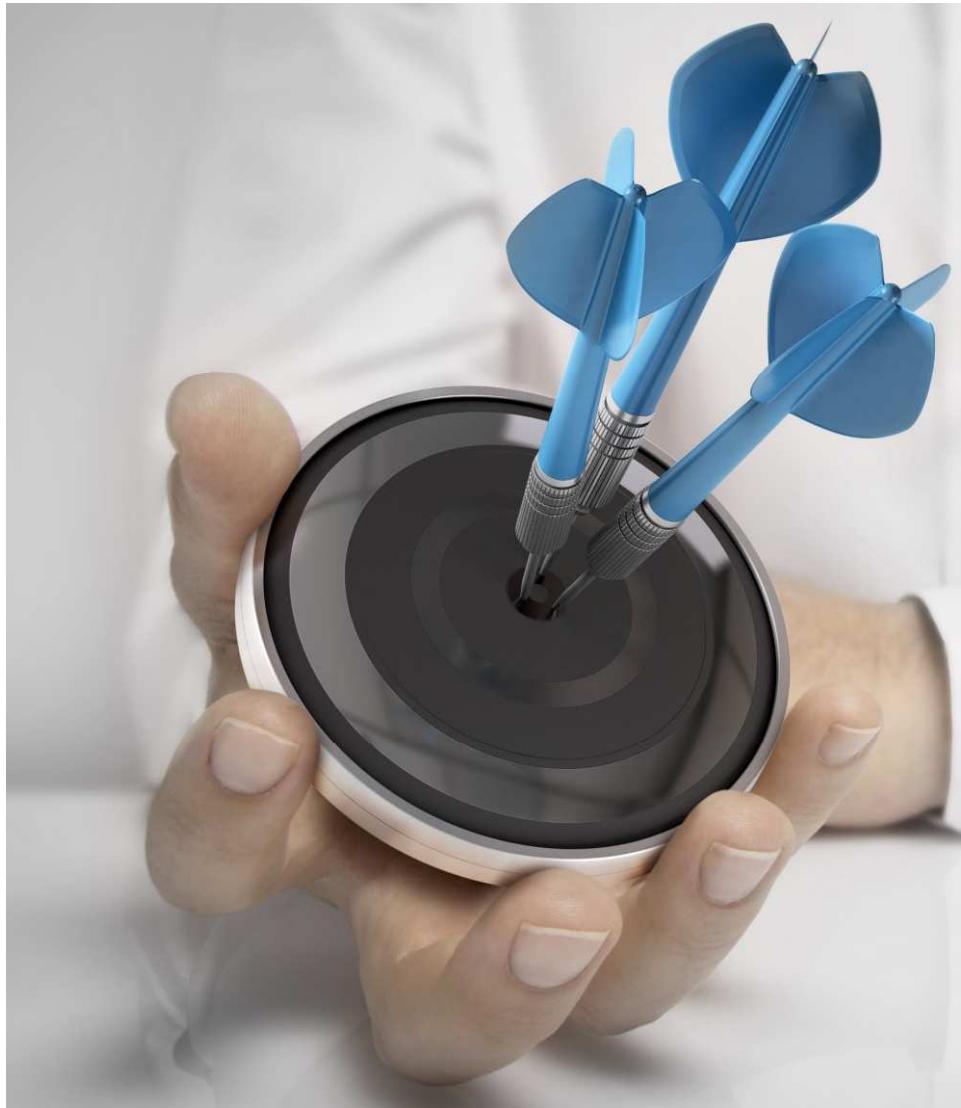
Trabajando con componentes

index.component.html

```
<H1>Películas <button *ngIf="(film!=undefined)" (click)=reset()>
Dejar de ver</button></H1>
<h2 *ngIf="(film!=undefined)">
    ¿Seguro que quieres ver {{film.name}}?
    <button (click)=ver(film.id)>Si, seguro</button>
</h2>
<app-single *ngFor="let film of films" [filmInput]='film'
(viewEvent)=view($event)></app-single>
```

index.component.ts

```
import { Film } from './film';
export class IndexComponent implements OnInit {
    films : Array<Film>;
    film:Film;
    constructor() {
        this.films = [{id:1, name:"", description:"", images:[], video:"", points:3.5}];
    }
    view(film:Film) { this.film = film; }
    reset() { this.film = undefined; }
    ver(id:number) {console.log("Ver película:" + id);}
}
```

**b**

Trabajando con componentes

Modificar el módulo Film, para exportar y hacer públicos los componentes necesarios.

film.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { IndexComponent } from './index/index.component';
import { SingleComponent } from './single/single.component';
import { ViewComponent } from './view/view.component';

@NgModule({
  imports: [ CommonModule ],
  declarations: [IndexComponent, SingleComponent],
  exports: [IndexComponent, ViewComponent]
})
export class FilmModule {}
```

C

Implementando tuberías

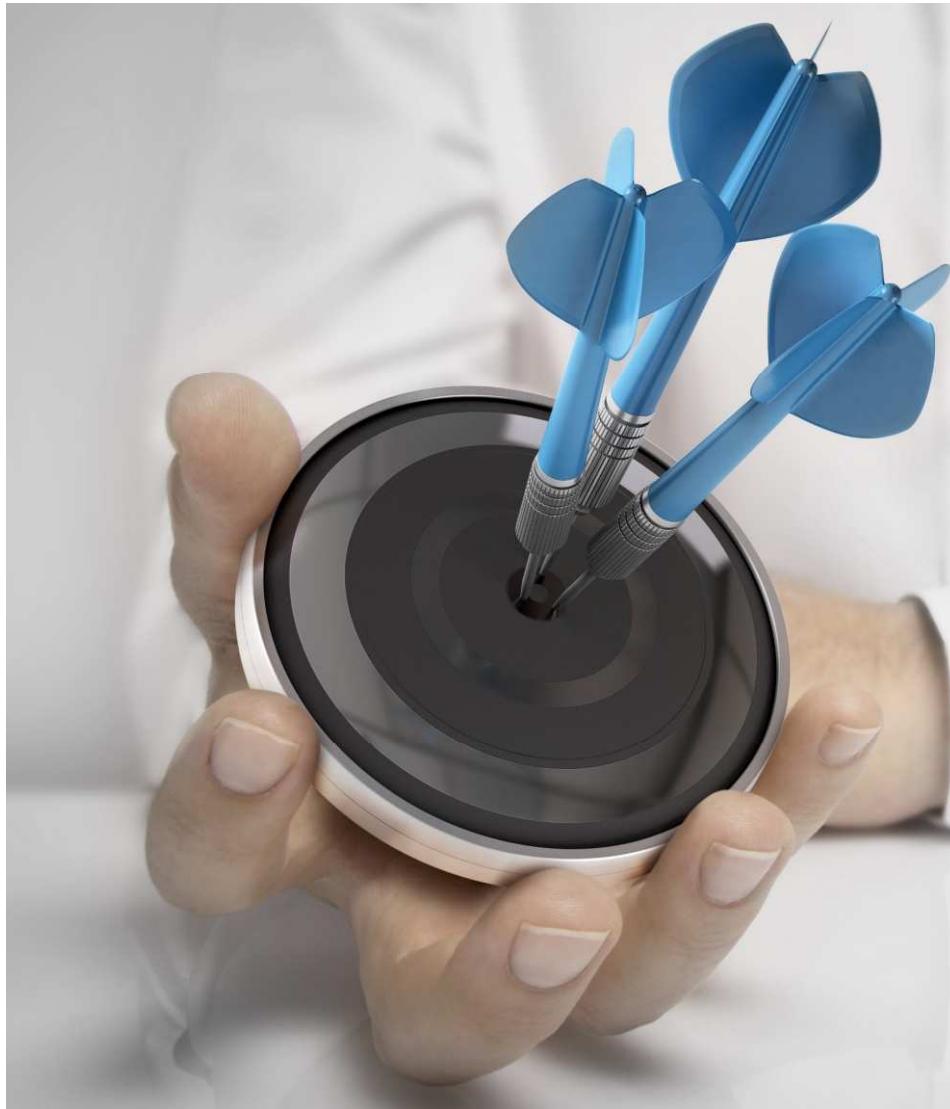
Usar las tuberías existentes: <https://angular.io/api?type=pipe>

Ejemplos: <https://stackblitz.com/edit/ejemplos>

Ahora vamos a hacer nuestra tubería personalizada "Maño"

[ng generate pipe manio](#)

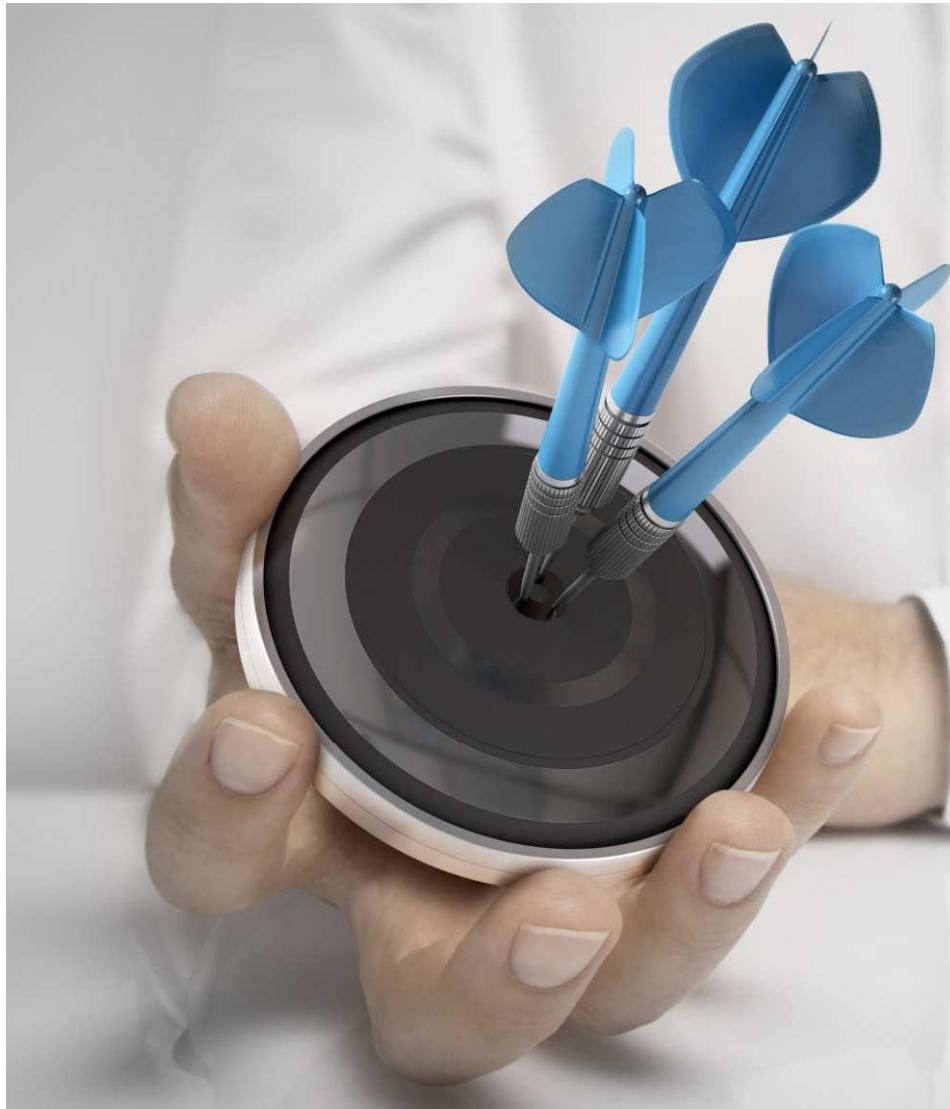
```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
    name: 'manio'
})
export class ManioPipe implements PipeTransform {
    transform(value: string): string {
        let conAcento : Array<string> = ['á','é','í','ó','ú','Á','É','Í','Ó','Ú'];
        let sinAcento : Array<string> = ['a','e','i','o','u','A','E','I','O','U'];
        for (var i = 0; i < sinAcento.length; i++) {
            value = value.replace(new RegExp(sinAcento[i], 'g'), conAcento[i]);
        }
        return value;
    }
}
```

**d**

Agregando reviews

[ng generate pipe Reviews \(en la carpeta Film\)](#)

```
transform(value: number, position: number): string {  
    if (value < (position - 1 + 0.5))  
        return  
        'https://cdn4.iconfinder.com/data/icons/simplicio/128x128/star_empty.png';  
    else if (value >= (position - 1 + 0.5) &&  
            value < position)  
        return  
        'https://cdn4.iconfinder.com/data/icons/simplicio/128x128/star_half.png';  
    else  
        return  
        'https://cdn4.iconfinder.com/data/icons/simplicio/128x128/star_full.png';  
}
```

**d**

Agregando reviews

Modificar single.component.html para mostrar reviews

single.component.html

```
<table style="width:360px;display: inline-block;float;padding:5px;vertical-align: text-top">
<tr><td>
<h2>{{filmInput.name}} <button
(click)=view(filmInput)>Ver</button></h2>
</td></tr><tr><td>
<img [src]={{filmInput.images[0]}} width="100%">
</td></tr>
<tr><td>
<img [src]={{filmInput.points | reviews:1}} style="height:25px" />
<img [src]={{filmInput.points | reviews:2}} style="height:25px"/>
<img [src]={{filmInput.points | reviews:3}} style="height:25px"/>
<img [src]={{filmInput.points | reviews:4}} style="height:25px"/>
<img [src]={{filmInput.points | reviews:5}} style="height:25px"/>
</td></tr>
<tr><td>
{{filmInput.description}}
</td></tr></table>
```



Mañana más



centro
SANVALERO
GRUPO SANVALERO



fundación dominicana
FUNDOSVA
GRUPO SANVALERO



formación
CPA SALDUIE
GRUPO SANVALERO



estudios abiertos
SEAS
GRUPO SANVALERO



universidad
SANJORGE
GRUPO SANVALERO

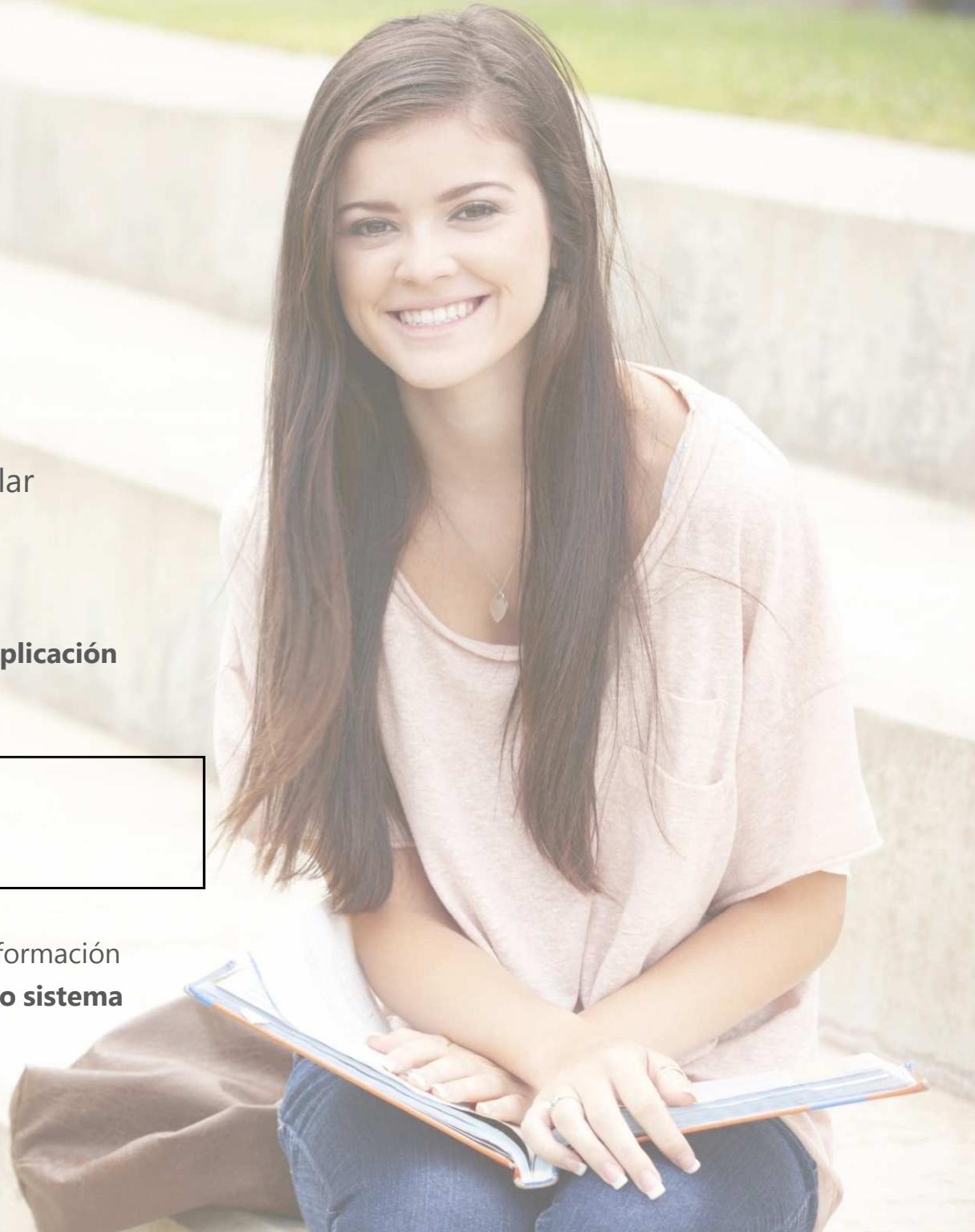
Índice

Primera sesión

1. Primeros pasos
 - a) Orígenes, historia y versiones.
 - b) Herramientas de trabajo
 - c) Instalando Angular
 - d) Mi primera aplicación**
2. Conociendo los recursos de Angular
 - a) ¿Con qué elementos contamos?
 - b) Trabajando con componentes
 - c) Implementando tuberías.
 - d) Agregando reviews a nuestra aplicación**

Segunda sesión

1. Conectando nuestras páginas
 - a) Motor de rutas
 - b) Desarrollando la arquitectura**
2. Accediendo a datos
 - a) Servicios para la obtención de información
 - b) Gestionando datos para nuestro sistema**



a

Motor de rutas

Angular es SPA (Single Page Application), todo se carga en la misma página.

Complicado para este sistema aplicar SEO, si se requiere de SEO necesitamos "[Angular Universal - server side rendering](#)"

Para que Angular sepa el componente que tiene que cargar por url debemos configurar las [rutas de navegación](#).

Se puede configurar en un módulo (al crear la aplicación nos ofrece la opción), pensado para aplicaciones grandes.

También se puede configurar en el módulo principal a través de una variable/constante.

<router-outlet> </router-outlet> Esta la etiqueta HTML donde se va a cargar el contenido.

RouterModule.forRoot(routes) -> Configura Rutas

RouterModule.forChild(routes) -> Configura sub Rutas

a

Motor de rutas

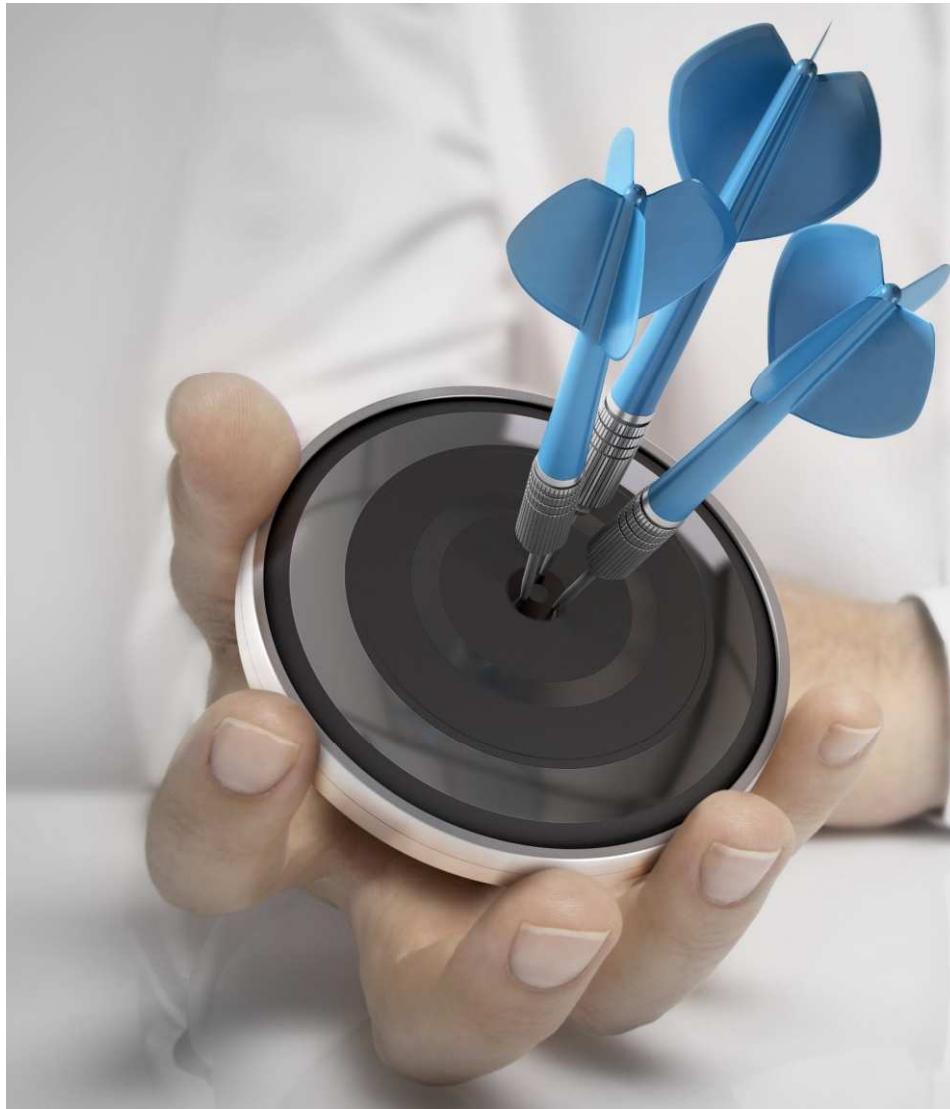
`app.route.ts`

```
import { Routes } from '@angular/router';
import { AComponent } from './a/a.component';
import { BComponent } from './b/b.component';
import { CComponent } from './c/c.component';
```

```
export const routes: Routes = [
  {path : '', redirectTo: '/A', pathMatch:'full'},
  {path : 'A', component : AComponent},
  {path : 'B', component : BComponent},
  {path : 'C', component : CComponent},
  //{path : '**', component : NotFoundComponent},
];
```

`app.module.ts`

```
@NgModule({
  imports:[
    BrowserModule,
    FormsModule,
    RouterModule,
    UIModule,
    AccountModule,
    FilmModule,
    RouterModule.forRoot(routes)
  ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
```

**b**

Arquitectura

app.routes.ts

```
import { Routes } from '@angular/router';
import { IndexComponent } from './film/index/index.component';
import { ViewComponent } from './film/view/view.component';
import { LoginComponent } from './account/login/login.component';
import { RegisterComponent } from './account/register/register.component';

export const routes: Routes = [
  {path : "", redirectTo: '/Index', pathMatch:'full'},
  {path : 'Index', component : IndexComponent},
  {path : 'Login', component : LoginComponent},
  {path : 'Register', component : RegisterComponent},
  {path : 'View/:id', component: ViewComponent}
  // {path : '**', component : NotFoundComponent},
];
```

d

Arquitectura

Nos situamos dentro de la carpeta del módulo UI

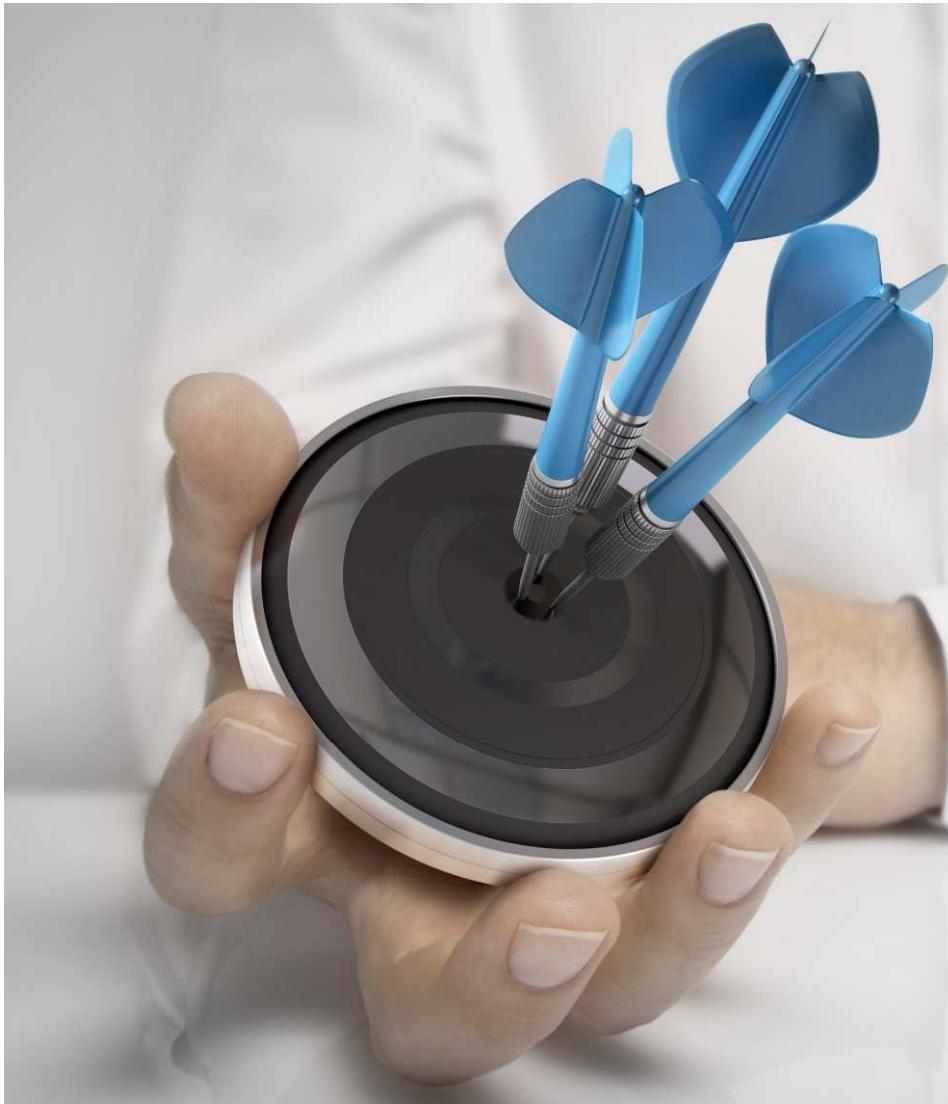
[ng generate component Header](#)

header.component.ts

```
@Component({
  selector: 'app-header',
  template: `
    <div class="header">
      <a routerLink="/" class="logo">YouFilms</a>
      <div class="header-right">
        <a routerLink="/Login" routerLinkActive='active'>Login</a>
        <a routerLink="/Register" routerLinkActive='active'>Register</a>
      </div>
    </div> `,
  styleUrls: ['./header.component.css']
})
```

Importar en el módulo ui.module.ts el RouterModule
import { RouterModule } from '@angular/router';
y añadirlo en imports.

Exportar el componente Header en el módulo ui.module.ts
exports: [HeaderComponent]



**b**

Arquitectura

header.component.css

```
/* https://www.w3schools.com/howto/howto_css_responsive_header.asp */
/* Style the header with a grey background and some padding */
.header {
    overflow: hidden;
    background-color: #f1f1f1;
    padding: 20px 10px;
}
/* Style the header links */
.header a {
    float: left;
    color: black;
    text-align: center;
    padding: 12px;
    text-decoration: none;
    font-size: 18px;
    ...
}
```

b

Arquitectura

app.module.ts

```
import { RouterModule, Routes } from '@angular/router';
import { routes } from './app.routes';
```

@NgModule({

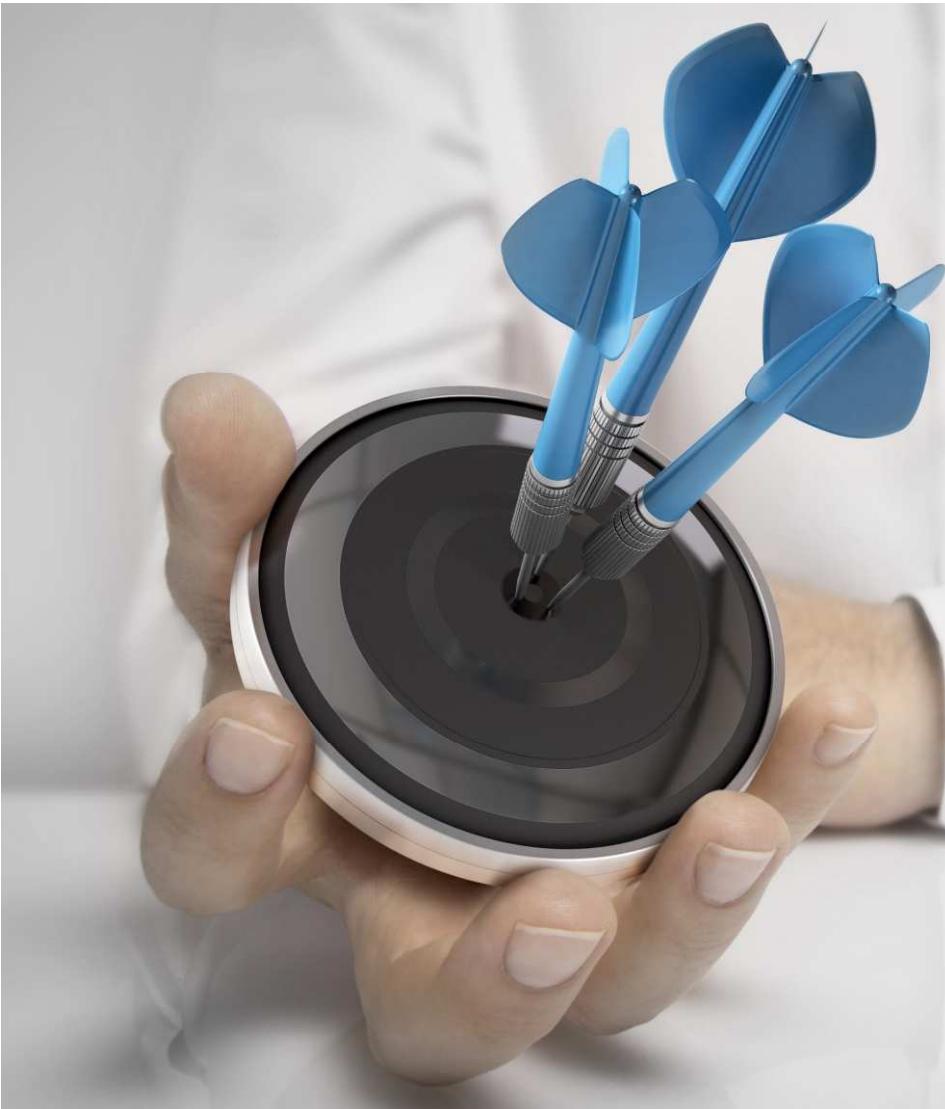
imports:[

...

```
RouterModule.forRoot(routes)
```

app.component.html

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





¿Un café?



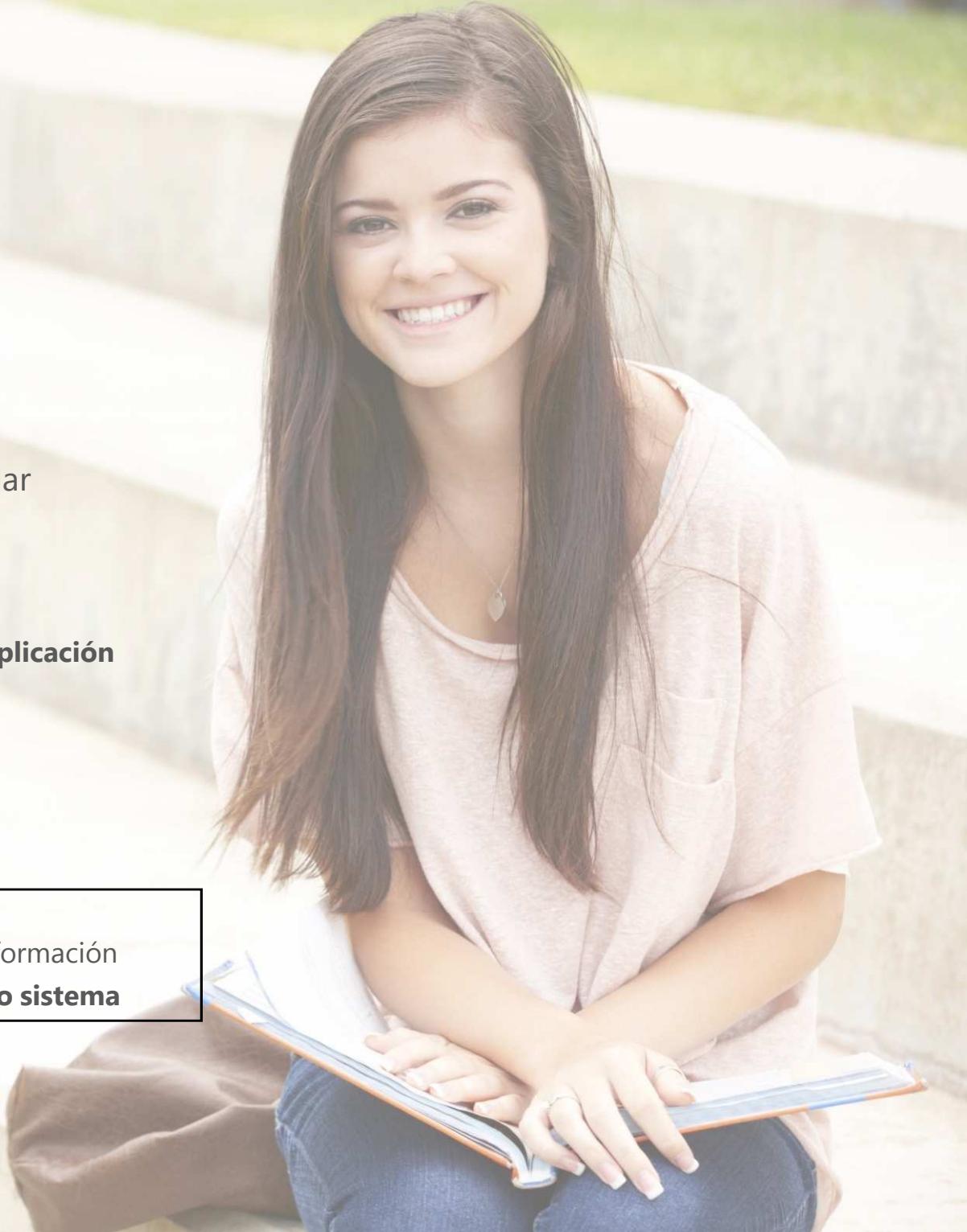
Índice

Primera sesión

1. Primeros pasos
 - a) Orígenes, historia y versiones.
 - b) Herramientas de trabajo
 - c) Instalando Angular
 - d) Mi primera aplicación**
2. Conociendo los recursos de Angular
 - a) ¿Con qué elementos contamos?
 - b) Trabajando con componentes
 - c) Implementando tuberías.
 - d) Agregando reviews a nuestra aplicación**

Segunda sesión

1. Conectando nuestras páginas
 - a) Motor de rutas
 - b) Desarrollando la arquitectura**
2. Accediendo a datos
 - a) Servicios para la obtención de información
 - b) Gestionando datos para nuestro sistema**



a

Servicios para la obtención de información

El propósito de los servicios es contener la lógica de negocio.

No tienen una directiva Service, como tienen los componentes, los módulos y tuberías que hemos visto.

Se usan en los objetos injectados a través de [Dependency Injection](#)

Para ser injectados, tenemos que marcar la clase como disponible para ser injectada. [@Injectable\(\)](#)

Para poder ser injectados, previamente han tenido que ser registrados. Existen dos formas:

- En el mismo servicio: `@Injectable({providedIn: 'root' })`;
- Con providers: [] , tanto en el módulo como en el componente, en este caso es una instancia para él mismo.

Por defecto se crea una instancia [singleton](#) por cada módulo, pero se puede configurar con [useClass](#), [useValue](#) y [useFactory](#)

Para hacer uso de los elementos injectados en los componentes por ejemplo, debemos recibirlos en el constructor: `constructor(private xxxService:XxxService) {}`

Diapositiva 43

RMN5

@Injectable({providedIn:'root'});

Se carga en el módulo raíz y es visible para toda la aplicación.

Esto es útil y cómodo en una gran cantidad de casos.

El módulo raíz es visible para toda la aplicación de forma que cualquier componente puede reclamar un servicio suyo sin problema.

Excepto que el problema sea el tamaño.

El módulo raíz se carga al arrancar y todas sus referencias van el bundle principal.

Si queremos repartir el peso debemos llevar ciertos servicios al módulo funcional que los necesite a través de providers.

<https://academia-binaria.com/servicios-inyectables-en-Angular/>

Rubén Magallón Nuño; 01/01/2019

a

Servicios para la obtención de información

[ng generate service math](#)

math.service.ts

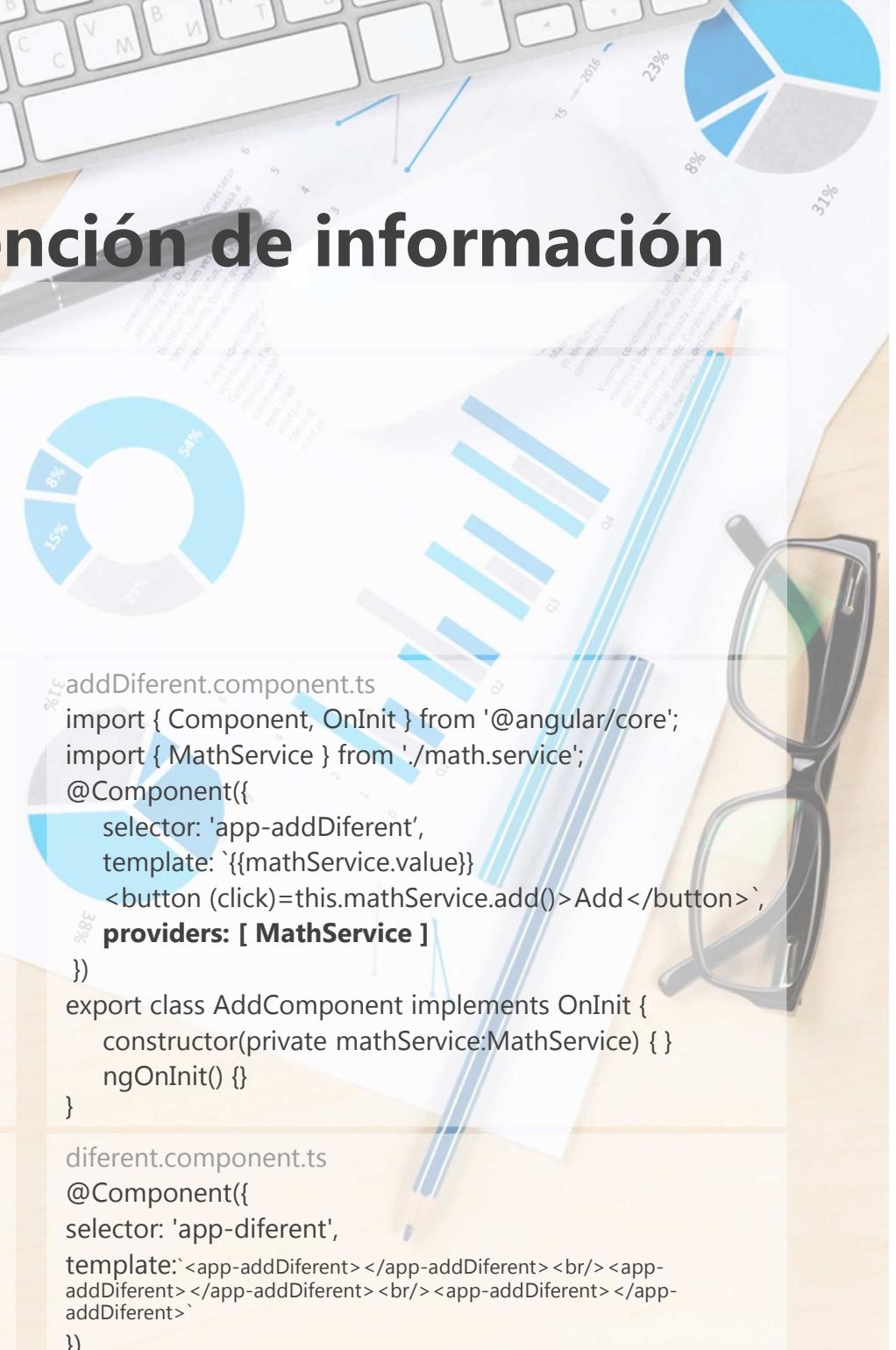
```
import { Injectable } from '@angular/core';
@Injectable()
export class MathService {
  public value:number = 0;
  constructor() {}
  add() { this.value++;}
}
```

addSame.component.ts

```
import { Component, OnInit } from '@angular/core';
import { MathService } from './math.service';
@Component({
  selector: 'app-addSame',
  template: `{{mathService.value}}
    <button (click)=this.mathService.add()>Add</button>`,
})
export class AddComponent implements OnInit {
  constructor(private mathService:MathService) {}
  ngOnInit() {}
}
```

same.component.ts

```
@Component({
  selector: 'app-same',
  template:<><app-addSame></app-addSame><br/><app-addSame></app-addSame><br/><app-addSame></app-addSame><br/><app-addSame></app-addSame></>
```



Diapositiva 44

RMN5

@Injectable({providedIn:'root'});

Se carga en el módulo raíz y es visible para toda la aplicación.

Esto es útil y cómodo en una gran cantidad de casos.

El módulo raíz es visible para toda la aplicación de forma que cualquier componente puede reclamar un servicio suyo sin problema.

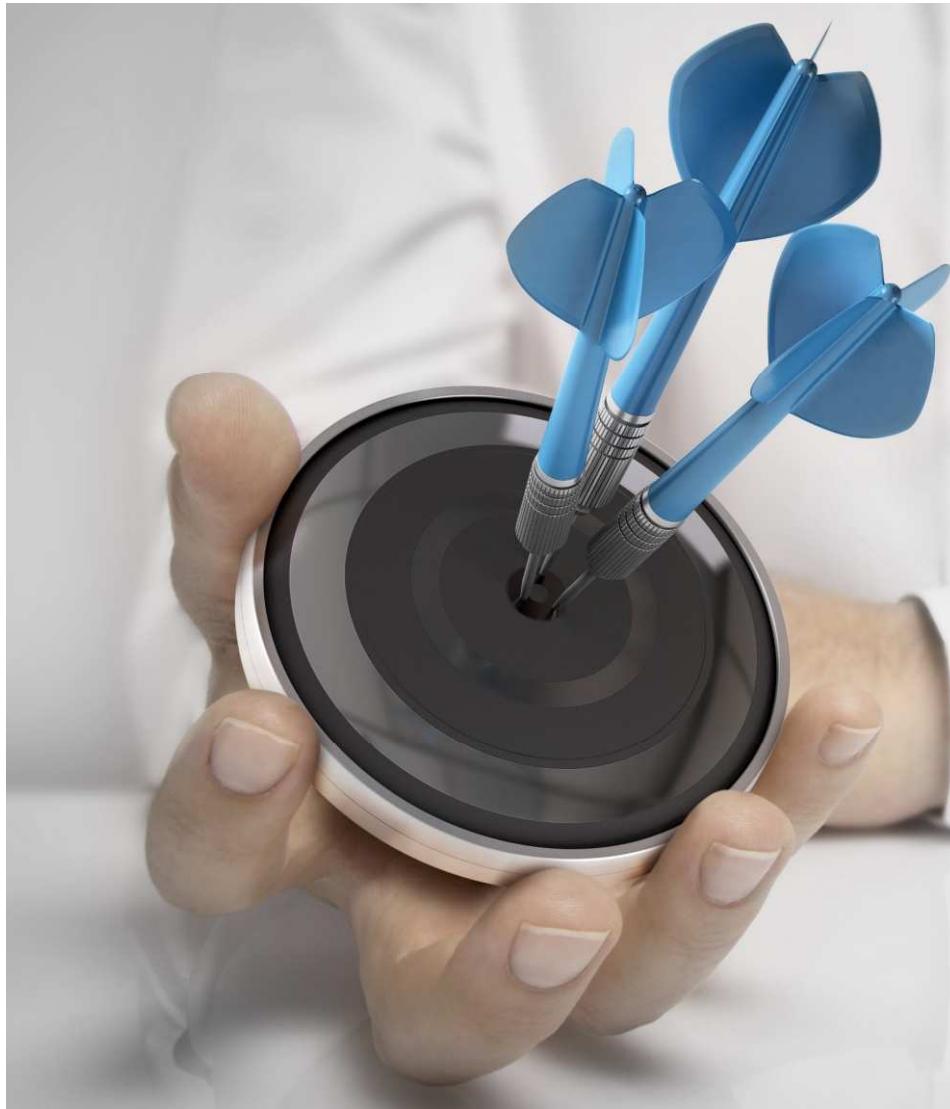
Excepto que el problema sea el tamaño.

El módulo raíz se carga al arrancar y todas sus referencias van el bundle principal.

Si queremos repartir el peso debemos llevar ciertos servicios al módulo funcional que los necesite a través de providers.

<https://academia-binaria.com/servicios-inyectables-en-Angular/>

Rubén Magallón Nuño; 01/01/2019

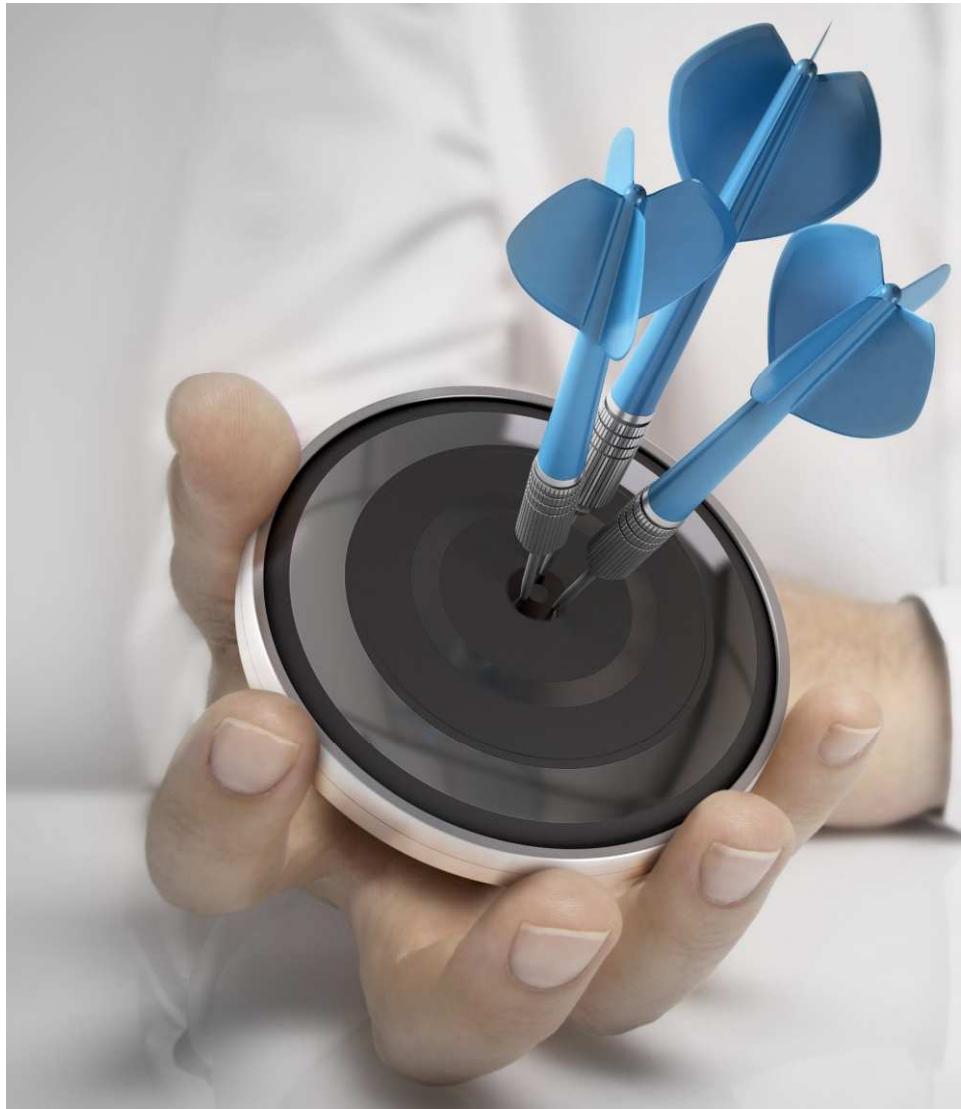
**b**

Gestionando datos

En la carpeta Film

[ng generate service Film](#)

```
import { Injectable } from '@angular/core';
import { Film } from './film';
@Injectable({providedIn:'root'})
export class FilmService {
  Films: Film[] = [{id:1, name:"", description:"", images:[], video:"", points:3.5}];
  constructor() {}
  getAll(){ return this.Films; }
  getById(id:number){ return this.Films.filter(x=> x.id ==id); }
}
```

**b**

Gestionando datos

En la carpeta Account

[ng generate class user](#)

```
user.ts
export class User {
    username:string;
    password:string;
}
```

[ng generate service account](#)

**b**

Gestionando datos

account.service.ts

```
import { Injectable } from '@angular/core';
import { User } from './user';
@Injectable({providedIn: 'root'})
export class AccountService {

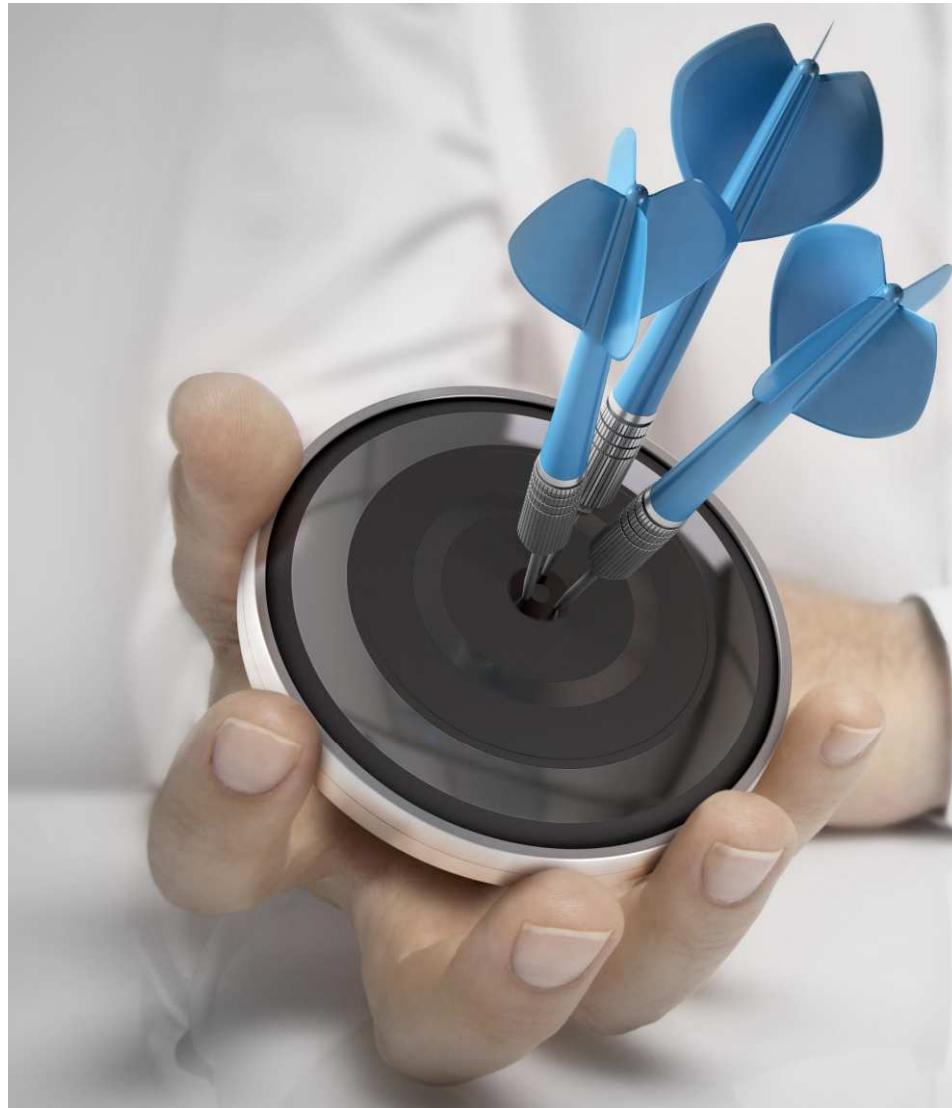
  constructor() { }

  public token:string = null;
  private users:User[] = [];

  login(username:string, password:string) {
    var user = this.users.filter(x=> x.username == username
      && x.password == password);
    if(user.length!=0) { this.token=user[0].username; return true; }
    else { this.token = null; return false; }
  }

  logout(){ this.token = null; }

  register(username:string, password:string) {
    var user = this.users.filter(x=> x.username == username);
    if(user.length==0) {
      let newUser:User = {username:username, password:password};
      this.users.push(newUser);
      return true;
    } else{ return false;}
  }
}
```

**b**

Gestionando datos

En la carpeta Account, en LoginComponent y en RegisterComponent

Importar AccountService y Router de '@angular/router'

Inyectar el servicio y Router con el constructor de ambos componentes

login.component.ts

```
Login(){  
    if(!this.accountService.login(this.username, this.password))  
        alert("Ha ocurrido un error al hacer login");  
    else{  
        this.router.navigate(['Index']);  
    }  
}
```

register.component.ts

```
Register(){  
    if(!this.accountService.register(this.username, this.password))  
        alert("Ha ocurrido un error al registrar");  
    else {  
        console.log("Registro completo");  
        this.router.navigate(['Login']);  
    }  
}
```

b

Gestionando datos

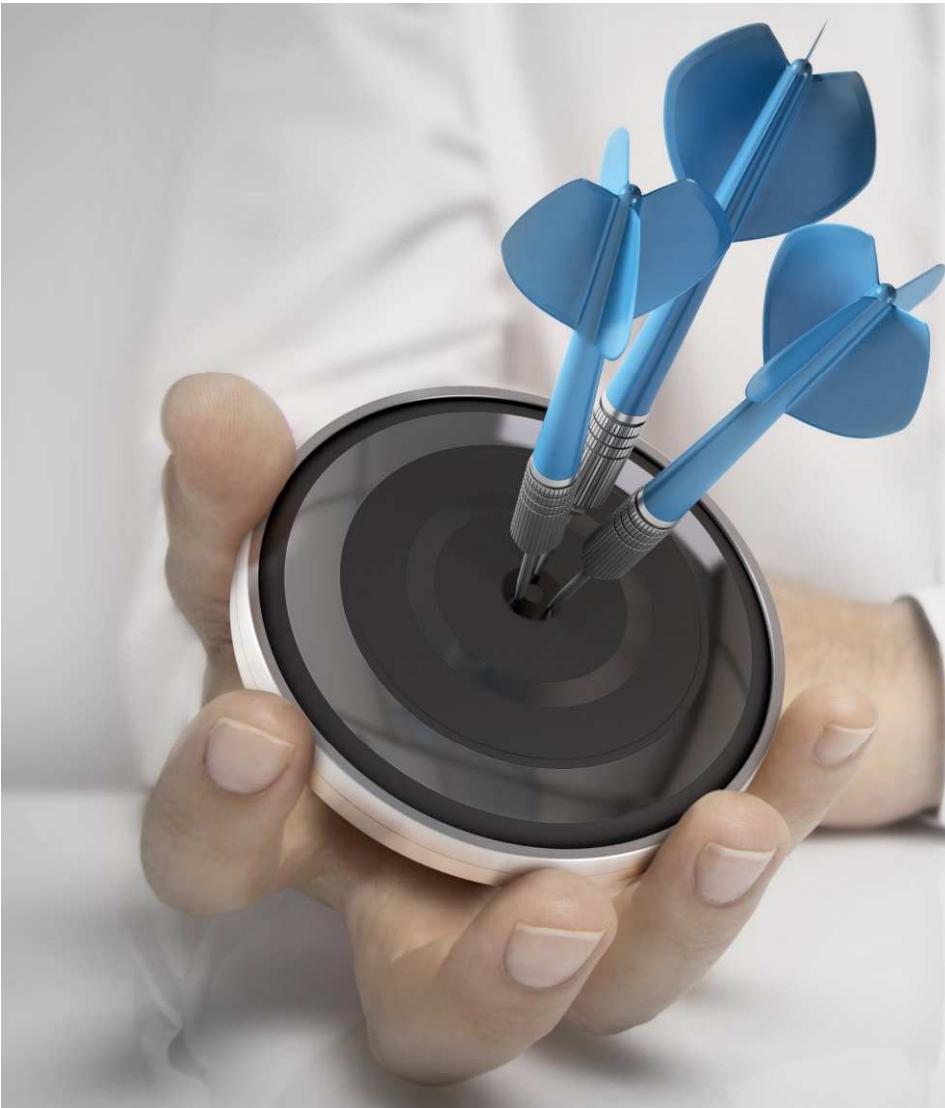
En la carpeta Film, vamos a modificar IndexComponent

Importar FilmService y Router de '@angular/router'

Inyectarlos con el constructor

index.component.ts

```
constructor(private router:Router, private  
filmService:FilmService) {  
    this.films = filmService.getAll();  
}  
ver(id:number) {  
    this.router.navigate(['View', id]);  
}
```



**b**

Gestionando datos

En la carpeta Film, vamos a modificar ViewComponent

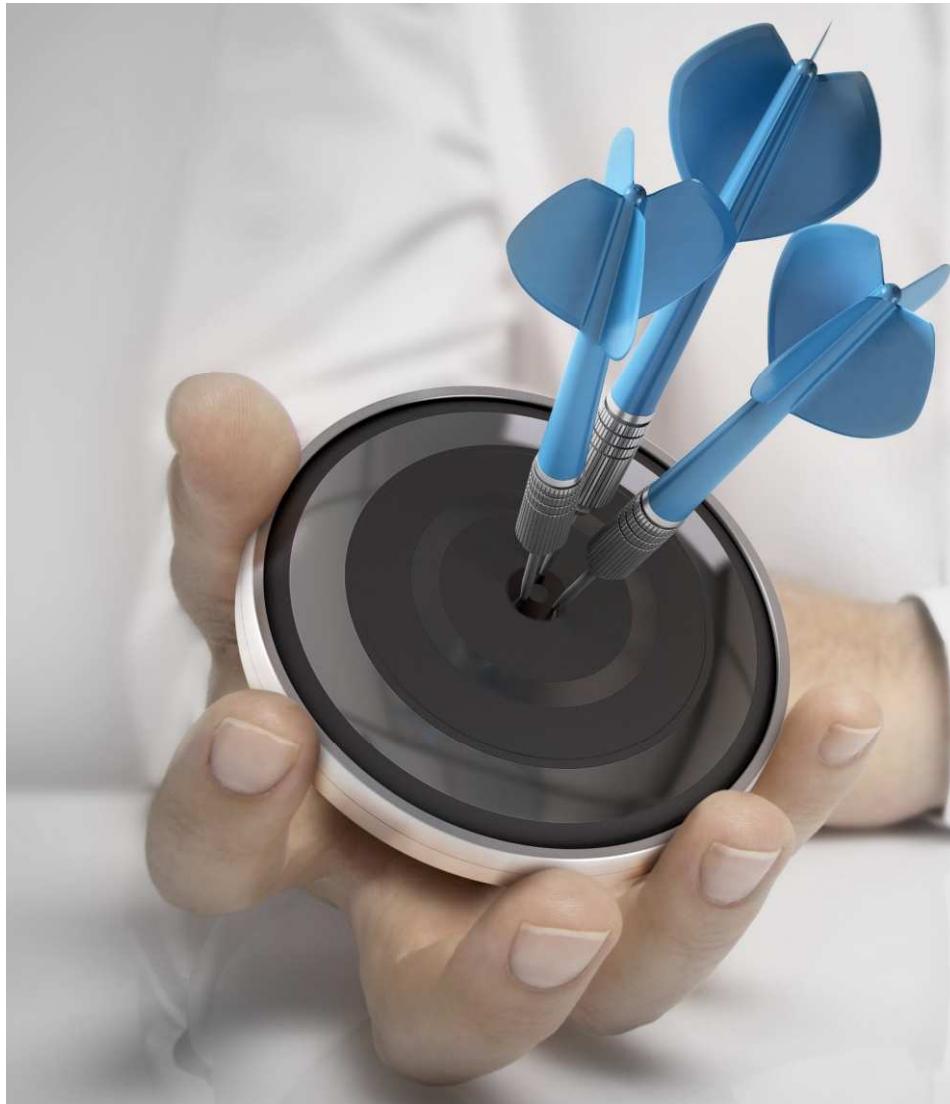
Importar Film, FilmService, AccountService y ActivatedRoute de '@angular/router'

Inyectarlos con el constructor

view.component.ts

```
id:number;  
film: Film;  
video:string;
```

```
constructor(private route: ActivatedRoute, private  
filmService:FilmService, public accountService:AccountService) {  
  this.route.params.subscribe(params => {  
    this.id = +params['id']; // (+) converts string 'id' to a number  
  });  
  let film = this.filmService.getById(this.id);  
  if(film.length!=0) {  
    this.film = film[0];  
    this.video = this.film.video;  
  }  
}  
ngOnInit() {}
```

**b**

Gestionando datos

Para ver videos de youtube necesitamos ngx-youtube-player
En la carpeta raíz del proyecto: npm i ngx-youtube-player
Importar YoutubePlayerModule de 'ngx-youtube-player'
Añadir en el imports del móduloYoutubePlayerModule

view.component.ts

```
private player; private ytEvent;  
onStateChange(event) { this.ytEvent = event.data; }  
savePlayer(player) { this.player = player; }  
playVideo() { this.player.playVideo(); }  
pauseVideo() { this.player.pauseVideo(); }
```

view.component.html

```
<div *ngIf="(accountService.token==null)">  
    Para poder ver un video necesitas estar autenticado  
</div>  
<div *ngIf="(accountService.token!=null)">  
    <youtube-player  
        [videoId]="video"  
        (ready)="savePlayer($event)"  
        (change)="onStateChange($event)">  
    </youtube-player>  
</div>
```



Quieres publicar el proyecto?

El comando ng build es lo que buscas

- Te pones en el directorio raíz y ejecuta ng build --prod
- Se crea la carpeta dist y el resultado es lo que tienes que publicar.





Quieres llevarte el proyecto?

Toda la carpeta ocupa mucho verdad?

- No copies la carpeta `node_modules`, es la que más ocupa
- No copies la carpeta `dist`, es el resultado de publicar, lo puedes hacer otra vez.
- En otro equipo te pones en el directorio raíz y ejecuta `npm install`, esto regenera la carpeta `node_modules`





Continuará...

Nos hemos dejado en el tintero:

- Servicios Http (Observables, Promesas)
- Formularios
- Guard
- Token JWT
- LocalStorage
- Pruebas unitarias
- Etc.





Gracias por tu asistencia

Esperamos haya sido de tu agrado



centro
SAN VALERO
GRUPO SANVALERO



fundación dominicana
FUNDOSVA
GRUPO SANVALERO



formación
CPA SALDUIE
GRUPO SANVALERO



estudios abiertos
SEAS
GRUPO SANVALERO



universidad
SANJORGE
GRUPO SANVALERO



SANVALERO
grupo