

Angular – introduzione

Cos'è **Angular**?

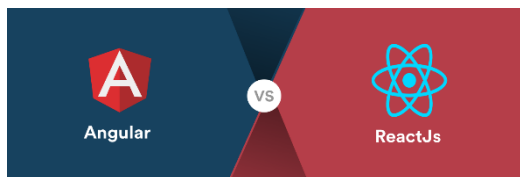
Angular è un framework open source (in informatica software non protetto da copyright).

È stato creato da Google nel 2010 e la sua prima versione si chiamava AngularJS basata sul linguaggio JavaScript, successivamente, nel 2016 è stata rilanciata una nuova versione senza il "JS" basando quest'ultima sul linguaggio TypeScript, offrendo così un'esperienza di sviluppo più moderna e potente.

Angular si basa sul concetto di Single Page Application (SPA), che significa che l'applicazione viene caricata una sola volta nel browser e le pagine vengono aggiornate dinamicamente senza dover ricaricare completamente la pagina.

Confronto fra Framework

Qual è la differenza fra Angular e React?



Sono entrambi molto popolari e potenti ma ci sono alcune differenze significative:

1. **Linguaggio**: Angular è scritto in TypeScript, React è scritto in JavaScript puro
2. **Struttura**: Angular è un framework più strutturato, React invece è una libreria più malleabile che si concentra principalmente sulla visualizzazione dei componenti
3. **Complessità**: Angular è più complesso da imparare, React risulta più semplice e flessibile

La scelta tra Angular e React dipende dalle esigenze del progetto, Angular è più adatto a livello enterprise.

Angular SPA (Single Page Application)

Come accennavo poco fa, si parla di architettura SPA quando costruiamo un'applicazione in un unico file html, dove si sviluppa tutta l'interna applicazione.

Quindi, dopo il primo caricamento della pagina iniziale, non viene più inviato HTML sulla rete, continua soltanto uno scambio di dati, che rende così l'applicazione più veloce.

Quali potrebbero essere gli svantaggi di una SPA?

A livello SEO, le SPA sono di difficile indicizzazione per i motori di ricerca, ma grazie ad Angular Universal questo problema viene risolto.



Il pre-rendering di Angular Universal è una tecnica che consente di generare versioni statiche delle pagine del tuo sito web Angular al server prima che vengano servite agli utenti.

In parole povere, dobbiamo immaginare che il server prima di mostrare l'app all'utente, se la legge e la divide, così come se fosse una MPA (multi-page application).

Atomic Design

L'Atomic Design è un metodo di programmazione (o di organizzazione del lavoro) che propone di scomporre i componenti dell'interfaccia in unità più piccole e riutilizzabili chiamati "atomi".

La gerarchia dell'Atomic Design è composta tra 5 livelli:

1. ATOMI: Sono gli elementi di base, ad esempio un pulsante
2. MOLECOLE: Sono combinazione di atomi, ad esempio un pulsante con una label
3. ORGANISMI: Sono una combinazione di molecole, ad esempio una navbar completa
4. TEMPLATE: Rappresentano la struttura di base di una pagina, tipo nav-main-footer
5. PAGINE: Sono le istanze finali dell'interfaccia utente

CLI (Angular Command Line Interface)

È un'interfaccia a riga di comando per inizializzare, sviluppare, analizzare e mantenere le applicazioni Angular, in poche parole IL TERMINALE.

Ecco alcuni comandi utili:



```
1 /*Per creare una nuova app:*/ ng new <nome dell app>
2 /*Per avviare l'app una volta nel terminale di VS Code digitare: */ ng serve -o
3 /*Per generare nuovi elementi (es. components) bisogna digitare: */ ng generate
4 /*Per aggiungere librerie e funzionalità bisogna digitare:*/ ng add
5 /*Per aggiornare librerie e progetti bisogna digitare:*/ ng gupdate
6 /*Per chiudere l'app bisogna digitare nel terminale di compilazione:*/ ctrl + c
7 /*Per reinstallare i node Modules una volta cancellati bisogna digitare:*/ npm i
8 /*Per importare Bootstrap da LOCALE con Angular bisogna digitare:*/ npm i bootstrap
9 /* e poi copiare quest'import nello style.scss globale ==> */ @import '~bootstrap/scss/bootstrap'
```

Components I

ARCHITETTURA CON I COMPONENTS

Ogni applicazione Angular ha almeno un componente, il componente radice che collega la gerarchia di componenti al DOM, ovvero l'**app.component**

Ma cos'è un componente?

Come dice la parola stessa, il component, è uno dei tanti elementi che può comporre la nostra pagina web e grazie ad Angular può diventare riutilizzabile.

È composto da tre parti principali, ovvero:

- **IL TEMPLATE**: ovvero l'HTML principale
- **CLASS**: ovvero la classe TypeScript che gestisce la logica del componente
- **METADATA**: ovvero i decorator che utilizziamo per dire al component cosa deve fare. I decorator sono funzioni speciali che vengono utilizzate per estendere o modificare un comportamento di una classe in Angular (es. *@Component*)

DATABINDING

Cosa vuol dire databinding?

Vuol dire trasmissione dei dati; da CHI a CHI

È un meccanismo che consente di collegare e sincronizzare i dati tra il TEMPLATE HTML e la classe TypeScript.

Ci sono due tipi principali di binding in Angular:

- **ONE WAY DATABINDING**(unidirezionale):

es.

Nel template HTML

```
<h1>Benvenuto, {{nome}}</h1>
```

Nella classe TypeScript

```
nome: string = 'Sofia';
```

In questo caso, semplicemente, la classe TypeScript manda il valore della variabile nome al template HTML per la stampa, ONE WAY

- **TWO WAY DATABINDING**(bidirezionale):

es.

Nel template HTML

```
<input [(ngModel)]='nome'>
<p>Ciao, {{nome}}!</p>
```

Nella classe TypeScript

```
nome: string = 'Elia';
```

In quest'altro caso, se l'utente all'interno dell'input cambia il valore della variabile nome, il Template manda il dato alla classe TypeScript, lui lo memorizza e lo rimanda indietro ed il template lo stampa, TWO WAY DATABINDING

DIRETTIVE BUILD-IN

Cosa sono le directives?

Le directives sono classi che aggiungono un **comportamento** aggiuntivo agli **elementi** nelle Applicazioni Angular.

Sono direttive predefinite fornite dal framework stesso, pronte all'uso e che ci risparmiano di scrivere righe di codice personalizzato.

Due categorie principali:

1. **DIRETTIVE STRUTTURALI**: iniziano con il prefisso `*`` e consentono di modificare la struttura del DOM manipolando la presenza o l'assenza di elementi.
 - ***ngIf** : per decidere se renderizzare o meno un elemento
 - ***ngFor** : per iterare un array e generare dinamicamente elementi HTML
2. **DIRETTIVE DI ATTRIBUTO**: vengono applicate ad un elemento HTML e forniscono comportamenti aggiuntivi
 - **ngClass** : per aggiungere o rimuovere classi CSS
 - **ngStyle** : per applicare stili CSS dinamici ad un elemento
 - **ngModel** : Fornisce il binding bidirezionale per gli elementi di input, consentendo di sincronizzare i dati tra il modello e la vista (*vedi es. two-way databinding sopra*)

NAVIGATION

Angular Routing, a cosa serve?

Il routing è essenziale per far sì che la pagina web possa funzionare come una vera SPA(Single Page Application). In pratica il routing consente di navigare tra le diverse sezioni dell'applicazione.

Con il routing ottengo:

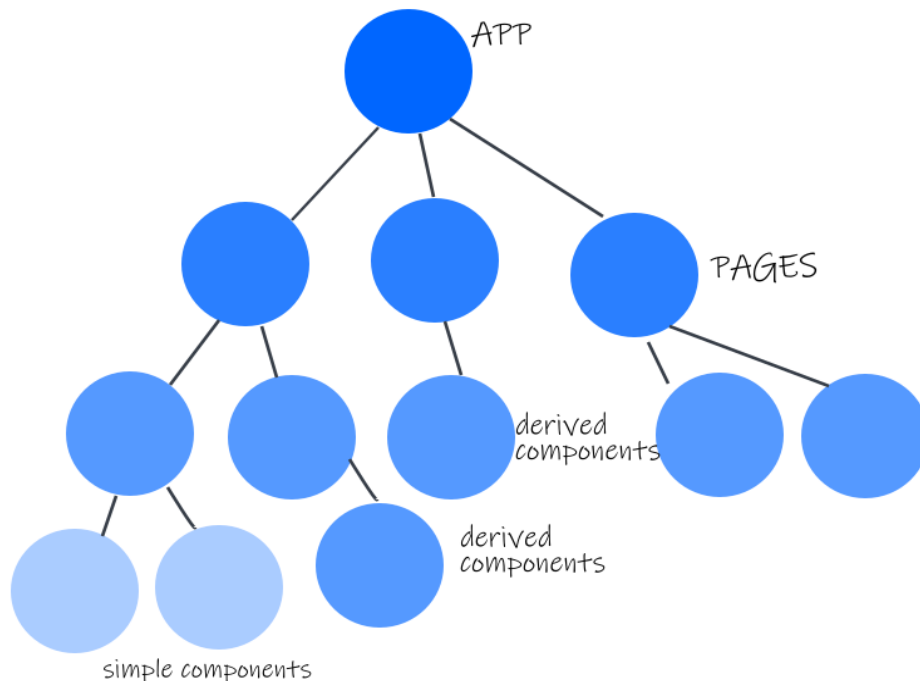
- **Navigazione tra le pagine:** Ci consente di navigare tra le pagine senza dover ricaricare di nuovo l'HTML. *(vedi descrizione SPA negli appunti precedenti)*
- **Preservo l'usabilità, grazie al mantenimento dello stato dell'applicazione:** Quindi il tasto per tornare indietro (back) funziona e consente agli utenti di tornare alla pagina precedentemente visitata.
- **Gestione dei parametri dell'URL:** ad esempio, quando un parametro dell'URL indica l'ID di un oggetto specifico da visualizzare nella pagina di dettaglio.

Il routing in Angular viene gestito attraverso il modulo `RouterModule` e la definizione della route all'interno del file di configurazione. Le route specificano le URL associate a ciascuna visualizzazione, insieme al componente da caricare e visualizzare per quella specifica route.

Components II

CONDIVISIONE DATI TRA COMPONENTS

In una struttura di solito avremo sempre come componente principale/padre `app.component.html`, che a sua volta avrà al di sotto gli altri component figli che formeranno la nostra pagina web, si crea così quella che viene chiamata gerarchia dei component



Invece, per definire il passaggio dei dati tra un componente padre e un componente figlio possiamo utilizzare i decorator `@Input` e `@Output`,

dove `@Input` consente di passare dei dati al componente figlio, e `@Output` consente al componente figlio di emettere degli eventi che possono essere ascoltati e gestiti dal componente padre.

! adesso però al posto di `@Output` sono molto più utilizzati gli `observables`

INTERAZIONE TRA COMPONENTS

Cos'è il Content Projection?

Il Content Projection (*nota anche come slot singolo o multi-slot di proiezione*) è una tecnica di Angular che ci permette di inserire del contenuto all'interno di un component, che gli viene mandato da un component genitore.

Nel component del figlio possiamo scrivere dove vogliamo `<ng-content><ng-content>`, nel component genitore invece scriviamo `<app-child-component><app-child-component>`.

es.

nel component child

```
1 <div class="child-component">
2   <h2>Titolo del Componente Figlio</h2>
3   <ng-content></ng-content>
4 </div>
```

nel component parent

```
1
2 <app-child-component>
3   <p>Questo è il contenuto inserito nel punto di inserimento del componente figlio.</p>
4 </app-child-component>
5
```

Nella versione multi-slot invece, si scrive così:

```
<ng-content select="[qui dentro si scrive il nome dell'elemento specifico che si vuole inserire]"><ng-content select>
```

Cos'è @ViewChild()?


In Angular, `@ViewChild()` è un decoratore predefinito che ci consente di accedere ad un componente figlio che si trova all'interno di un componente genitore. È una delle modalità di interazione tra componenti in Angular.

Ecco come viene utilizzato:

Nel componente genitore, importa `@ViewChild()`

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

All'interno del componente genitore, dichiara una variabile utilizzando `@ViewChild()` e specifica il selettore del componente o dell'elemento che desideri ottenere dal figlio




```
1 @ViewChild(NomeComponenteFiglio) nomeVariabile: TipoComponenteFiglio;
```

Cos'è una Local Variable?

Le Variabili Locali in Angular sono un altro modo per far interagire i component fra di loro, possiamo immaginarle come delle “etichette” che assegniamo a un elemento nel nostro template per poi poterlo richiamare successivamente.

es.


Nel template del componente figlio, assegniamo una variabile locale utilizzando ‘#’



```
1 <app-child-component #childComponentRef></app-child-component>
```

D’ora in poi, il componente padre potrà richiamare il componente figlio utilizzando questa variabile, potendo accedere a proprietà e metodi.

es.

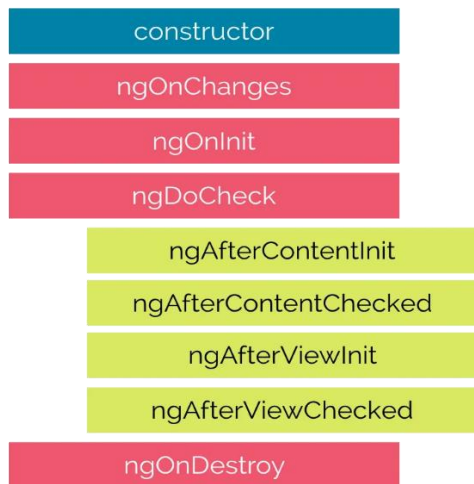


```
1 <app-child-component #childComponentRef></app-child-component>
2 <p>{{ childComponentRef.nomeProprieta }}</p>
3 <button (click)="childComponentRef.metodo()"></button>
```

Nell’esempio sopra, stiamo utilizzando la variabile locale #childComponentRef per accedere all’eventuale nome di una proprietà e all’eventuale metodo del componente figlio, ovvero <app-child-component>

COMPONENT LIFECYCLE

Cosa succede durante il ciclo di vita di un componente?



constructor() → Inizializza la classe del componente

ngOnChanges() → Risponde ai cambiamenti delle proprietà di input

ngOnInit() → Inizializza il componente dopo la configurazione delle proprietà

ngDoCheck() → Esegue controlli personalizzati per rilevare cambiamenti

ngAfterContentInit() → Inizializza il contenuto proiettato all'interno del componente

ngAfterContentChecked() → Verifica se il contenuto proiettato è stato modificato

ngAfterViewInit() → Inizializza la vista del componente

ngAfterViewChecked() → Verifica se la vista del componente è stata modificata

ngOnDestroy() → Esegue operazioni di pulizia prima della distruzione del componente

I lifecycle sono impliciti in un componente e vengono eseguiti sempre; li importiamo quando in conseguenza di un lifecycle debba accadere qualcosa.

LO STILE DEI COMPONENT

Cos'è la View Encapsulation?

Grazie alla View Encapsulation possiamo decidere per **ogni componente** come si deve comportare lo **stile**, in 3 modalità:

1. **Emulated(predefinito):** Simula lo shadow DOM nativo del browser. Creando delle classi uniche per ogni componente ma vede anche gli stili globali
2. **None:** In questa modalità, gli stili definiti all'interno di un componente non vengono incapsulati e possono essere applicati globalmente all'intera applicazione. Gli stili definiti all'interno di un componente possono interferire con gli stili di altri componenti o con lo stile globale dell'applicazione. Questa modalità dovrebbe essere utilizzata con cautela.
3. **ShadowDom:** Specifico nelle SPA, questa modalità utilizza il vero shadow DOM del browser per incapsulare gli stili all'interno del componente.

Dependency Injection

DI & SERVICE

Cosa sono i services?

I services sono un insieme di metodi che servono a più component. Infatti, vengono scritti in una cartella a parte (che si chiamerà *service*) e poi vengono importati tutti insieme nel .ts nei vari component e il service viene dichiarato nel costruttore.

Anche stavolta, questo ci permette di rendere il codice più riutilizzabile ed efficiente.

```
5 //importo il service
6 import { TodoService } from 'src/app/service/todo.service';
7
8
9 You, 4 minuti fa | 1 author (You)
10 @Component({
11   selector: 'app-todo',
12   templateUrl: './todo.component.html',
13   styleUrls: ['./todo.component.scss']
14 })
15 export class TodoComponent implements OnInit {
16
17   inputValue: string = '';
18
19   constructor(private todoSrv: TodoService) {}
20
21   ngOnInit(): void {
22   }
23 }
24
```

! È bene sapere che i service posso essere utilizzati anche in altri due modi che risultano essere meno usati, ad esempio:

1. Importando il service nell' .ts nell'app module e dichiararlo nell'array providers del decoratore @NgModule
2. Importando singolarmente i vari metodi del service che ci servono nel .ts nel componente

PIPES

Le pipe vengono utilizzate per trasformare e formattare i dati in modo da visualizzarli nel modo desiderato all'interno del template.

Le più comuni sono:

- **DataPipe** → formatta una data in base a un formato specifico
- **UpperCasePipe** → converte una stringa in maiuscolo
- **LowerCasePipe** → converte una stringa in minuscolo

- **CurrencyPipe** → formatta un numero come una stringa di valuta, utilizzando il simbolo di valuta specificato
- **DecimalPipe** → formatta i numeri in una rappresentazione decimale
- **PercentPipe** → formatta i numeri come percentuali

es.

```
<p>La data corrente è: {{ currentDate | date }}</p>
```

In questo esempio, **currentDate** è una variabile nel componente Angular e la pipe **date** viene utilizzata per formattare la data prima di visualizzarla nel template.

! È possibile anche creare delle Custom Pipes.

ROUTING

Come anticipato prima (vedi **NAVIGATION**), il routing in Angular è un meccanismo che ci consente di gestire la navigazione tra le pagine all'interno dell'applicazione.

Voglio approfondire la gestione delle **rotte nidificate**:

Angular ci consente anche di definire rotte nidificate, in cui un componente può avere rotte interne. Questo può essere utile quando vogliamo creare una struttura gerarchica delle viste per gestire le categorie e i prodotti in modo organizzato.

es.

Supponiamo che il nostro negozio online abbia le seguenti categorie di prodotti: "Abbigliamento", "Scarpe", "Accessori".

< DONNA	×
Novità	▼
Abbigliamento	▼
Lingerie	▼
Scarpe	
Accessori	▼

Definiamo le rotte principali per le categorie all'interno del nostro modulo di routing.

```
const routes: Routes = [
  { path: 'abbigliamento', component: ClothingComponent },
  { path: 'scarpe', component: ShoesComponent },
  { path: 'accessori', component: AccessoriesComponent },
];
```

Ora, all'interno di ogni componente principale, possiamo definire ulteriori rotte nidificate per i prodotti specifici di quella categoria.

< DONNA

×

Novità

▼

Abbigliamento

^

T-shirt, top, canotte

Camicie, bluse

Vestiti

```
const routes: Routes = [  
  { path: 'abbigliamento', component: ClothingComponent, children: [  
    { path: 'magliette', component: MaglietteComponent },  
    { path: 'camicie', component: CamicieComponent },  
    { path: 'vestiti', component: VestitiComponent }  
  ]},  
];
```

Quando un utente naviga alla rotta principale di una categoria, ad esempio `'/abbigliamento'`, verrà caricato il componente **ClothingComponent**. All'interno di **ClothingComponent**, il suo template potrà contenere un'area `<router-outlet>` che verrà utilizzata per visualizzare i componenti corrispondenti alle rotte nidificate. Ad esempio, se l'utente naviga a `'/abbigliamento/magliette'`, il componente `T-shirtsComponent` verrà caricato e visualizzato all'interno dell'area `<router-outlet>` di **ClothingComponent**.

Questo ci consente di gestire in modo separato le logiche specifiche per le categorie principali e per i prodotti all'interno di ciascuna categoria. Possiamo avere i nostri componenti dedicati per la visualizzazione dei prodotti, la gestione del carrello, i dettagli del prodotto, ecc.