Component

È una classe Angular responsabile di esporre dei dati su una view, e di gestire tutte le interazioni dell'utente.

Il componente è uno dei più importanti elementi costitutivi di Angular.

Si tratta, in pratica, di una direttiva con un template.

Tutti i componenti risiedono all'interno della cartella `src/app`, ognuno all'interno della propria cartella.

L'unica eccezione è `app-root`.

```
-src
--app
---app.component.ts
---message
----...
```

---

nome file

Il nome del file deve rispecchiare il seguente pattern:

`comp-name.component.ts`

Un componente, per essere tale, ha bisogno anche di una view(.html). Lo standard per il nome del file è il medesimo del componente: `comp-name.component.html`

```
-src
--app
---message
----message.component.ts
----message.component.html
```

---

il component è una classe decorata con l'annotazione `@Component`

```
/* src/app/message/message.component.ts */
import { Component } from '@angular/core';
@Component({
 selector: 'app-message',
 templateUrl: './message.component.html'
})
export class AppComponent {
}
```

Registrare il nuovo modulo in `app.module.ts`

Prima di utilizzare un componente, bisogna registrarlo all'interno del modulo.

```
/* src/app/app.module.ts */
import { MessageComponent } from './message/message.component';

@NgModule({
 declarations: [
   AppComponent,
   MessageComponent
 ],
 imports: [
   BrowserModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
```

utilizzare il component

Una volta registrato, il componente può essere utilizzato all'interno di qualsiasi altro componente.

```
/* src/app/app.component.html */
<app-message></app-message>
<hr />
<app-message></app-message>
```

Decorator

È una funzione che aggiunge dei meta-dati ad una classe, ai suoi membri (proprietà e metodi) o agli argomenti di una funzione.

I decorator sono una funzionalità sperimentale introdotta in JavaScript: TypeScript ne permette l'utilizzo fin da subito.

Un decorator va inserito immediatamente sopra o sulla sinistra dell'oggetto da decorare.

All'interno di Angular, ogni cosa è una classe: moduli, componenti, service, pipe, ecc, sono tutte delle semplici classi.

Grazie ai decorator, possiamo decidere che cosa rappresenta quella classe e farla agire come un componente, piuttosto che come un modulo, ecc.

decorare un modulo.

```
@NgModule()
/* src/app/app.module.ts */
import { NgModule } from '@angular/core';
@NgModule({
 declarations: [
   AppComponent
 ],
 imports: [
   BrowserModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
```

---

declarations

Un array contenente tutti i componenti da utilizzare per questo modulo.

```
/* src/app/app.module.ts */
import { NgModule } from '@angular/core';

@NgModule({
 declarations: [
   AppComponent
 ],
 imports: [
   BrowserModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
```

---

imports

Un array contenente tutti i moduli richiesti per questo modulo. BrowserModule è il modulo necessario per far funzionare l'applicazione su un browser.

```
/* src/app/app.module.ts */
import { NgModule } from '@angular/core';

@NgModule({
 declarations: [
   AppComponent
 ],
```

```
 imports: [
   BrowserModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
```

---

providers

Un array contenente tutti i service condivisi da tutti i componenti. Non per forza di cose, un service deve essere riportato su un modulo.

```
/* src/app/app.module.ts */
import { NgModule } from '@angular/core';

@NgModule({
 declarations: [
   AppComponent
 ],
 imports: [
   BrowserModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
```

---

bootstrap

Un array contenente il componente di root, ossia il componente che viene inizializzato al primo avvio e che contiene tutti gli altri componenti.

```
/* src/app/app.module.ts */
import { NgModule } from '@angular/core';
@NgModule({
 declarations: [
   AppComponent
 ],
 imports: [
   BrowserModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
```

---

decorare un componente

@Component()

```
/* src/app/app.component.ts */
import { Component } from
'@angular/core';
@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {
}
```

---

selector

Il nome del **tag HTML** del componente. Può essere utilizzato anche come attributo o come classe (entrambi utilizzati molto di rado)

```
/* src/app/app.component.ts */
import { Component } from '@angular/core';
@Component({
 selector: 'app-root',
 ...
})
export class AppComponent {
}
```

---

templateUrl

L'url della view associata a questo componente

```
/* src/app/app.component.ts */
import { Component } from '@angular/core';
@Component({
 templateUrl: './app.component.html',
 ...
})
export class AppComponent {
}
```

---

template

La view può essere definita anche inline, utilizzando le string interpolation (back-tick)

```
/* src/app/app.component.ts */
import { Component } from '@angular/core';
@Component({
 template: `<p>My Comp</p>`,
 ...
})
export class AppComponent {
}
```

---

styleUrls

Array contenente tutti i file CSS associati a questo componente.

```
/* src/app/app.component.ts */
import { Component } from '@angular/core';
@Component({
 styleUrls: ['./app.component.css'],
 ...
})
export class AppComponent {
}
```

---

styles

Array contenente gli stili CSS associati a questo componente, scritti direttamente all'interno del componente.

```
/* src/app/app.component.ts */
import { Component } from '@angular/core';
@Component({
 styles: [`p{color:red;}`],
 ...
})
export class AppComponent {
}
```

---

encapsulation

Specifica come template e stili vengono incapsulati all'interno del componente.

```
/* src/app/app.component.ts */
import { Component } from '@angular/core';
@Component({
 encapsulation: ViewEncapsulation.Emulated,
```

```
... })
export class AppComponent {
}
```

I tre possibili valori sono: Emulated, Native e None.