



Introduzione ad Angular 16

A cura di Christian Girardi

Installazione

Per questo corso, installeremo la versione di angular 16.2.10 la versione LTS.
Per fare questo partiamo installando nel nostro prompt dei comandi:

```
npm i @angular/cli@16.2.10
```

Successivamente apriamo VSC ed iniziamo a creare un nuovo progetto:

```
Ng new nuovoprogetto
```



Installazione

A questo punto il prompt dei comandi ci chiederà:

```
PROBLEMI  OUTPUT  TERMINALE  PORTE  CONSOLE DI DEBUG

Microsoft Windows [Versione 10.0.19045.3693]
(c) Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\Christian\Desktop\da cancellare html>cd angular

C:\Users\Christian\Desktop\da cancellare html\angular>ng new progettonuovo
? Would you like to add Angular routing? (y/N) █
```

Scriviamo N

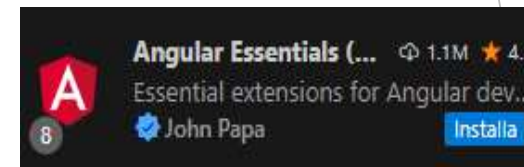
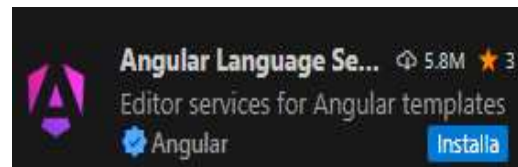
Selezioniamo CSS

Ed attendiamo la fine del
processo di installazione



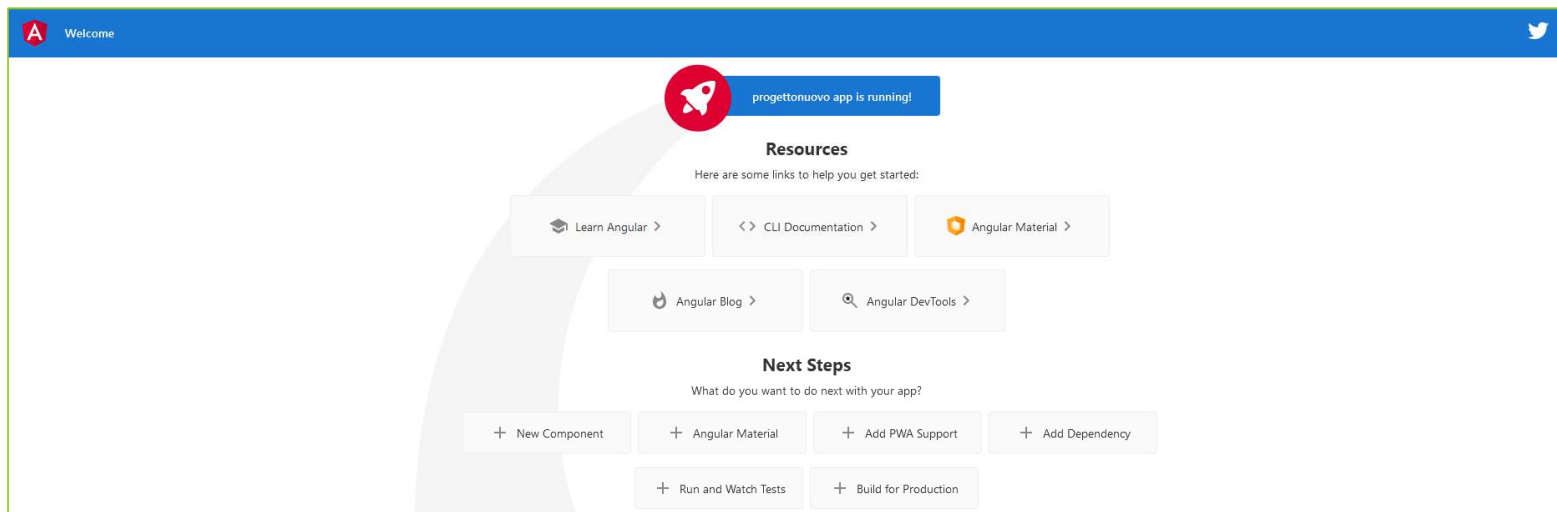
Installazione

Installiamo alcuni componenti esterni per poter lavorare piu agilmente

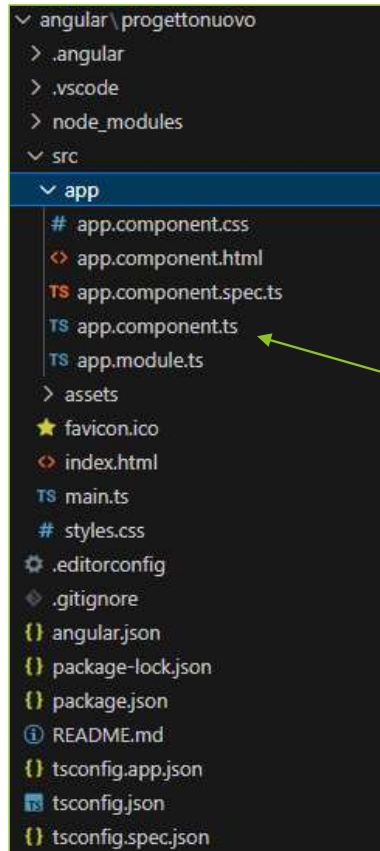


Test Avvio

Digitiamo `ng serve` ed otteniamo:



Primi passi



Parliamo di componenti, i componenti sono parti più piccole della nostra app principale.

Il bello dei componenti è la possibilità di riutilizzarli secondo necessità in più punti, e la possibilità di modificarli secondo necessità in base al contesto d'utilizzo.

Partiamo analizzando nello specifico `app.component.ts`

Questo sarà il cervello del nostro componente

Vediamo com'è strutturato:



Primi passi

```
angular > progettonuovo > src > app > TS app.component.ts > ...
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'progettonuovo';
10 }
11
```

Importiamo la libreria base di Angular per il Funzionamento.

Questa notazione dichiara il componente (Decoratore) per far capire ad angular che questo è un componente

Esportiamo il componente per poterla utilizzare.



Primi passi

```
angular > progettonuovo > src > app > TS app.component.ts > ...
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'progettonuovo';
10 }
11
```

Il selettore definisce come dichiarare
nel body il nostro componente

Dichiariamo il template Url

Dichiariamo il foglio di stile associato

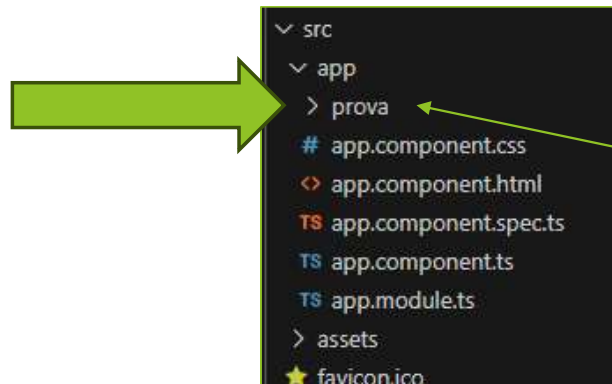


Primi passi

Andiamo a svuotare la pagina html rimuovendo tutto il codice html contenuto nella pagina app.component.html.

Ora andiamo a creare un componente utilizzando la cli per farlo velocemente utilizziamo:

Ng g c prova



Il comando ha creato i file necessari per il nuovo componente, Inoltre ha aggiornato app module con il nuovo componente

```

@NgModule({
  declarations: [
    AppComponent,
    ProvaComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```



Utilizzo bootstrap e JQuery

Integriamo bootstrap e jquery nel nostro listato, installando le relative npm:

```
npm install bootstrap --save  
npm install jquery --save  
npm install popper.js --save
```

Ora nel nostro **angular.json**:

```
"styles": [  
  "node_modules/bootstrap/dist/css/bootstrap.min.css",  
  "src/styles.css"  
],  
"scripts": [  
  "./node_modules/jquery/dist/jquery.min.js",  
  "./node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"  
]
```



Utilizzo bootstrap e JQuery

Torniamo al nostro prova.component.html e inseriamo alcuni componenti bootstrap:

```
<div class="container">
  <h1>Installare Bootstrap </h1>
  <div class="card">
    <div class="card-header">
      Caratteristiche
    </div>
    <div class="card-body">
      <h5 class="card-title">Titolo</h5>
      <p class="card-text">Testo da inserire per testare il bootstrap.</p>
      <a href="#" class="btn btn-primary">Fai qualcosa</a>
    </div>
  </div>
</div>
```

Installare Bootstrap

Caratteristiche

Titolo

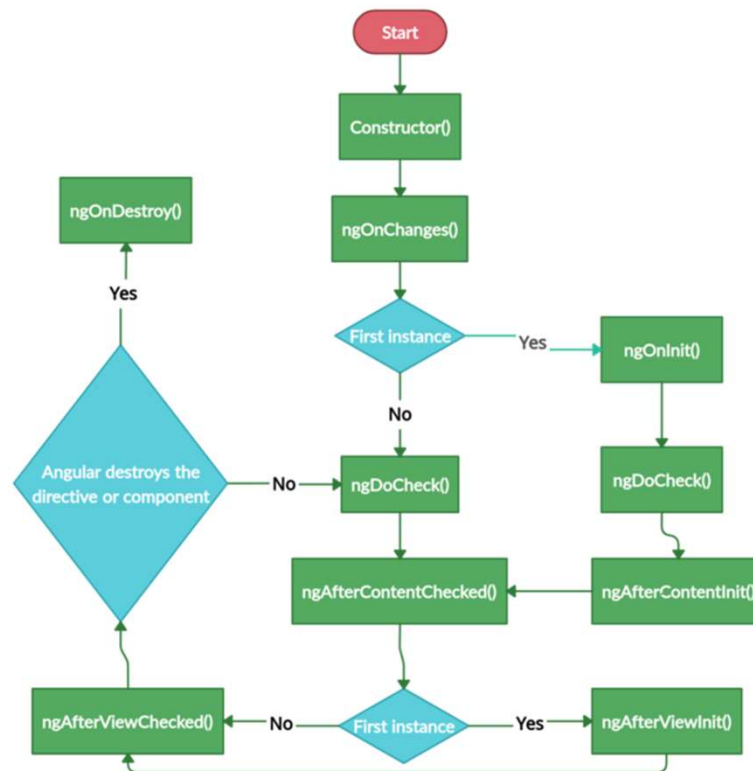
Testo da inserire per testare il bootstrap.

Fai qualcosa



Cicli di vita di un componente

<https://angular.io/guide/lifecycle-hooks>



Cicli di vita di un componente

Per verificare i cicli di vita di un componente dobbiamo modificare il nostro codice entriamo in prova.component.ts:

```
angular > progettonuovo > src > app > prova > TS prova.component.ts > .  
1  import { Component } from '@angular/core';  
2  
3  @Component({  
4    selector: 'app-prova',  
5    templateUrl: './prova.component.html',  
6    styleUrls: ['./prova.component.css']  
7  })  
8  export class ProvaComponent {  
9  
10 }  
11
```

Iniziamo ad aggiungere i vari elementi:



Cicli di vita di un componente

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

@Component({
  selector: 'app-prova',
  templateUrl: './prova.component.html',
  styleUrls: ['./prova.component.css']
})
export class ProvaComponent implements OnInit {
  constructor() {
    console.log("costruttore");
  }

  ngOnInit() {
    console.log("cngOnInit");
  }
}
```

Anche se improprio anche il costruttore fa parte del ciclo di vita del componente, per tanto utilizziamolo.

Inseriamo OnInit e scriviamo il metodo associato

Ricordiamoci che queste sono interfacce e per tanto vanno implementate



Cicli di vita di un componente

```
//
export class ProvaComponent implements OnInit,AfterContentChecked,AfterContentInit,AfterViewChecked,AfterViewInit,DoCheck,OnDestroy {
  constructor(){
    console.log("costruttore");
  }
  ngAfterViewInit(): void {
    console.log("ngAfterViewInit");
  }
  ngOnInit(){
    console.log("ngOnInit");
  }
  ngAfterContentChecked(): void {
    console.log('ngAfterContentChecked. ');
  }
  ngAfterContentInit(): void {
    console.log('ngAfterContentInit. ');
  }
  ngAfterViewChecked(): void {
    console.log('ngAfterViewChecked. ');
  }
  ngDoCheck(): void {
    console.log('ngDoCheck. ');
  }
  ngOnDestroy(): void {
    console.log('ngOnDestroy. ');
  }
}
```



Cicli di vita di un componente

```
[webpack-dev-server] Server started: Hot Module Replacement disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
costruttore
ngOnInit
ngDoCheck.
ngAfterContentInit.
ngAfterContentChecked.
ngAfterViewInit
ngAfterViewChecked.
Angular is running in development mode. Call enableProdMode() to enable production mode.
ngDoCheck.
ngAfterContentChecked.
ngAfterViewChecked.
```

Ecco la struttura di un ciclo di vita di un nostro componente.

OnDestroy al momento non viene visualizzato in quanto non stiamo ancora scartando il componente



Data Binding

In Angular, il data binding è un meccanismo che stabilisce una connessione tra il modello dei dati (componenti TypeScript) e la visualizzazione dell'interfaccia utente (template HTML).

Esistono diversi tipi di data binding in Angular:

- **Interpolation ({{ }}):** È il metodo più comune. Consente di inserire dinamicamente valori dalle proprietà del componente TypeScript direttamente nel template HTML.

Esempio:

```
<p>{{ variabile }}</p>
```





Data Binding

- **Property binding** ([property]="value"): Consente di assegnare dinamicamente un valore a una proprietà di un elemento HTML.

Esempio:

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

- **Event binding** ((event)="handler()"): Consente di rispondere agli eventi (come clic, input, ecc.) generati dall'utente nel template HTML e richiamare una funzione nel componente TypeScript.

Esempio:

```
<button (click)="onClick()">Clicca qui</button>
```

Data Binding

- **Two-way binding ([ngModel]):** È un tipo di binding che combina la proprietà binding e l'event binding. È utilizzato principalmente con le direttive come ngModel per ottenere un'associazione bidirezionale tra i dati nel modello e un elemento di input dell'utente nel template.

Esempio:

```
<input [(ngModel)]="nome">
```

Il data binding semplifica la gestione dei dati nell'applicazione, consentendo agli sviluppatori di sincronizzare automaticamente i dati tra il modello e la vista senza dover scrivere codice manuale per aggiornare costantemente l'interfaccia utente.



Data Binding

Per iniziare a comprendere il data binding partiamo utilizzando il nostro componente di prova, l'idea di base è legare i nostri componenti a schermo con dati della parte logica (nel caso di componenti ripetuti anche dati diversi)

Esistono due tipologie di data binding :

- One Way (dalla logica al componente grafico)
- Two Way (scambio bidirezionale fra le due)

Ad esempio la string interpolation è la piu semplice da rappresentare, vediamo come:



Data Binding

```
@Component({
  selector: 'app-prova',
  templateUrl: './prova.component.html',
  styleUrls: ['./prova.component.css']
})
export class ProvaComponent implements OnInit, AfterContentChecked, AfterContentInit, AfterViewChecked, AfterViewInit, DoCheck, OnDestroy {
  titolo = "test titolo stringa";
}
```

Aggiungiamo titolo

Modifichiamo il
contenuto del HTML
Inserendo {{titolo}}

```
<h1>Installare Bootstrap </h1>
<div class="card">
  <div class="card-header">
    Caratteristiche
  </div>
  <div class="card-body">
    <h5 class="card-title">{{titolo}}</h5>
    <p class="card-text">Testo da inserire per testare il bootstrap.</p>
    <a href="#" class="btn btn-primary">Fai qualcosa</a>
  </div>
</div>
```

Installare Bootstrap

Caratteristiche

test titolo stringa

Testo da inserire per testare il bootstrap.

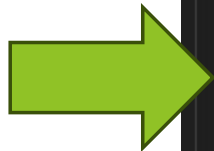
Fai qualcosa



String interpolation

Lo scopo ultimo dell'interpolazione delle stringhe è mandare a video dati(**STRINGHE**) che possono essere prelevati da più fonti, nella maggior parte dei casi la sorgente è un database, al momento ci adatteremo costruendo un contenitore di oggetti e proveremo a gestire questa sorgente dati.

Per tanto iniziamo a modificare il nostro componente:



```
titolo = "test titolo stringa";

macchine = [
  {
    modello: 'Porsche 718 Cayman',
    potenza: '500 CV'
  },
  {
    modello: 'Ferrari Daytona SP3',
    potenza: '840 CV'
  },
  {
    modello: 'Audi RS3',
    potenza: '399 CV'
  }
]
```



String interpolation

```
<h1>Installare Bootstrap </h1>
<div class="card">
  <div class="card-header">
    Caratteristiche
  </div>
  <div class="card-body">
    <h5 class="card-title">{{macchine[0].modello}}</h5>
    <p class="card-text">la potenza è: {{macchine[0].potenza}}</p>
    <a href="#" class="btn btn-primary">Fai qualcosa</a>
  </div>
</div>
```



Abbiamo modificato il codice HTML aggiungendo l'interpolazione a `macchine[0]`

Estraendo il dato con la dot notation.

`Macchina[0].modello` ad es.



String interpolation

Possiamo anche gestire nelle interpolazioni il ternary operator
condizione ? Se Vero : Se Falso

Proviamo a metterlo in pratica nel ts aggiungiamo:

```
export class ProvaComponent implements  
titolo = "test titolo stringa";  
disponibile=true;
```

Nel HTML modifichiamo:

```
<div class="card-body">  
  <h5 class="card-title">{{disponibile ? macchine[0].modello : 'non disponibile'}}</h5>  
  <p class="card-text">la potenza è: {{disponibile ? macchine[0].potenza : 'non disponibile'}}</p>  
  <a href="#" class="btn btn-primary">Fai qualcosa</a>
```

Avremmo potuto anche
inserire direttamente un
operazione di comparazione
al posto di disponibile
Ad esempio $2 < 6$ (sempre
vera)

Proviamo ora mettere false a disponibile.

Property binding

Consiste nel collegare dati alle proprietà dei nostri componenti ad esempio colore, css la classe

Prendiamo in esame il bottone (va trasformato da a button) che abbiamo già nel nostro esempio, ora proveremo a disabilitarlo da codice:



Andiamo sul html e possiamo ad esempio:

UMANA
FORMA



Property binding

```
<p class="card-text">la potenza è: {{disponibile ? macchine[0].potenza : 'non disponibile'}}</p>  
<button class="btn btn-primary" disabled="true">Fai qualcosa</button>
```

Questa operazione disabiliterà il bottone:



Ora utilizzando il ts potremmo pensare di fare:

```
titolo = "test titolo stringa";  
disponibile=true;  
bottoneattivo = false;
```



Property binding

Per correggere la situazione dobbiamo modificare il nostro codice HTML nel modo seguente:

```
<p class="card-text">la potenza è: {{disponibile ? macchine[0].potenza : 'non disponibile'}}</p>  
<button class="btn btn-primary" [disabled]="bottoneattivo">Fai qualcosa</button>  
</div>
```



Property binding

Per verificare il corretto funzionamento andiamo nel ngOnInit del nostro componente e scriviamo:

```
ngOnInit()  
  console.log("ngOnInit");  
  
  setInterval(()=>{  
    this.bottoneattivo =!this.bottoneattivo  
  },3000)
```

Al termine della verifica andiamo a commentare il codice.

Quindi abbiamo capito che possiamo collegare le proprietà dei nostri componenti racchiudendo la proprietà stessa fra parentesi [] e collegando tale proprietà la nostro ts.

Provate ora ad esempio a collegare un immagine "bindando" il parametro src





Event Binding

La capacità di Angular di reagire agli eventi ad esempio il click oppure rilevare il contenuto di un input in fase di inserimento.

Come sappiamo nei bottoni in HTML la gestione dell'evento click è rilevata e gestita da `onclick=""`, in Angular dobbiamo modificare il nostro codice HTML così:



```
<button class="btn btn-primary" [disabled]="bottoneattivo" (click)=">Fai qualcosa</button>
```

Utilizziamo `(click)` e possiamo ad esempio chiamare un metodo:

```
<button class="btn btn-primary" [disabled]="bottoneattivo" (click)="OnClick()">Fai qualcosa</button>
```

L'errore è dovuto al fatto che non abbiamo definito nel TS il metodo per tanto facciamo:

```
OnClick(){  
    
}
```



Focus: Event Binding - This

Focus: provate a passare

```
<button class="btn btn-primary" [disabled]="bottoneattivo" (click)="OnClick(this)">Fai qualcosa</button>
```

Sistemiamo il ts:

```
OnClick(e:any){  
  console.log(e);  
}
```

Guardate il risultato:

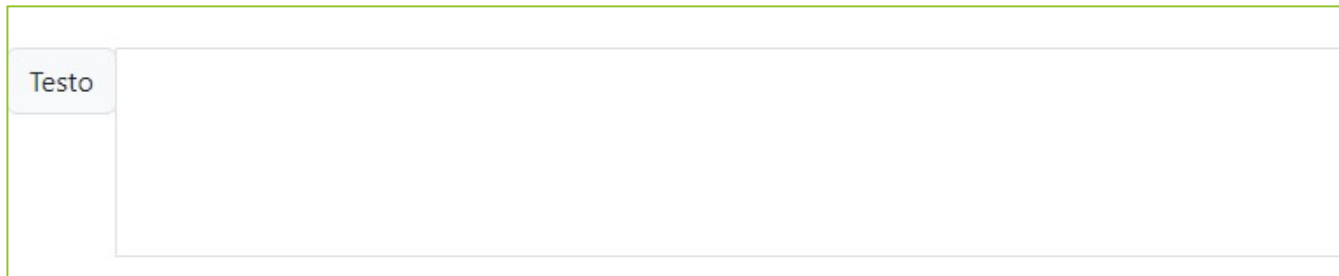
```
prova.component.ts:61  
▼ ProvaComponent {titolo: 'test titolo stringa', disponibile: true, bottone  
  attivo: false, macchine: Array(3), __ngContext__: 1} 1  
  bottoneattivo: false  
  disponibile: true  
  ► macchine: (3) [{...}, {...}, {...}]  
    titolo: "test titolo stringa"  
    __ngContext__: 1  
  ► [[Prototype]]: Object
```

Ripuliamo il codice.

Event Binding

Procediamo andando a recuperare un input ad esempio da bootstrap:

```
<div class="input-group">
  <div class="input-group-prepend">
    <span class="input-group-text">Testo</span>
  </div>
  <textarea class="form-control" aria-label="Testo"></textarea>
</div>
```

A visual representation of the HTML code above. It shows a Bootstrap input group. On the left, there is a small, light gray button-like element with the text "Testo". To its right is a large, empty text area with a thin gray border. The entire group is enclosed in a thin green rectangular frame.

Event Binding

Modifichiamo il codice aggiungendo l'evento di input:



```
<textarea class="form-control" aria-label="Testo" (input)="OnClick()"></textarea>
```

Collegiamoci a alla stessa funzione creata precedentemente, e modifichiamola:

```
OnClick(){  
  console.log("sto scrivendo");  
}
```

Avviamo il progetto e i risultati saranno evidenti nella console, facciamo però una piccola modifica utilizzando la variabile di sistema di Angular \$event

```
<textarea class="form-control" aria-label="Testo" (input)="OnClick($event)"></textarea>
```

```
OnClick(e:any){  
  console.log(e);  
}
```


Event Binding

```
prova.component.ts:61  
InputEvent {isTrusted: true, data: 's', isComposing: false, inputType: 'insertText', dataTransfer: null, ...}
```

Questo risultato è molto complesso e completo di tutte le informazioni relative al nostro evento, al momento ci interessa `e.target.value`:

```
OnClick(e:any){  
  console.log(e.target.value);  
}
```

Provate a osservare la console ed il risultato.

Nel codice avanzato si sostituisce `any` con `Event`

Ciò comporta un casting per il value necessario:

```
<HTMLInputElement>e.target).value
```



Two Way Binding

Consiste nel collegare Typescript ed il componente HTML in maniera bidirezionale, Prendiamo come esempio l'input precedentemente visto, per comodita sistemiamo OnClick in OnInput nel ts e ricreiamo un OnClick (e sistemiamo anche HTML) :


```
OnInput(e:Event){  
  console.log(<HTMLInputElement>e.target).value);  
}  
OnClick()  
}
```

Aggiungiamo nel nostro html un interpolazione a titolo:

```
<div class="input-group">  
  <div class="input-group-prepend">  
    <span class="input-group-text">Testo</span>  
  </div>  
  <textarea class="form-control" aria-label="Testo" (input)="OnInput($event)"></textarea>  
</div>  
{{titolo}}
```

Two Way Binding


Prima di procedere nel nostro app.module.TS dobbiamo importare :



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

import { AppComponent } from './app.component';
import { ProvaComponent } from './prova/prova.component';

@NgModule({
  declarations: [
    AppComponent,
    ProvaComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ]
})
```



Per collegare il nostro input a titolo dobbiamo aggiungere modificare la sintassi di input introducendo ngModel:



Two Way Binding



```
<div>
  <div class="input-group">
    <div class="input-group-prepend">
      <span class="input-group-text">Testo</span>
    </div>
    <input class="form-control" aria-label="Testo" [(ngModel)]="titolo">
  </div>
  {{titolo}}
```

Provate ad avviare e a modificare ora il contenuto dell'input

Vedrete come risulti collegato a due vie, per verificare il tutto modificate anche il bottone aggiungendo nel ts:

```
onClick(){
  this.titolo = "test fatto click";
}
```



UMANA
FORMA

