

ng-template

`<ng-template></ng-template>` este un tag care continut in interiorul lui cod html.

```
<ng-template #sayHelloTemplate>
  <p> Say Hello</p>
</ng-template>
```

Codul de mai sus nu va afisa nimic, deoarece codul html din interiorul tag-ului ng-template nu este afisat.

Pentru a afisa acest template putem folosi :

- ngTemplateOutlet directive
- TemplateRef & ViewContainerRef

ngTemplateOutlet

Este un structural directive folosit pentru a afisa un template.

Pentru a afisa template-ul trebuie sa i asignam acestuia un template variable si aceea sa o folosim ca si input pentru ngTemplateOutlet

Un `<ng-template>` este de tip TemplateRef, sayHelloTemplate are tipul TemplateRef

```
<ng-template #sayHelloTemplate>
  <p> Say Hello</p>
</ng-template>
```

```
<ng-container *ngTemplateOutlet="sayHelloTemplate">
  This text is not displayed
</ng-container>
```

Textul din ng-container va fi inlocuit cu codul html din ng-template : `<p> Say Hello</p>`

ng-container

Este doar un tag syntactic, in DOM nu va fi afisat, este folosit atunci cand nu vrem sa adaugam div-uri fara folos.

```
<h1> ng-Container</h2>
<p>Hello world! </p>
<ng-container>           //This is removed from the final HTML
  Container's content.
</ng-container>
```

Rezultatul final va fi randat astfel :

```
<h1> ng-Container</h2>
<p>Hello world! </p>
Container's content.
```

Exemple cand este util ng-container :

1.Impreuna cu *ngFor :

```
<ul>
  <span *ngFor="let item of items;">
    <li *ngIf="item.active">
      {{item.name}}
    </li>
  </span>
</ul>
```

Aici trebuie sa folosim un span in plus, deoarece nu putem avea ngFor si ngIf impreuna pe acelasi element (nu poti avea 2 directive structurale)

Putem evita asta astfel :

```
<ul>
  <ng-container *ngFor="let item of items;">
    <li *ngIf="item.active">
      {{item.name}}
    </li>
  </ng-container>
</ul>
```

2. Impreuna cu *ngIf :

```
<div *ngIf="items1">           //Replace the div with ng-container as shown below
<div *ngFor="let item of items1;">
  {{item.name}}
</div>
</div>
```

Primul div, care continue directive ngIf, este doar un element in plus ce se adauga in DOM.

Solutie :

```
<ng-container *ngIf="items1">
<div *ngFor="let item of items1;">
  {{item.name}}
</div>
</ng-container>
```

TemplateRef & ViewContainerRef

TemplateRef este o clasa si este type-ul unui template, reprezinta codul html- care se afla inuntrul <ng-template></ng-template>

In template(cod html) avem astfel :

```
<ng-template #sayHelloTemplate>
  <p> Say Hello</p>
</ng-template>
```

Pentru a accesa acest template din codul typescript(in clasa pentru componenta) ne vom folosi de decoratorul @ViewChild()

```
@ViewChild('sayHelloTemplate', { read: TemplateRef }) sayHelloTemplate:TemplateRef<any>;
```

Acum variabila sayHelloTemplate contine codul din inuntrul <ng-template>

Avem acces la codul html din inainturul `<ng-template>` acum se pune problema unde il vom afisa?

ViewContainerRef

Ca si `TemplateRef`, `ViewContainerRef` este o clasa si reprezinta locatia unui element in interiorul elementului parinte

```
constructor(private vref:ViewContainerRef) {  
}  
  
ngAfterViewInit() {  
  this.vref.createEmbeddedView(this.sayHelloTemplate);  
}
```

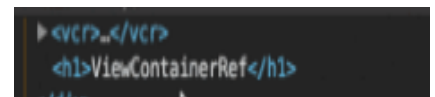
Daca obtinem `ViewContainerRef` prin DI, atunci vom avea locatia elementului actual in interiorul elementului parinte

Folosind `createEmbeddedView` vom insera acel template imediat dupa elementul actual(cel care are `viewContainerRef` in constructor)

Exemplu 1 :

```
@Component({  
  selector: 'vcr',  
  template: `  
    <template #tpl>  
      <h1>ViewContainerRef</h1>  
    </template>  
  `,  
})  
export class VcrComponent {  
  @ViewChild('tpl') tpl;  
  constructor(private _vcr: ViewContainerRef) {  
  }  
  
  ngAfterViewInit() {  
    this._vcr.createEmbeddedView(this.tpl);  
  }  
}  
  
@Component({  
  selector: 'my-app',  
  template: `  
    <vcr></vcr>  
  `,  
})  
export class App {  
  
}
```

Rezultat:



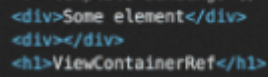
Prin constructor se obtine locatia elementului VCR in pagina, apoi prin `createEmbeddedView` adaugat template-ul imediat dupa

Exemplu 2:

```
@Component({
  selector: 'vcr',
  template: `
    <template #tpl>
      <h1>ViewContainerRef</h1>
    </template>
    <div>Some element</div>
    <div #container></div>
  `,
})
export class VcrComponent {
  @ViewChild('container', { read: ViewContainerRef }) _vcr;
  @ViewChild('tpl') tpl;

  ngAfterViewInit() {
    this._vcr.createEmbeddedView(this.tpl);
  }
}

@Component({
  selector: 'my-app',
  template: `
    <div>
      <vcr></vcr>
    </div>
  `,
})
export class App {
}
```



```
<div>Some element</div>
</div>
<h1>ViewContainerRef</h1>
```

Prin ViewChild obtinem locatia div-ului in pagina, apoi inseram template-ul mediat dupa acel div

Directive structurale si sugar syntax

1.*ngIf

```
<div *ngIf="selected">
  <p>You are selected</p>
</div>
```

Angular face urmatoarele schimbari la run-time

```
<ng-template [ngIf]="selected">
  <div>
    <p>You are selected</p>
  </div>
</ng-template>
```

2.*ngFor

```
<ul>
  <li *ngFor="let movie of movies ">
    {{ movie.title }} - {{movie.director}}
  </li>
</ul>
```

Devine

```
<ul>
<ng-template
  ngFor let-movie [ngForOf]="movies">

  <li>
    {{ movie.title }} - {{movie.director}}
  </li>

</ng-template>
</ul>
```

Documentatie :

[How to use ng-template & TemplateRef in Angular - TekTutorialsHub](#)

[Understanding ViewContainerRef in Angular 2 | by Netanel Basal | Netanel Basal \(netbasal.com\)](#)

[Angular ng-template, ng-container and ngTemplateOutlet: Guided Tour \(angular-university.io\)](#)

[Angular - Writing structural directives](#)

[ng-container in Angular - TekTutorialsHub](#)