Fundamentals @2

<ng-template> in Angular

Notes on `ng-template` in Angular

Definition:

`ng-template` is an Angular element used to define a template that contains HTML snippets. However, the contents of `ng-template` do not get rendered in the DOM by default. It is primarily used to create reusable templates that can be conditionally rendered in the DOM.

Key Points:

- `ng-template` defines a block of HTML, but it is not displayed until explicitly told to render.
- The template is rendered using structural directives like `ngTemplateOutlet`.

```
<h2>Learn ng-template</h2>

<!-- Define a template -->
<ng-template #myTemplate>
  <h3>This is a template</h3>
  <p>This is an example paragraph to understand ng-template.</p>
</ng-template>

<!-- Render the template using ngTemplateOutlet -->
<div *ngTemplateOutlet="myTemplate"></div>
```

Explanation:

1. The `ng-template` contains an `h3` and `p` element but will not render until specified.
2. The `ngTemplateOutlet` directive is used on a `div` to render the `ng-template`.

Rendering Logic with `ngTemplateOutlet`:

- `ngTemplateOutlet` is a structural directive used to inject the content of a `ng-template` into the DOM. A template reference variable (e.g., `#myTemplate`) is assigned to the template and passed to `ngTemplateOutlet`.

Example for Practical Use:

In an Angular project, you can conditionally show buttons based on product availability:

```
<!-- Add to Cart Button for Available Products -->
<button *ngIf="product.isInInventory; else notifyTemplate">
  Add to Cart
</button>

<!-- Notify Button for Unavailable Products (defined in ng-template) -->
<ng-template #notifyTemplate>
  <button>Notify Me</button>
</ng-template>
```

The ng-template is an Angular element which wraps an HTML snippet. This HTML snippet acts and can be used like a template and can be rendered in the DOM.

app.component.html

src > app > app.component.html > ng-template > p

Go to component

```
1  <h2>Learn NG Template</h2>
2  <ng-template>
3    <h3>This is a template</h3>
4    <p>This is an example paragraph to understand ng-template</p>
5  </ng-template>
```


```
<h2>Learn NG Template</h2>

<ng-template #myTemplate>
  <h3>This is a template</h3>
  <p>This is an example paragraph to understand ng-template</p>
</ng-template>

<!--ngTemplateOutlet Directive-->

<div *ngTemplateOutlet="myTemplate"></div>
```

<ng-container> in Angular

The ng-container is a special Angular element that can hold structural directives without adding new elements to the DOM.

Notes on `ng-container` in Angular

Definition:

`ng-container` is a special Angular element that allows you to apply structural directives like `*ngIf`, `*ngFor`, etc., without adding extra elements to the DOM. It helps in organizing templates cleanly without creating unnecessary wrapper elements in the rendered HTML.

example of extra elements would be like div or section wrapper elements.

Key Points:

- `ng-container` does not render as a real HTML element in the DOM.
- It is used to apply structural directives when you do not want an extra wrapper element.

Example 1: Using `ng-container` with `ngIf`

```
<button (click)="onToggle()">Toggle</button>

<ng-container *ngIf="toggle; else toggleOff">
  <p>The toggle is on</p>
</ng-container>

<ng-template #toggleOff>
  <p>The toggle is off</p>
</ng-template>
```

Explanation:

- The button toggles the `toggle` boolean property.
- When `toggle` is `true`, the paragraph inside the `ng-container` is displayed.
- When `toggle` is `false`, the content inside the `ng-template` (`toggleOff`) is shown instead.

Example 2: Using `ng-container` to Avoid Extra `div`

Copy code

```
<ng-container *ngFor="let product of products">
  <app-product [product]="product"></app-product>
</ng-container>
```

Explanation:

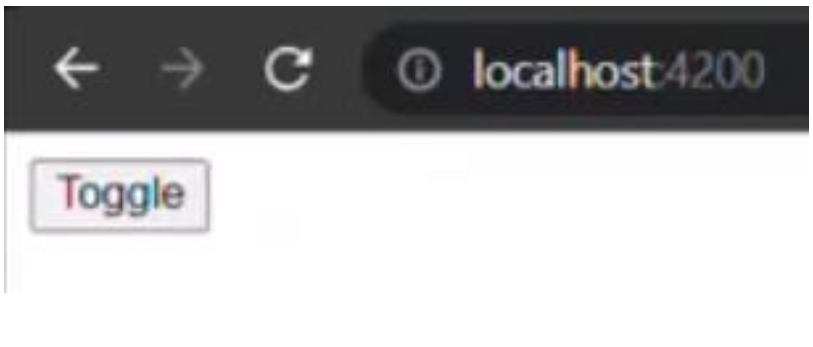
- Instead of using a `div` with `*ngFor` to loop through products, you can use `ng-container` to avoid adding an unnecessary `div` around each `app-product` element.

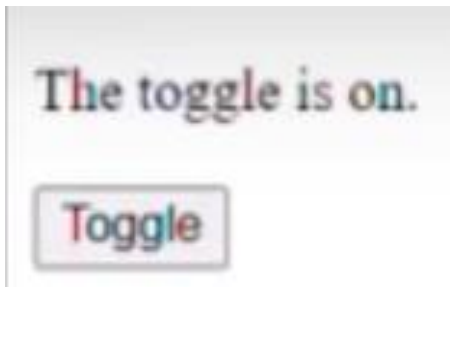
Benefits of `ng-container`:

- It keeps the DOM structure clean by avoiding redundant elements.
- It allows the use of multiple structural directives without violating Angular's restriction of using more than one structural directive on the same element.

```ts
export class AppComponent {
  title = 'angular-ng-container';

  toggle: Boolean = true;

  onToggle(){
    this.toggle = !this.toggle
  }
}
```

```
Go to component
<ng-container *ngIf="toggle">
  <p>The toggle is on.</p>
</ng-container>
<button (click)="onToggle()">Toggle</button>
```





app.component.html

src > app > app.component.html > ng-container

```
Go to component
<h2>Learn NG Template</h2>

<ng-template #myTemplate>
  <h3>This is a template</h3>
  <p>This is an example paragraph to understand ng-template</p>
</ng-template>

<!--ngTemplateOutlet Directive-->
<ng-container *ngTemplateOutlet="myTemplate"></ng-container>
```

app.component.html × | app.component.ts

src > app > app.component.html > ng-container

Go to component

```
1  <ng-container *ngIf="toggle; else toggleOff">
2    <p>The toggle is on.</p>
3  </ng-container>
4  <ng-template #toggleOff>
5    <p>The toggle is off.</p>
6  </ng-template>
7  <button (click)="onToggle()">Toggle</button>
```

The toggle is off.

Toggle

<ng-content> in Angular

Notes on `ng-content` in Angular

Definition:

`ng-content` is used in Angular for **content projection**, which allows the content from a parent component to be inserted into the view template of a child component. This is useful when a component's view depends on dynamic content provided by its parent.

Key Concept:

By default, any content placed inside the opening and closing tags of a component selector is **not rendered**. Using `ng-content`, we can specify places in the child component's template where this content should appear.

Example 1: Basic Content Projection

1. **Parent Component Template (e.g., `app.component.html`):**

```
<app-featured-brands>
  <h3>New Arrivals in Nike</h3>
  <button>Show new arrivals in Nike</button>
</app-featured-brands>
```

2. **Child Component Template (e.g., `featured-brands.component.html`):**

```html
<div>
  <ng-content></ng-content> <!-- Content from parent will be projected here -->
</div>
```

Output:

```html
<div>
  <h3>New Arrivals in Nike</h3>
  <button>Show new arrivals in Nike</button>
</div>
```

- The `h3` and `button` defined in the parent are rendered within the `div` in the child component.

Example 2: Multiple `ng-content` with `select` Attribute

You can project different parts of content into different parts of the child component's template using the `select` attribute. This is useful when you need to control where specific content should appear.

1. **Parent Component Template:**

```
<app-featured-brands>
  <h3 class="brand-heading">New Arrivals in Nike</h3>
  <button class="brand-button">Show new arrivals in Nike</button>
</app-featured-brands>
```

2. Child Component Template:

```
<div>
  <!-- Projecting the heading -->
  <ng-content select=".brand-heading"></ng-content>
  <p>Discover the latest products!</p>
  <!-- Projecting the button -->
  <ng-content select=".brand-button"></ng-content>
</div>
```

Output:

html

```
<div>
  <h3>New Arrivals in Nike</h3>
  <p>Discover the latest products!</p>
  <button>Show new arrivals in Nike</button>
</div>
```

- The `h3` with class `brand-heading` is projected at the first `ng-content`, and the `button` with class `brand-button` is projected after the paragraph.

Content Projection Process:

1. **Step 1: Default Behavior**:
   - Any content between component tags (like `<app-featured-brands>...</app-featured-brands>`) is ignored unless `ng-content` is used in the child component template.
2. **Step 2: Basic Content Projection**:
   - Using `<ng-content></ng-content>` in the child template will render whatever is placed inside the parent component tags.

3. **Step 3: Selective Content Projection**:
   - You can use the `select` attribute to specify which content to project into specific `ng-content` placeholders, allowing for more complex and flexible designs.

Summary:

- `ng-content` enables **content projection**, which allows child components to display content passed from the parent.
- **Selective content projection** can be achieved using the `select` attribute, making it easy to control where different parts of the parent content appear in the child's view.

This is a powerful feature in Angular for creating reusable and flexible components!

`ng-template`

Definition:

- `ng-template` is a directive that defines a template that can be rendered later, either conditionally or repeatedly.

Usage:

- It is used for creating reusable HTML chunks that can be inserted into the DOM dynamically using Angular's structural directives like `*ngIf`, `*ngFor`, etc.

`ng-content`

Definition:

- `ng-content` is used for content projection, allowing you to insert HTML from a parent component into a child component.

Usage:

- It enables you to define placeholder elements in a component that can be filled with content provided by the parent component.

Summary

- Use `ng-template` when you need to create reusable templates for dynamic rendering.
- Use `ng-content` when you want to project content from a parent to a child component, allowing for more flexible component designs.

@ContentChild() in Angular

The @ContentChild decorator is used to access a refence of a DOM element or a component or directive from the projected content into child component class.

Content Projection

parent.component.html

```
<h2>Parent Component</h2>
<app-child>
    <h2>Some Heading</h2>
    <p #paragraph>This is a paragraph</p>
    <app-test></app-test>
</app-child>
```

child.component.html

```
<h2>Parent Component</h2>
<ng-content></ng-content>
```

@ContentChild

parent.component.html

```
<h2>Parent Component</h2>
<app-child>
    <h2>Some Heading</h2>
    <p #paragraph>This is a paragraph</p>
    <app-test></app-test>
</app-child>
```

child.component.ts

```
@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  title = 'Angular App';

  @ContentChild('paragraph') paraEl: ElementRef
}
```

Content Child Decorator in Angular

Definition: The `@ContentChild` decorator is used to access a reference to a DOM element, component, or directive from the **projected content** within a child component. The reference can be accessed inside the child component's class.

Usage:

- Typically used when content is projected from a parent component into a child component via the `<ng-content>` tag.
- Allows the child component to manipulate or access the projected content.

Example:

html

```
<!-- Parent Component Template -->
<app-child>
  <p #paragraphRef>Projected paragraph content</p>
</app-child>
```

html

```
<!-- Child Component Template -->
<div>
  <ng-content></ng-content>
</div>
```

typescript

```
// Child Component Class
import { Component, ContentChild, ElementRef } from '@angular/core';

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
})
```

```
export class ChildComponent {
  @ContentChild('paragraphRef') paragraphEl: ElementRef;

  ngAfterContentInit() {
    console.log(this.paragraphEl.nativeElement); // Access the projected paragraph element
  }
}
```

Content Projection

Definition: Content projection refers to passing content from a parent component to a child component, where the child component defines where the content should be inserted using the `<ng-content>` directive.

Example:

html

```
<!-- Parent Component -->
<app-child>
  <h3>Projected Heading</h3>
  <p>Projected Paragraph</p>
</app-child>
```

html

```
<!-- Child Component Template -->
<div>
  <h2>Child Component Title</h2>
  <ng-content></ng-content> <!-- Projected content will be inserted here -->
</div>
```

View Child Decorator vs Content Child Decorator

- **ViewChild** is used to access elements or components **within the same component's template**.
- **ContentChild** is used to access **projected content** from the parent component into the child component.

Example of ViewChild:

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

@Component({
  selector: 'app-example',
  template: `<button #btn>Click Me</button>`,
})
```

```
export class ExampleComponent {
  @ViewChild('btn') button: ElementRef;

  ngAfterViewInit() {
    console.log(this.button.nativeElement); // Access button element in the component's own
  }
}
```

Differences Between ViewChild and ContentChild

1. **ViewChild** is used to access template elements that belong to the **same component**.
2. **ContentChild** is used to access elements that are **projected** into the child component from the parent.

Example:

- Use **ViewChild** to access elements from the current component's template.
- Use **ContentChild** to access elements that are passed into the child component from the parent using content projection.

child.component.ts | parent.component.html | child.component.html

src > app > parent > child > child.component.ts > ChildComponent > StyleParagraph

```ts
import { Component, ContentChild, ElementRef } from '@angular/core';

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  @ContentChild('para') paragraphEl: ElementRef;

  StyleParagraph(){
    console.log(this.paragraphEl.nativeElement);
  }
```

child.component.ts | parent.component.html | child.component.html

src > app > parent > parent.component.html > div > app-child > p

Go to component

```html
<div>
    <app-child>
        <h3>Some Heading</h3>
        <p #para>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
            incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
            exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
        </p>
    </app-child>
</div>
```

child.component.ts | parent.component.html | parent.component.ts

src > app > parent > parent.component.ts > ParentComponent > paraEl

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

@Component({
  selector: 'app-parent',
  templateUrl: './parent.component.html',
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {
  @ViewChild('para') paraEl: ElementRef;

  showParaValue(){
    console.log(this.paraEl);
  }
```

@ContentChildren() in Angular

The @ContentChildren decorator is used to access a refence of all the DOM elements , components or directives from the projected content in the child component class based on a given selector.

1. `@ContentChildren` Decorator - Definition

The `@ContentChildren` decorator is used to access references of **multiple elements**, components, or directives from **projected content** within a child component. It returns a **query list** containing all elements that match a specified selector.

2. Basic Example: Using `@ContentChildren`

Parent Component HTML (`parent.component.html`)

```html
<app-child>
  <p #para>First paragraph</p>
  <p #para>Second paragraph</p>
  <p #para>Third paragraph</p>
</app-child>
```

Child Component HTML (`child.component.html`)

```html
<ng-content></ng-content>
<button (click)="logParagraphs()">Log Paragraphs</button>
```

Child Component TypeScript (`child.component.ts`)

ts

```ts
import { Component, ContentChildren, QueryList, ElementRef } from '@angular/core';

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html'
})
export class ChildComponent {
  @ContentChildren('para') paraElements!: QueryList<ElementRef>;

  logParagraphs() {
    this.paraElements.forEach((el) => {
      console.log(el.nativeElement);
    });
  }
}
```

3. Explanation and Output

- In the **parent component**, three `<p>` elements are projected as content to the **child component**.
- Using `@ContentChildren('para')`, the **child component** accesses all `<p>` elements and logs them.

Output:

When clicking the "Log Paragraphs" button, the following is logged in the console:

plaintext

```
<p>First paragraph</p>
<p>Second paragraph</p>
<p>Third paragraph</p>
```

4. Difference Between `@ContentChild` and `@ContentChildren`

- `@ContentChild` retrieves **only the first** matching element/component from the projected content.
- `@ContentChildren` retrieves **all matching elements/components** as a `QueryList`.

Example of `@ContentChild`:

ts

```ts
@ContentChild('para') paragraphElement!: ElementRef;
```

Only the **first** `<p>` element will be accessed.

Example of `@ContentChildren`:

ts

```ts
@ContentChildren('para') paragraphElements!: QueryList<ElementRef>;
```

All `<p>` elements will be accessed.

5. Using `@ContentChildren` with Components

You can also use `@ContentChildren` to access instances of a specific component within the projected content.

Example:

In the parent component:

```
<app-child>
  <app-test [name]="'Component 1'"></app-test>
  <app-test [name]="'Component 2'"></app-test>
</app-child>
```

In the child component:

ts

```ts
@ContentChildren(TestComponent) testComponents!: QueryList<TestComponent>;

logTestComponents() {
  this.testComponents.forEach((comp) => {
    console.log(comp.name);
  });
```

Output:

When clicking a button to log the components, the names of the components are printed:

plaintext

```
Component 1
Component 2
```

6. Summary

- `@ContentChildren` is used for **multiple projected elements/components**.
- It returns a **QueryList** of all matched items, allowing you to loop through or operate on them.
- Use `@ContentChild` for accessing **a single** projected element, and `@ContentChildren` for **multiple**.

child.component.ts | parent.component.html ● | child.component.html

app > parent > parent.component.html > div > app-child > p

Go to component

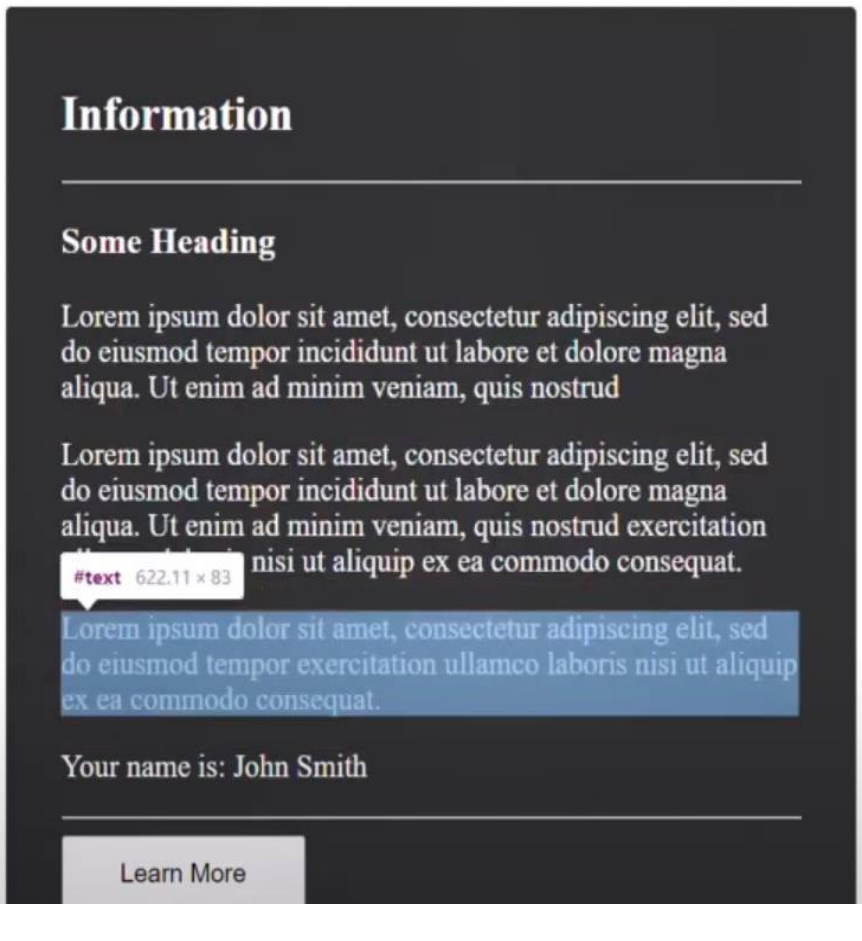
```
<div>
    <app-child>
        <h3>Some Heading</h3>
        <p #para>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
            incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
            exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
        </p>
        <p>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
            incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
        </p>
        <p>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
            exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
        </p>
        <app-test></app-test>
    </app-child>
    <!-- <button (click)="showParaValue()">Show Value</button> -->
</div>
```

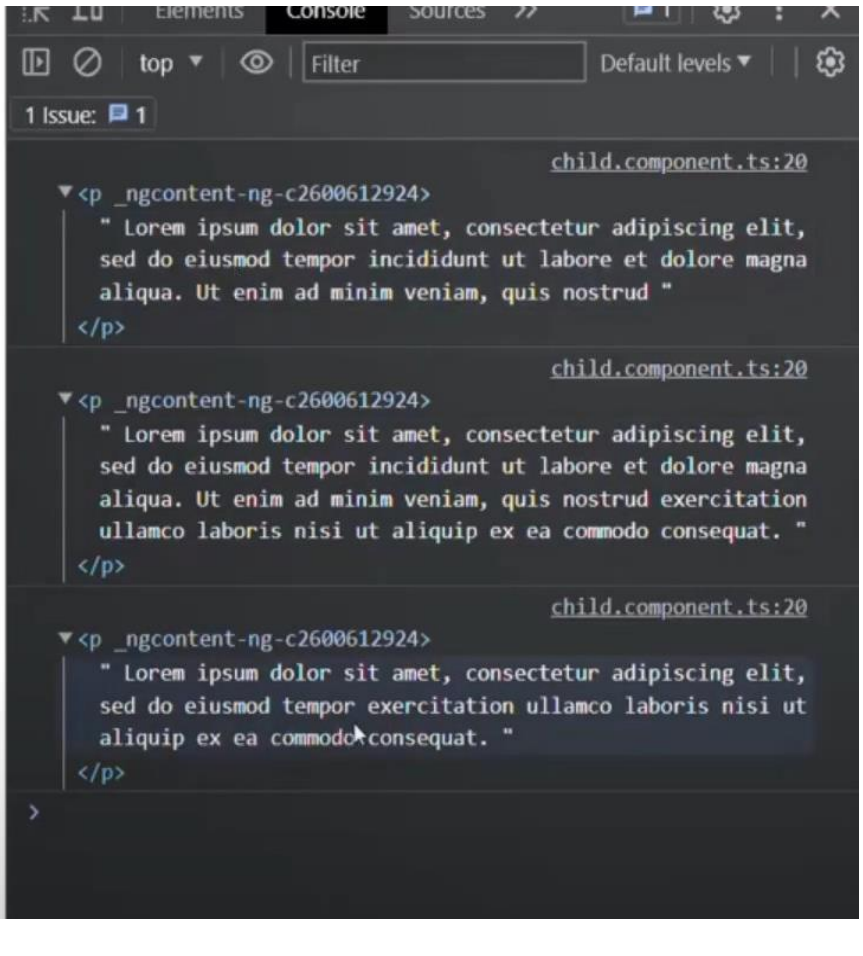
TS child.component.ts | parent.component.html | child.component.html

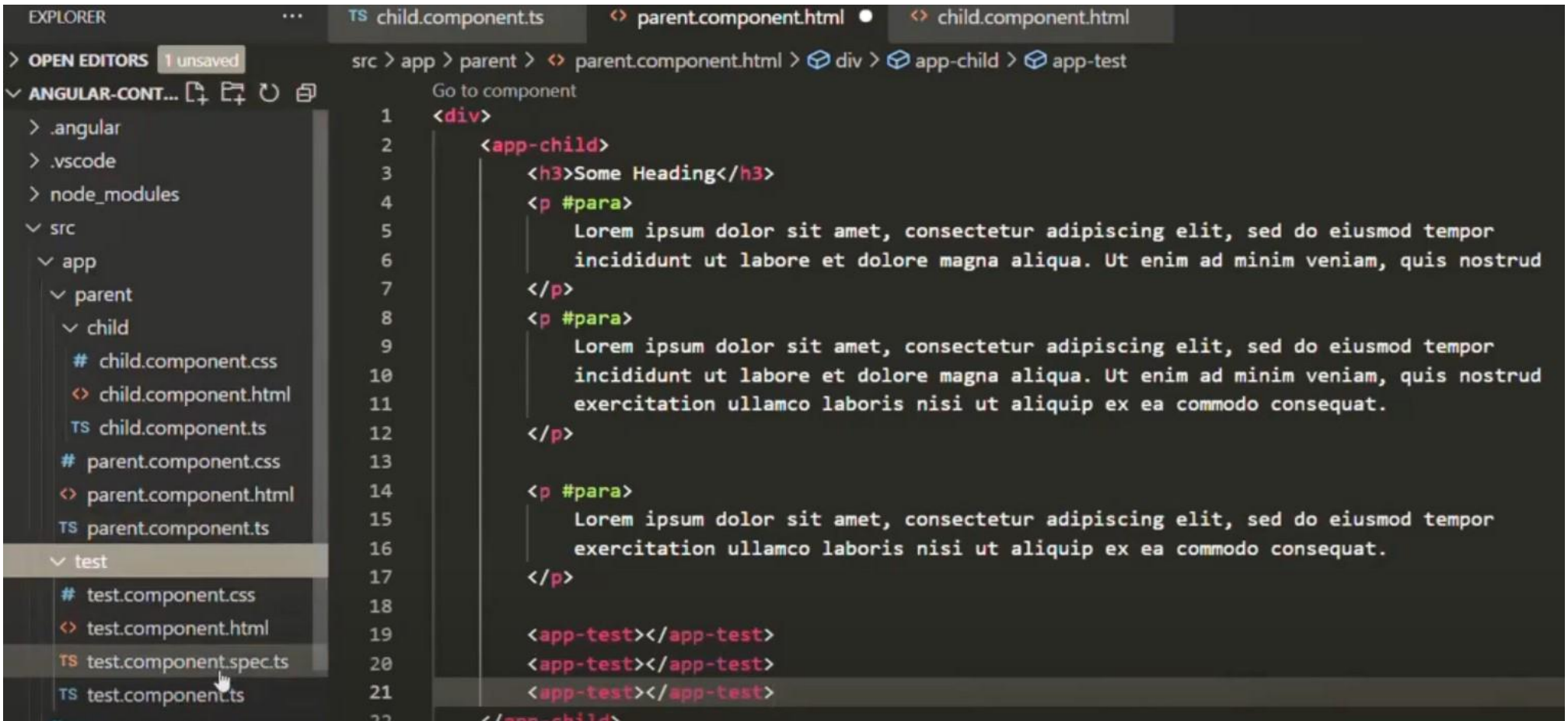
src > app > parent > child > TS child.component.ts > ChildComponent > StyleParagraph

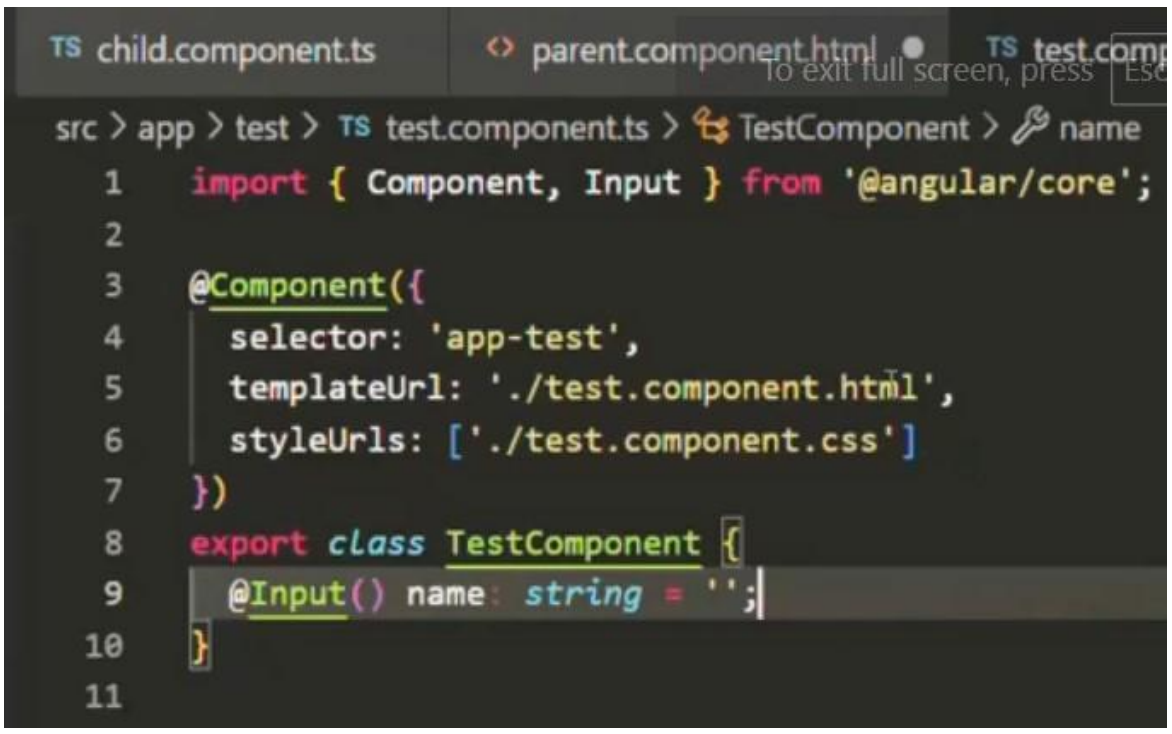
```
import { Component, ContentChild, ElementRef, ContentChildren, QueryList } from '@angular/core';
import { TestComponent } from 'src/app/test/test.component';

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  @ContentChild('para') paragraphEl: ElementRef;

  @ContentChild(TestComponent) testEl: TestComponent;

  @ContentChildren('para') paraElements: QueryList<ElementRef>;

  StyleParagraph(){
    console.log(this.paragraphEl.nativeElement);
    console.log(this.testEl.name);

    this.paraElements.forEach((el) => {console.log(el.nativeElement)});
  }
```

Content Children On Component









```
<app-test [name]="'Merry Jane'"></app-test>
<app-test [name]="'Mark Vought'"></app-test>
<app-test [name]="'John Smit'"></app-test>
</app-child>
```

child.component.ts | parent.component.html | test.component.ts | child.component.html

src > app > parent > child > child.component.ts > ChildComponent > paraElements

```ts
import { Component, ContentChild, ElementRef, ContentChildren, QueryList } from '@angular/core';
import { TestComponent } from 'src/app/test/test.component';

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  @ContentChild('para') paragraphEl: ElementRef;

  @ContentChild(TestComponent) testEl: TestComponent;

  @ContentChildren('para') paraElements: QueryList<ElementRef>;

  @ContentChildren(TestComponent) testElements: QueryList<TestComponent>;

  StyleParagraph(){
    // console.log(this.paragraphEl.nativeElement);
    // console.log(this.testEl.name);

    //this.paraElements.forEach((el) => {console.log(el.nativeElement)});

    this.testElements.forEach((el) => {console.log(el)});
  }
```

@ViewChild

demo.component.html

```html
<h2>Parent Component</h2>
<div>
    <h2>Some Heading</h2>
    <p #paragraph>This is a paragraph</p>
    <app-test></app-test>
</div>
```

demo.component.ts

```ts
@Component({
  selector: 'app-demo',
  templateUrl: './demo.component.html',
  styleUrls: ['./demo.component.css']
})
export class DemoComponent {
  title = 'Angular App';

  @ViewChild('paragraph') paraEl: ElementRef
}
```

@ContentChild

parent.component.html

```
<h2>Parent Component</h2>
<app-child>
    <h2>Some Heading</h2>
    <p #paragraph>This is a paragraph</p>
    <app-test></app-test>
</app-child>
```

child.component.ts

```
@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  title = 'Angular App';

  @ContentChild('paragraph') paraEl: ElementRef
}
```