**Content Projection**

**Single-slot Projection:** Single-slot projection is a basic form of content projection, It can be achieved through <ng-content>, in the template of component, add <ng-content> element where you want to project your content.

Note: <ng-content> element is placeholder, it does not create real DOM element. Custom attributes applied to <ng-content> is ignored.

**Multi-slot Projection:** Using select attribute of <ng-content>, a component has multiple slots, selector attribute defines which content goes into slot.

<ng-content select=’[question]’></ng-content>

In consumer component place the content between component tag.

Different type of selector with the ‘select’ attribute,

1.Select by name

2.Select by class name

3.Select by attribute

4.Select by attribute with value

5.Select by Multiple condition

6.Select by Direct child

**Conditional-slot Projection:** If the component wants to render the content conditionally or render content multiple times, you should configure that component to accept an <*ng-template*> element that contains the content to be rendered.

In this case <*ng-content*> is not recommended, consumer of component supplies content, always initialized, even if component does not define the <*ng-content*>.

In the component that accept the <*ng-template*> element, use an <ng-container> element to render that template using *ngTemplateOutlet* directive.

First create a directive that injects the template reference in constructor.

Then given that directive as attribute to <*ng-template*> element.

Create property with *@contentchild* decorator of type directive created above in the component that accept <*ng-template*> and give that property in <*ng-container*> element attribute [*ngTemplateOutlet*].

**Angular Developer Guide:**

***Standalone:*** it is a property in @component decorator, to tell application that this standalone component will declare its dependencies by its own instead of getting dependencies from ngModule and it can import another standalone component as dependencies using import array.

***Using ngModlue in Standalone component***:

If you want to access other components, directives or pipe some of these dependencies are not standalone component, but you can use it by import ngModule on standalone component.

***Using Standalone component in ngModule:***

You can import standalone component into ngModule by adding that in import array of ngModule.

***Bootstraping Angular Application:***

You can bootstrap an angular application using standalone component as a root component, by using function bootstrapApplication API we can achieve that.

***Configuring dependency Application***: To Bootstrap the application, we need to configure Angular’s Dependency Injection and provide configuration values and service to use throughout the application by **providers**

**ANGULAR APIs:**

***@angular/core***

***ngModule: Decorator***

*Ng-template:* ***Element****/****DIRECTIVE:*** It is an element/directive, that can refer a template in component, that will only render its content when any direct or indirectly instruct to do it.

It provides placeholder for content that can instantiate and rendered conditionally by other directives like ‘ngIf’, ‘ngFor’, ‘ngTemplateOutlet’.

*Ng-Conten****: Element/Directive****:* It is an element/directive, that is used to project the content in an particular place in template. It has ***select*** attribute to select particular selector from the template to show the content of it.

***@angular/forms***

*ControlValueAccessor*: It is an interface, acts like a bridge between angular forms API and native DOM element.

It has three method “writeValue”, “registerOnChange” and “registerOnTouch”

* writeValue set the value from parent component.
* registerOnChange is used to detect the changes in native DOM element to parent component.
* registerOnTouch is used to called when corresponding element get touched, mainly used for validation purpose.
* We should add *providers* and *implement* controlValueAccessor in the corresponding component, to above points to work correctly.

*When to use*

* When we use custom controls as component and that need to interact on angular forms at that time we need this controlValueAccessor
* Every native DOM element have their defaultValueAccessor, their value has been updated by formControl directives.

**Component Interaction:**

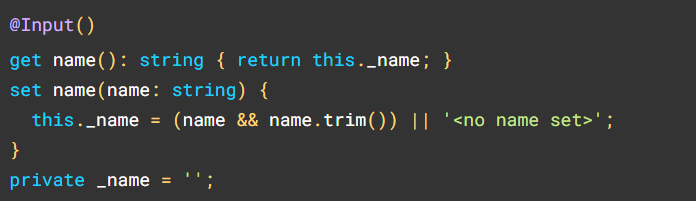
*In this section, we see different way interaction between components*

1.**Pass data from parent to child component via input bindings**

Property decorated with @input() is an input property usually placed in child component, it can be bound its value by parent component.

2.**Intercept input property changes with setter**

Use an input property setter to intercept and act upon a value from a parent.

****

**3.Intercept input property changes with ngOnchanges**

Detect and act upon changes to input property values with *ngOnChanges()* method of *OnChange* lifecycle hook interface.

Mostly prefer this method than property setter method (previous method), because by this method we can watch, handle, and interacting with multiple input property.

**4.Parent listens for child events**

Child component have *EventEmitter* property with which it emits event when something happens.

Parent binds to that property and react to those events. Child’s *EventEmitter* property is an output property, typically decorate with @Output() decorator.

**5.Parent Interact with child using *local* variable**

Parent component cannot use data binding to read child properties or invoke child methods. Do both by creating a *template reference* variable for the child element and reference that variable within parent template.

**6.Parent calls an @Viewchild**

In local variable, parent child relationship of component is not established within each component’s respective class, because the class instance is not connected to one another, the parent class cannot access the child class properties and method.

**7.Parent child communication using service**

Parent and child component share a same service whose interface enables bidirectional communication within family. Scope of service instance in parent component and its children.

**Angular Component Lifecycle**:

***ngOnchanges***: Respond when angular set data bound input property changed. And this method receives ***simplechange*** object which has **currentValue** and **previousValue** property. This life cycle hook firstly called when component or directive .

***ngOnInit***: Called once, after first ngOnChanges(). Initialized angular component or directive after display data bound properties and set component or directive input propertires.

***ngDoCheck***: Called every time once change detection is run on every changes

***ngAfterContentInit***: Called when external content into the component’s view. Called once after ngDoCheck.

***ngAfterContentChanged***: Called when any change detection in content projected in angular component. Called every subsequent ngDoCheck().

***ngAfterViewInit***: Called after angular initialize the component’s view or its child views, or the view that contains directives.

***ngAfterViewChanged***: Called after angular checks the component’s view or its child view or view that contains directives. Called on every subsequent ngAfterContenChanged

***ngDestroy***: Cleanup just before angular destroy the directive or component. Unsubscribe observables and detach even handles to avoid memory leaks.

Angular/core

Classes:

* ChangeDetectionRef
* ComponentRef
* DestoryRef
* ElementRef
* EmbeddedViewRef
* EventEmitter
* Injector
* NgModuleRef
* Query and QueryList
* SimpleChanges
* TemplateRef
* ViewContainerRef
* ViewRef

Angular/Core

Decorators

* Attribute
* Component
* ContentChild
* ContentChildren
* Directive
* Host
* HostBinding
* HostListener
* Inject
* Injectable
* Input
* NgModule
* Output
* Pipe
* ViewChild
* ViewChildren