**Component to Component Communication**

**Introduction**

Component-to-component communication is a key concept in Angular applications. As your application grows, it's essential to understand how to share data between components effectively. In Angular 18 (or any version), there are several strategies for achieving this. This tutorial will guide you through the most common methods:

1. **Parent to Child Communication (Input Binding)**
2. **Child to Parent Communication (Output Binding)**
3. **Sibling Communication (Service with a Subject)**
4. **Using @ViewChild and @ContentChild**

**1. Parent to Child Communication (Input Binding)**

**Step 1: Create a Parent and Child Component**

First, create a parent and child component. For this example, let's name them ParentComponent and ChildComponent.

ng generate component parent

ng generate component child

**Step 2: Use @Input Decorator**

In your ChildComponent, you'll define an input property using the @Input decorator.

**child.component.ts**

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

@Component({

selector: 'app-child',

template: `<p>{{ message }}</p>`,

})

export class ChildComponent {

@Input() message: string = '';

}

**Step 3: Pass Data from Parent to Child**

In your ParentComponent, you can pass data to the ChildComponent by binding to the input property.

**parent.component.html**

<app-child [message]="'Hello from Parent!'"></app-child>

This will display "Hello from Parent!" in the ChildComponent.

**2. Child to Parent Communication (Output Binding)**

**Step 1: Use @Output and EventEmitter**

In your ChildComponent, define an output property using the @Output decorator and an EventEmitter.

**child.component.ts**

import { Component, Output, EventEmitter } from '@angular/core';

@Component({

selector: 'app-child',

template: `<button (click)="sendMessage()">Send Message to Parent</button>`,

})

export class ChildComponent {

@Output() messageEvent = new EventEmitter<string>();

sendMessage() {

this.messageEvent.emit('Hello from Child!');

}

}

**Step 2: Listen for Events in the Parent Component**

In your ParentComponent, listen for the event and handle it accordingly.

**parent.component.html**

<app-child (messageEvent)="receiveMessage($event)"></app-child>

<p>{{ receivedMessage }}</p>

**parent.component.ts**

import { Component } from '@angular/core';

@Component({

selector: 'app-parent',

templateUrl: './parent.component.html',

})

export class ParentComponent {

receivedMessage: string = '';

receiveMessage(message: string) {

this.receivedMessage = message;

}

}

Now, when you click the button in the ChildComponent, the ParentComponent will display "Hello from Child!".

**3. Sibling Communication (Service with a Subject)**

**Step 1: Create a Service**

Create a service to share data between sibling components.

ng generate service data

**data.service.ts**

import { Injectable } from '@angular/core';

import { Subject } from 'rxjs';

@Injectable({

providedIn: 'root',

})

export class DataService {

private messageSource = new Subject<string>();

currentMessage = this.messageSource.asObservable();

changeMessage(message: string) {

this.messageSource.next(message);

}

}

**Step 2: Inject the Service in Sibling Components**

Inject the service into both sibling components and use it to communicate between them.

**sibling1.component.ts**

import { Component } from '@angular/core';

import { DataService } from '../data.service';

@Component({

selector: 'app-sibling1',

template: `<button (click)="sendMessage()">Send Message</button>`,

})

export class Sibling1Component {

constructor(private dataService: DataService) {}

sendMessage() {

this.dataService.changeMessage('Hello from Sibling 1');

}

}

**sibling2.component.ts**

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

import { DataService } from '../data.service';

@Component({

selector: 'app-sibling2',

template: `<p>{{ message }}</p>`,

})

export class Sibling2Component implements OnInit {

message: string = '';

constructor(private dataService: DataService) {}

ngOnInit(): void {

this.dataService.currentMessage.subscribe(message => this.message = message);

}

}

When Sibling1Component sends a message, Sibling2Component will receive it and display it.

**4. Using @ViewChild and @ContentChild**

**@ViewChild**

@ViewChild allows you to access a child component or DOM element from the parent component.

**parent.component.html**

<app-child #childComp></app-child>

<button (click)="accessChild()">Access Child</button>

**parent.component.ts**

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

import { ChildComponent } from '../child/child.component';

@Component({

selector: 'app-parent',

templateUrl: './parent.component.html',

})

export class ParentComponent {

@ViewChild('childComp') child!: ChildComponent;

accessChild() {

alert(this.child.message);

}

}

**@ContentChild**

@ContentChild is used to query projected content within a component.

**parent.component.html**

<app-child>

<p #projectedContent>Projected content</p>

</app-child>

**child.component.ts**

import { Component, ContentChild, AfterContentInit } from '@angular/core';

@Component({

selector: 'app-child',

template: `<ng-content></ng-content>`,

})

export class ChildComponent implements AfterContentInit {

@ContentChild('projectedContent') content!: ElementRef;

ngAfterContentInit() {

console.log(this.content.nativeElement.textContent);

}

}

This logs the text "Projected content" to the console after the content has been initialized.

**Conclusion**

These are the common methods of component-to-component communication in Angular 18:

* Use @Input for parent to child communication.
* Use @Output and EventEmitter for child to parent communication.
* Use a shared service with a Subject for sibling communication.
* Use @ViewChild and @ContentChild to access child components or projected content.

Understanding these patterns will help you build complex, scalable Angular applications.