1. **CÀI ĐẶT VỚI CLI**

Cài đặt **nodeJS**

Check **node –v, npm –v**

Install CLI: (command line interface): **npm install –g @angular/cli**

Check: **ng –v**

Cd/ hoặc vào thư mục cần tạo Project

Tạo Project: **ng new *tênProject***

Start Project: **ng serve -o \\\ để mở luôn trên Brower**

**Các biến trong class là dấu = ko phải :**

1. **COMPONENT**

Component: chứa các HTML, chứa các Component khác

**@Component({**

templateUrl: ‘*path*’, //// chỉ có 1 thôi

Selector:’app-name’

styleUrls: ‘[‘*path’*]’ /// nếu có s thì đây là arr[]

**})**

Tạo = CLI: **ng g component/c *<name>***

Tạo 1 thư mục đó gồm 4 files

Trong App.module đã import và đã thêm vào Declaration

1. **STRING INTERPOLATION**

*Name.*component.ts: export class *Name {*

en= ‘English’;

vn=’Vietnam’;

*}*

*Name.*component.html: <div> **{{ en }}** </div> <div> **{{ vn }}** </div>

*Dùng để*  lấy giá trị của **property/biến trong class** và xuất ra template.

Note: trong cùng 1 Component mới lấy ra dc

1. **PROPERTIES BINDING**

*Name*.component.ts: export class Name {

En= ‘English’;

Vn=’Vietnam’;

Src= ‘google.com’;

isShow= true;

}

1. <img src=”{{ src }}” > (string interpolation) \\\ nhớ phải có dấu nháy kép
2. <img *[src]*=”src” > (giá trị hay biến trong class) \\\ nhớ phải có dấu nháy kép

<div *[hidden]*=”isShow” > {{ en }} </div> \\\ nhớ phải có dấu nháy kép

Đây là cách Class tương tác tới Template

1. **EVENT BINDING**

<button *(click)*=”isShow = !isShow” ></button>

<button *(click)*=”toggle()” ></button>

Thường bind bằng 1 function

*Name*.component.ts: export class Name {

toggle() { \\\ đây gọi là **Handler**

this.isShow = !this.isShow;

}

}

Đây là cách Template tương tác lại Class

1. **TWO-WAY BINDING**

**Import** { FormModule } from ‘@angular/forms’;

Thêm vào trong mảng **import của @NgModule**

Tip: thêm vào trong **import của @NgModule** VS code sẽ gợi ý import trên đầu

<input [(ngModel)] =’name’> \\\\ nghĩa: đã gán biến name vào giá trị của input

**ngModel** = ‘[biến trong class]’

<h3> {{ name }} </h3>

1. **TEMPATE REFERENCE VARIABLE**

Tham chiểu đến các web component, directive

1. **STYLE BINDING AND CLASS BINDING**

<h3 [style.*color*]=” ‘red’ ” > {{ name }} </h3>

<h3 [class]=” ‘square’ ” > {{ name }} </h3>

<h3 [class.square]=” true/false ” > {{ name }} </h3>

1. **DIRECTIVE**

Đưa ra các chỉ thị/chỉ dẫn để angular chuyển đổi template thành DOM

1. Component => Directive với 1 template
2. Attribute directive => thay đổi hành vi hiển thị hoặc hiển thị của các DOM, component hoặc directive khác
3. Structural Directive => thay đổi/ thêm bớt DOM
4. **Attribute directive**
5. **ngStyle (như là 1 tập hợp các style props)**

<h3 **[ngStyle]=”{ color: ‘red’, fontSize: ‘20px’ }”** > {{ name }} </h3> (có thể thay = Obj)

1. **ngClass (như là 1 tập hợp các Class)**

<h3 **[ngClass]=”{ square, border}”** > {{ name }} </h3> (có thể thay = Obj) \\\ ko có dấu nháy tên class

1. **Structural Directive**

Name.component.ts: export class Name {

arr = [1, 2, 3, 4, 6, 7, 8]

}

1. **\*ngIf:**

<h3 *\*****ngIf***=”true/false”> {{ name }} </h3>

1. **\*ngFor:** dùng vòng lặp For .... of để lặp các item trong arr

<ul>

<li *\*****ngFor****=”let item* ***of*** *arr ”*>

<p> {{ *item* }} </p>

</li>

</ul>

1. **Điều kiện khi filter**

Sẽ có 3 trường hợp: all, đã nhớ, chưa nhớ

Thì viết điều kiện cho từng trường hợp đó, dựa trên 1 prop của 1 obj có kiểu boolean

1. **INPUT (như prop trong React từ Cha > Con)**

ComCon:

Import {input} from @angluar/core

Export class{

@Input() *input1:* [type];

@Input() *input2:* [type];

construtor(){

}

}

ComCha-template: <div [*input1*]=”name” [*input2*]=”age” ></div>

**Kết hợp \*ngFor + Input**

<div \*ngFor=”let item of arr”>

<div [*input1*]=”item.name” [*input2*]=”item.age” ></div>

</div>

1. **OUTPUT (truyền 1 event Con > Cha)**
2. ComCha: truyền vào ComCon 1 event tự đặt tên (nameEvent) =”function()”.  
   \\ **function() này ở trong ComCha**
3. ComCon: lấy được Event:
   1. Import {Output, EventEmitter} from @angular/core
   2. Trong class : @Output() *nameEvent* = new EventEmitter();  
       \\\ *lúc này ComCon đã hứng dc event từ Cha* **this.nameEvent**
   3. Template: (click)=”func()”
   4. Define func() {

This.*nameEvent*.emit();

}

1. **OUTPUT CÓ THAM SỐ ($event )**

1. ComCha: truyền vào ComCon 1 event tự đặt tên (nameEvent) =”function(**$event**)”.  
\\ **function() này ở trong ComCha**

2. ComCon: lấy được Event:

1. Import {Output, EventEmitter} from @angular/core
2. Trong class : @Output() *nameEvent* = new **EventEmitter<type>();**  
   \\\ typescript yêu cầu nhập kiểu dữ liệu nữa   
    \\\ *lúc này ComCon đã hứng dc event từ Cha* **this.nameEvent**
3. Template: (click)=”func()”
4. Define func1() {

This.*nameEvent*.emit(**biến**);

}  
\\ khi emit thì cũng truyền giá trị theo lun

1. Define func2() {

This.*nameEvent*.emit(**biến**);

}  
\\ khi emit thì cũng truyền giá trị theo lun

1. **VIEWCHILD**

**Cách 1 – ko xài Viewchild**

Trường hợp là 1 btn trong ComCha muốn tương tác 1 element trong ComCon

Trong ComCha:Tạo 1 template variable: #child   
\\như vậy #child dc attach vào app-child và nó là ChildComponent => có được các biến và hàm trong Com đó.

<button *(click)*=“child.value”>++</button>  
<app-child **#child**></app-child>

**Cách 2 – xài ViewChild**

**ComCha:**

**Import {ComCon} ...**

**Import {**viewchild**} from a**ngular/core

Class{

@ViewChild(ComCon);

*tenBien*: ComCon;

Define func(){

This. *tenBien.*value++   
 \\\\\ vì *tenBien là ConCom* => có được các biến và hàm trong Com đó.

}

}

1. **ngContent (giống như prop children trong React)**

thay đổi linh hoạt cái element trong template: <p>,Component,<h2>,...

1. **Built-in PIPES**

Template: {{ tenBien | date/json/async }}

*Name*.component.ts: export class Name {

tenBien = new Date()

}

Json: chuyển các [object object] thành file string để có thể read

Async: hứng cái data dạng promise từ server. **Promise.resolve**(‘*cái sẽ hiện ra template*’)

1. **CUSTOM PIPES**

**IMPORT {Pipe, PipeTransform}**

**@Pipe({** name: ‘tenPipe’**});**

Export class **tenClass implements PipeT**ransform

**Transform(**value: number**): number {   
\\\\ nghĩa value là** type:number và kiểu trả về là number

**}**

1. **SERVICE**

Tạo 1 service. Import các thứ cần. Và phải có @**Injectable**. Thường nó sẽ có provider: root \\ là global

Khai báo service đầy đủ

Trong Com nào cần **Dependancy Injection(DI)** thì sẽ khởi tạo cái service trong hàm **constructor()**

*Name*.component.ts: export class Name {

tenBien = new Date()

constructor(private *tenBienservice*: **tenService**) {

this.*tenBienservice.*func();

}

}