**Section5: Components & Databinding Deep Drive**

**65) Splitting Apps Into Components**

-create a new component manually(cockpit, server-element)

(name,content,buttons)

ng g c cockpit –spec false(--skip-test) -> test file not getting created

ng g c server-element --skip-test

app.component.html

* First row cut it and add to cockpit

cockpit.component.html

* Paste the code

app.component.ts

cut both methods

Move methods calling OnAddServer & OnAddBlueprint

cockpit.component.html

paste the methods on cockpit component

app.component.ts

move of two properties ,used in cockpit.component.html

newServerName = '';

newServerContent = '';

app.component.html

* second row cut it and add to server-element

server-element.component.html

* Paste the code

app.component.html

<app-cockpit></app-cockpit>

<app-server-element \*ngFor="let serverElement of serverElements"></app-server-element>

**67)Binding to Custom Properties**

Cockpit.component.ts

Comment methods

Server-element.component.ts

Access the single server element , create a property to this file

export class ServerComponent {

element: {type: string, name: string, content: string}

}

app.component.ts

export class AppComponent {

serverElements = [

{type:'server',name:'testserver',content:'just a text'}];

}

app.component.html

bind element property of server-element component

<app-server-element

\*ngFor="let serverElement of serverElements"

[element]="serverElement">

</app-server-element>

server-element.component.ts

Properties of component are only accessiable inside these componenr not from outside

@input Decorator missing I will throw the error in browser, stating element is not a known property

export class ServerComponent{

@Input() element: {type: string, name: string, content: string}

}

**68)Assigning an Alias to custom properties**

Server-element.component.ts

export class ServerElementComponent{

@Input('srvElement') element: {type: string, name: string, content: string}

}

in case if we want pass value to server component need to use 'srvElement'

**69)Binding to Custom Events**

Cockpit.componenr.ts

Copy method

App.component.ts

New server and new blueprint was created,two methods add

OnServerAdded(){},OnBlueprintAdded(){},still create a new server or a new Blueprint however this will not work as expected here b/c referering newserverName & newServerContent which not available in the app component

Export class AppComponent{

serverElement =[{type:’server’, name:’testserver’,content:’just a test’}]

OnSeverAdded(){

This.serverElement.push({

Type:’server’,

Name:this.newserverName,

Content:this.newServerContent

});

}

OnBlueprintAdded(){

This.serverElement.push({

Type:’Blueprint’,

Name:this.newserverName,

Content:this.newServerContent

});

}

App.component.html

<app-cockpit (serverCreated)='OnServerAdded($event)'

(blueprintCreated)='OnblueprintAdded($event)'

></app-cockpit>

App.component.ts

export class AppComponent {

  serverElements = [{type:'server',name:'testserver',content:'just a text'}];

  onServerAdded(serverData:{serverName:string,serverContent:string}) {

    this.serverElements.push({

      type: 'server',

      name: serverData.serverName,

      content: serverData.serverContent,

    });

  }

  onBlueprintAdded(blueprintData:{serverName:string,serverContent:string}) {

    this.serverElements.push({

      type: 'blueprint',

      name: blueprintData.serverName,

      content:blueprintData.serverContent

    });

  }

}

Cockpit.component.ts

Create two new properties

serverCreated=new EventEmitter<{serverName:string,serverContent:string}>();

blueprintCreated=new EventEmitter<{serverName:string,serverContent:string}>();

then import EventEmitter from @angular/core

onAddServer() {

        this.serverCreated.emit({

         serverName: this.newServerName,

         serverContent: this.newServerContent

        });

      }

      onAddBlueprint() {

        this.blueprintCreated.emit({

          serverName: this.newServerName,

          serverContent: this.newServerContent

         });

      }

@output display output template

 @Output()serverCreated=new EventEmitter<{serverName:string,serverContent:string}>();

    @Output()blueprintCreated=new EventEmitter<{serverName:string,serverContent:string}>();

**70)Assigning an Alias Custom Events**

Cockpit.component.ts

@output(‘sampleCreated’) serverCreated=new EventEmitter<{serverName:string,serverContent:string}>();

app.component.html

<app-cockpit(‘sampleCreated’)=”onServerAdded($event)”

**73)More on View Encapsulation**

Server-element.component.css

P{

Color:blue;

}

Server-element.component.ts

Add to the Component decorator

Encapsulation:ViewEncapsulation.None

None,Emulated-different content style,native-This is called Shadow Dom ,instead of ‘native’ now the functionality is the same though

**74) Local Reference in Template**

Two way data binding to get the servername &content,a local reference can be placed on any html element

Cockpit.component.html

<input type=”text” class=”form-control” #serverNameInput>

<button class=”btn btn-primary

(click)=”onAddServer(ServerNameInput)”>AddServer</button>

Cockpit.component.ts

onAddServer(nameInput){

console.log(nameInput.value);

local reference-get access to some element in your template and use that either directly in the template(object created or not)

cockpit.component.ts

//newserverName=’’;

onAddServer(nameInput:HTMLInputElement){

this.serverCreated.emit({

serverName:nameInput.value,

serverContent:this.newServerContent

});

} onAddBlueprint(nameInput:HTMLInputElement){

this.blueprintCreated.emit({

serverName:nameInput.value,

serverContent:this.newServerContent

});

}

button class=”btn btn-primary

(click)=”onAddBlueprint(ServerNameInput)”>AddBlueprint

</button>

**76)Getting Access to the Template & Dom with @viewchild**

Viewchild is a view query

Cockpit.component.html

<input type=”text” class=”form-control” #serverContentInput>

cockpit.component.ts

//newserverContent=’’;

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

@viewchild(‘serverContentInput’)ServerContentInput;

onAddServer(nameInput:HTMLElement){

this.serverCreated.emit({

serverName:nameInput.value,

serverContent:this.serverContentInput.native.Element.value

});

}

onAddBlueprint(nameInput:HTMLElement){

this.blueprintCreated.emit({

serverName:nameInput.value,

serverContent:this.serverContentInput.native.Element.value

});

}

**77)projecting content into component with ng-content**

Server-element.component.html to app.component.html

<p>

<strong \*ngIf="element.type === 'server'" style="color: red">{{ element.content }}</strong>

      <em \*ngIf="element.type === 'blueprint'">{{ element.content }}</em>

</p>

Server-element.component.html

<ng-content></ng-content>

**79)Lifecycle Hooks in Action**

Server-element.component.ts

import { Component,OnInit,Input,ViewEncapsulation,OnChanges,SimpleChanges,

DoCheck,AfterContentInit,AfterContentChecked,AfterViewInit,AfterViewChecked,

OnDestroy,ViewChild,ElementRef } from '@angular/core';

export class ServerElementComponent implements OnInit, OnChanges, DoCheck,

  AfterContentInit,AfterContentChecked,AfterViewInit,AfterViewChecked,

  OnDestroy {

@Input('srvElement') element: {type: string, name: string, content: string};

  @Input() name: string;

constructor() {

    console.log('constructor called!');

  }

  ngOnChanges(changes: SimpleChanges) {

    console.log('ngOnChanges called!');

    console.log(changes);

  }

  ngOnInit() {

    console.log('ngOnInit called!');

}

 ngDoCheck() {

    console.log('ngDoCheck called!');

  }

ngAfterContentInit() {

    console.log('ngAfterContentInit called!');

}

ngAfterContentChecked() {

    console.log('ngAfterContentChecked called!');

  }

  ngAfterViewInit() {

    console.log('ngAfterViewInit called!');

}

  ngAfterViewChecked() {

    console.log('ngAfterViewChecked called!');

  }

  ngOnDestroy() {

    console.log('ngOnDestroy called!');

  }

}

App.component.html(demo)(onchage)

<button class=”btn btn-primary” (click)=”onchangeFirst()”>change irst Element

</button>

App.component.ts

onchangeFirst(){}

Server-element.component.html

@Input()name:string;--------declare top

Server-element.component.html

<div class="panel-heading">{{ name }}</div>

app.component.html

<app-server-element

[name]=”serverElement.name”>

App.component.ts

onChangeFirst(){

this.serverElement[0].name=’changed!”;

}

App.component.html(destroy button)

<button class=”btn btn-danger” (click)=”onDestroyFirst()”>Destroy First Element

</button>

App.component.ts

onDestroyFirst(){

this.serverElementr.splice(0,1);

}

**80)Lifecycle Hooks and Template Access**

Server-element.html

<div class="panel-heading" #heading>{{ name }}</div>

Server-element.component.ts

  @ViewChild('heading')header: ElementRef;

  ngOnInit() {

console.log('Text Content: ' + this.header.nativeElement.textContent);

}

  ngAfterViewInit() {

console.log('Text Content: ' + this.header.nativeElement.textContent); }

82)@contentchild

App.component.html

<p

#contentparagraph>

<strong \*ngIf="element.type === 'server'" style="color: red">{{ element.content }}</strong>

      <em \*ngIf="element.type === 'blueprint'">{{ element.content }}</em>

</p>

Server-element.component.ts

@ContentChild('contentParagraph', {static: true}) paragraph: ElementRef;

 ngOnInit() {

console.log('Text Content of paragraph: ' + this.paragraph.nativeElement.textContent);

  }

 ngAfterContentInit() {

    console.log('Text Content of paragraph: ' + this.paragraph.nativeElement.textContent); }