**Angular 5**

**Section 4**

Section 4: Lecture 55 //Understanding Angular Error Messages

No need for notes

Section 4: Lecture 56 //Debugging code in the browser using Source maps

1. Go into sources and web pack to access various files in your angular application and also check their values.
2. You can also debug line by line here.

Section 4: Lecture 57 //Using Augury to dive into Angular Apps

1. Google Angular Augury - it’s a tool for debugging.
2. Once you click on install it will add the Augury extension to your crome browser.

**Section** **5**

Section 5: Lecture 58 //Module Introduction

1. Understanding Components and Databinding – now we will dive deeper into components and data binding.
2. We will understand deeper connection between components and databinding.
3. Then we will modify our app.

Section 5: Lecture 59 //Splitting app into components

1. We need to split the app into components to reuse the components as our app grows bigger and this also focuses on separation of concerns and reusability of various components.
2. Let’s create a new cockpit component:

ng g c cockpit --spec false

1. Let’s create one more component i.e. server element:

ng g c server-element –spec false

1. Add below code to cockpit.component.html

<div class="container">

<div class="row">

<div class="col-xs-12">

<p>Add new Servers or blueprints!</p>

      <label>Server Name</label>

      <input type="text" class="form-control" [(ngModel)]="newServerName">

      <label>Server Content</label>

      <input type="text" class="form-control" [(ngModel)]="newServerContent">

      <br>

      <button class="btn btn-primary"

          (click)="onAddServer()">Add Server</button>

      <button class="btn btn-primary"

          (click)=" onAddBlueprint()">Add Server Blueprint</button>

</div>

</div>

</div>

1. Add below code to cockpit.component.ts
2. import { Component, OnInit } from '@angular/core';
3. @Component({
4. selector: 'app-cockpit',
5. templateUrl: './cockpit.component.html',
6. styleUrls: ['./cockpit.component.css']
7. })
8. export class CockpitComponent implements OnInit {
9. //serverElements = [];
10. newServerName = '';
11. newServerContent = '';
13. constructor() { }
14. ngOnInit() {
15. }
16. onAddServer(){
18. this.serverElements.push({
19. type: 'server',
20. name: this.newServerName,
21. content: this.newServerContent
22. });
23. }
25. onAddBlueprint(){
26. this.serverElements.push({
27. type: 'blueprint',
28. name:this.newServerName,
29. content: this.newServerContent
30. });
31. }
32. }

6. Put below code in server-element.component.html:

<div

class="panel panel-default"

>

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

<div class="panel-body">

<p>

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

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

</p>

</div>

</div>

7. Add below code to app.comonent.ts :

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

@Component({

selector: 'app-root',

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

//styleUrls: ['./app.component.css']

styles: [`

h3{

color: dodgerblue;

}

`]

})

export class AppComponent {

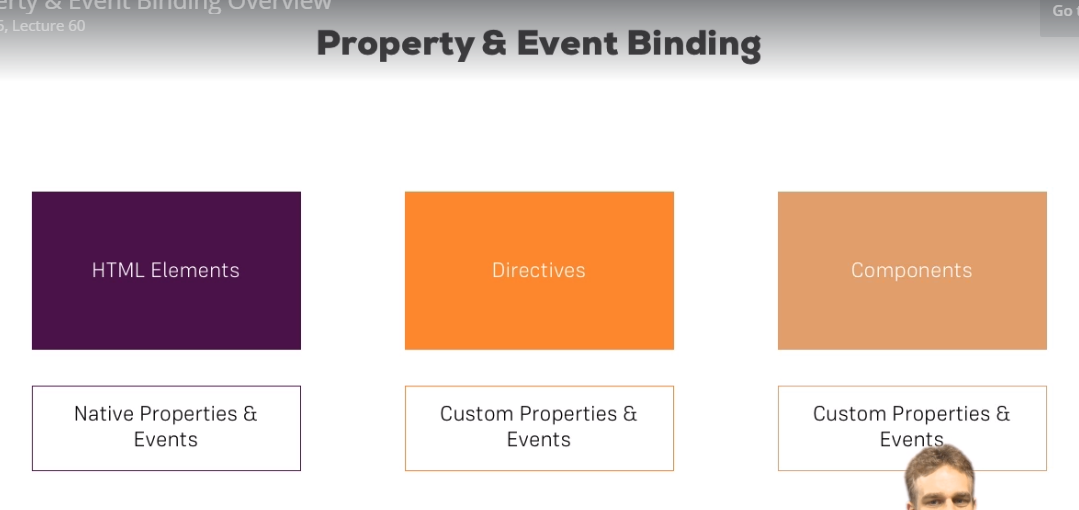
serverElements = [];

name = '';

}

Section 5: Lecture 60 //Property and Event Binding overview

1. In the last lecture we split up our app into components, but one issue is that we need to pass the data into these components.
2. In the event binding, when we click to a button something happens i.e. an event gets emitted to which we are able to listen, and we can use that data with **$event.**
3. Now we need the above behaviour from our components i.e. we need to send data into a component or receive data from the component.
4. We need to send and receive an event, and Angular gives us the great tools to implement this flow.
5. We can use this not only in the HTML elements but also in Directives and finally we can also use this in our own components, as shown in the screenshot below:



1. We can have our own custom properties and custom events.

Section 5: Lecture 61 //Binding to custom properties

1. Now, we have learnt that we can use our own properties to bind the properties, properties of our own components.
2. Add the below code in server-element.component.ts :
3. import { Component, OnInit } from '@angular/core';
4. @Component({
5. selector: 'app-server-element',
6. templateUrl: './server-element.component.html',
7. styleUrls: ['./server-element.component.css']
8. })
9. export class ServerElementComponent implements OnInit {
10. element: {type: string, name: string, content: string};
11. constructor() { }
12. ngOnInit() {
13. }
14. }
15. Add below code in app.component.ts :
16. import { Component } from '@angular/core';
17. @Component({
18. selector: 'app-root',
19. templateUrl: './app.component.html',
20. styleUrls: ['./app.component.css']
21. // styles: [`
22. // h3{
23. // color: dodgerblue;
24. // }
25. // `]
26. })
27. export class AppComponent {
28. serverElements = [{type: 'server', name: 'Testserver', content: 'Just a test!'}];
29. name = '';
30. }

5. Now, to allow the parent component to be able to bind to this property, you need to add a decorator; as we know that the decorators are not only available for the classes.

Below is the decorator we need to add:

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

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

1. So the contents of the server-element.component.ts become as below:
2. import { Component, OnInit, Input } from '@angular/core';
3. @Component({
4. selector: 'app-server-element',
5. templateUrl: './server-element.component.html',
6. styleUrls: ['./server-element.component.css']
7. })
8. export class ServerElementComponent implements OnInit {
9. @Input() element: {type: string, name: string, content: string};
10. constructor() { }
11. ngOnInit() {
12. }
13. }

Section 5: Lecture 62 //Assigning an Alias to Custom Properties

1. You can also give the name in the parenthesis of the Input that you want to use outside i.e. when this property is used by some other component as below:

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

1. Above is the way in which we can assign an alias to the any property of a component.
2. Now the code of server-element.component.ts becomes like below:
3. import { Component, OnInit, Input } from '@angular/core';
4. @Component({
5. selector: 'app-server-element',
6. templateUrl: './server-element.component.html',
7. styleUrls: ['./server-element.component.css']
8. })
9. export class ServerElementComponent implements OnInit {
10. @Input('srvElement') element: {type: string, name: string, content: string};
11. constructor() { }
12. ngOnInit() {
13. }
14. }
15. And the code of app.component.html becomes as below:
16. <div class="container">
17. <app-cockpit></app-cockpit>
18. <hr>
19. <div class="row">
20. <div class="col-md-12">
21. <app-server-element
22. \*ngFor="let serverElement of serverElements"
23. [srvElement]="serverElement"
24. ></app-server-element>
25. </div>
26. </div>
27. </div>

Section 5: Lecture 63 //Binding to the custom events

1. We learnt how to pass information from one component down (i.e. to the child component) to another component.
2. Now, we will learn how to pass the communication from the child component to the parent component.
3. Here we want to inform our parent component i.e. the app.component in this case new server was created, because the code which would have simply executed is commented out.
4. Now in the cockpit.component we want to implement the above methods.
5. Update the code of app.component.ts as below:
6. import { Component } from '@angular/core';
7. @Component({
8. selector: 'app-root',
9. templateUrl: './app.component.html',
10. styleUrls: ['./app.component.css']
11. // styles: [`
12. // h3{
13. // color: dodgerblue;
14. // }
15. // `]
16. })
17. export class AppComponent {
18. serverElements = [{type: 'server', name: 'Testserver', content: 'Just a test!'}];
19. name = '';
20. onServerAdded(serverData: {serverName: string, serverContent: string}){
22. this.serverElements.push({
23. type: 'server',
24. name: serverData.serverName,
25. content: serverData.serverContent
26. });
27. }
29. onBlueprintAdded(bluePrintData: {serverName: string, serverContent: string}){
30. this.serverElements.push({
31. type: 'blueprint',
32. name: bluePrintData.serverName,
33. content: bluePrintData.serverContent
34. });
35. }
36. }

6. Update the code of app component.html as below:

<div class="container">

<app-cockpit

(serverCreated)="onServerAdded($event)"

(bluePrintCreated)="onBluePrintAdded($event)"

></app-cockpit>

<hr>

<div class="row">

<div class="col-md-12">

<app-server-element

\*ngFor="let serverElement of serverElements"

[srvElement]="serverElement"

></app-server-element>

</div>

</div>

</div>

6. Now, we need to make both serverCreated and blueprintCreated as the events that we will emit from cockpit.component.ts, this we will do by assigning the property to new EventEmitter<> ; There are greater than and smaller than sign behind the EventEmitter because the event emitter is a generic type. So here we need to define the type of event we are going to emit.

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

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

7. Now, we need to add the decorator to make the property listenable from outside, below is the code for the same with the decorator:

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

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

8. Contents of cockpit.component.ts will become:

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

@Component({

selector: 'app-cockpit',

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

styleUrls: ['./cockpit.component.css']

})

export class CockpitComponent implements OnInit {

//serverElements = [];

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

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

newServerName = '';

newServerContent = '';

constructor() { }

ngOnInit() {

}

onAddServer(){

this.serverCreated.emit({serverName : this.newServerName,

serverContent: this.newServerContent});

}

onAddBlueprint(){

this.blueprintCreated.emit({serverName : this.newServerName,

serverContent: this.newServerContent});

}

}

9. Contents of app.component.ts will become:

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

@Component({

selector: 'app-root',

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

styleUrls: ['./app.component.css']

// styles: [`

// h3{

// color: dodgerblue;

// }

// `]

})

export class AppComponent {

serverElements = [{type: 'server', name: 'Testserver', content: 'Just a test!'}];

name = '';

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

});

}

}

10. Content of server-element.component.html will become:

<div

class="panel panel-default"

>

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

<div class="panel-body">

<p>

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

<!-- <strong \*ngIf="element.type === 'blueprint'" style="color: blue">{{ element.content }}</strong> -->

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

</p>

</div>

</div>

Section 5: Lecture 64 //Assigning an Alias to custom events

1. Just like @Input(), you can also assign alias to @Output() as shown in the example below:

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

1. App.component.html will get updated as below:
2. <div class="container">
3. <app-cockpit
4. (serverCreated)="onServerAdded($event)"
5. (bpCreated)="onBlueprintAdded($event)"
6. ></app-cockpit>
8. <hr>
9. <div class="row">
10. <div class="col-md-12">
11. <app-server-element
12. \*ngFor="let serverElement of serverElements"
13. [srvElement]="serverElement"
14. ></app-server-element>
15. </div>
16. </div>
17. </div>

Section 5: Lecture 65 //Custom properties and event binding summary

1. We will use services when the distance between the component will grow too much i.e. there are chains of components communicating.

Section 5: Lecture 66 //Understanding view encapsulation

1. Now the properties applied in the app.component.css file will not work for the child components as it clearly belongs to the app.component.html template.
2. The behaviour that the properties defined in the CSS file of a particular component will be applied to that component only – is the behaviour given to us by angular and not default behaviour.
3. This is the default behaviour of the view encapsulation in Angular.

Section 5: Lecture 67 //More on view encapsulation

1. In the last lecture we learnt that how angular encapsulates your styles, now you can overwrite this encapsulation though.
2. If we add the below code in a component, then that component will not use view encapsulation:

encapsulation: ViewEncapsulation.None

1. Now the code of server-element.component.ts becomes as shown below:

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

@Component({

selector: 'app-server-element',

templateUrl: './server-element.component.html',

styleUrls: ['./server-element.component.css'],

encapsulation: ViewEncapsulation.None

})

export class ServerElementComponent implements OnInit {

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

constructor() { }

ngOnInit() {

}

}

1. The default is Emulated which means that only your component will receive the style which you define for that component.

Section 5: Lecture 68 //Using local references in templates

1. We can use the local references only inside our HTML template, but not inside the typescript by using the # key as shown below;
2. <input
3. type="text"
4. class="form-control"
5. #serverNameInput>
6. Now, we can use any local reference anywhere in the app, updated code in cockpit.component.html is shown below:
7. <div class="container">
8. <div class="row">
9. <div class="col-xs-12">
10. <p>Add new Servers or blueprints!</p>
11. <label>Server Name</label>
12. <!-- <input type="text" class="form-control" [(ngModel)]="newServerName"> -->
13. <input
14. type="text"
15. class="form-control"
16. #serverNameInput>
17. <label>Server Content</label>
18. <input type="text" class="form-control" [(ngModel)]="newServerContent">
19. <br>
20. <button class="btn btn-primary"
21. (click)="onAddServer(serverNameInput)">Add Server</button>
22. <button class="btn btn-primary"
23. (click)="onAddBlueprint(serverNameInput)">Add Server Blueprint</button>
24. </div>
25. </div>
26. </div>
27. Updated code of cockpit.component.ts is as below:
28. import { Component, OnInit, Output, EventEmitter } from '@angular/core';
29. @Component({
30. selector: 'app-cockpit',
31. templateUrl: './cockpit.component.html',
32. styleUrls: ['./cockpit.component.css']
33. })
34. export class CockpitComponent implements OnInit {
35. //serverElements = [];
36. @Output() serverCreated = new EventEmitter<{serverName: string, serverContent: string}>();
37. @Output('bpCreated') blueprintCreated = new EventEmitter<{serverName: string, serverContent: string}>();
38. newServerName = '';
39. newServerContent = '';
41. constructor() { }
42. ngOnInit() {
43. }
44. onAddServer(nameInput: HTMLInputElement){
45. console.log(nameInput.value);
46. this.serverCreated.emit({serverName : nameInput.value,
47. serverContent: this.newServerContent});
48. }
50. onAddBlueprint(nameInput: HTMLInputElement){
51. this.blueprintCreated.emit({serverName : nameInput.value,
52. serverContent: this.newServerContent});
53. }
54. }

Start from 69