13)How does angular app gets loaded and started

We can see that when we run our app, we see the content of app component in browser. So how does angular know that it should show the contents of app.component.html? you can argue that it is only component in our app, right now. But that is not the reason and actually this not the file served by server. Instead the index,html file here is served by served and remember that I told you angular is the framework that allows you to create single page applications. This is single page that is served- index.html. lets see index.html-

Index.html file is server by server that is spun up by angular cli. In index.html file we use the selector of our root component.

Index.html-

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Angular - The Complete Guide</title>

<base href="/">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" type="image/x-icon" href="favicon.ico">

</head>

<body>

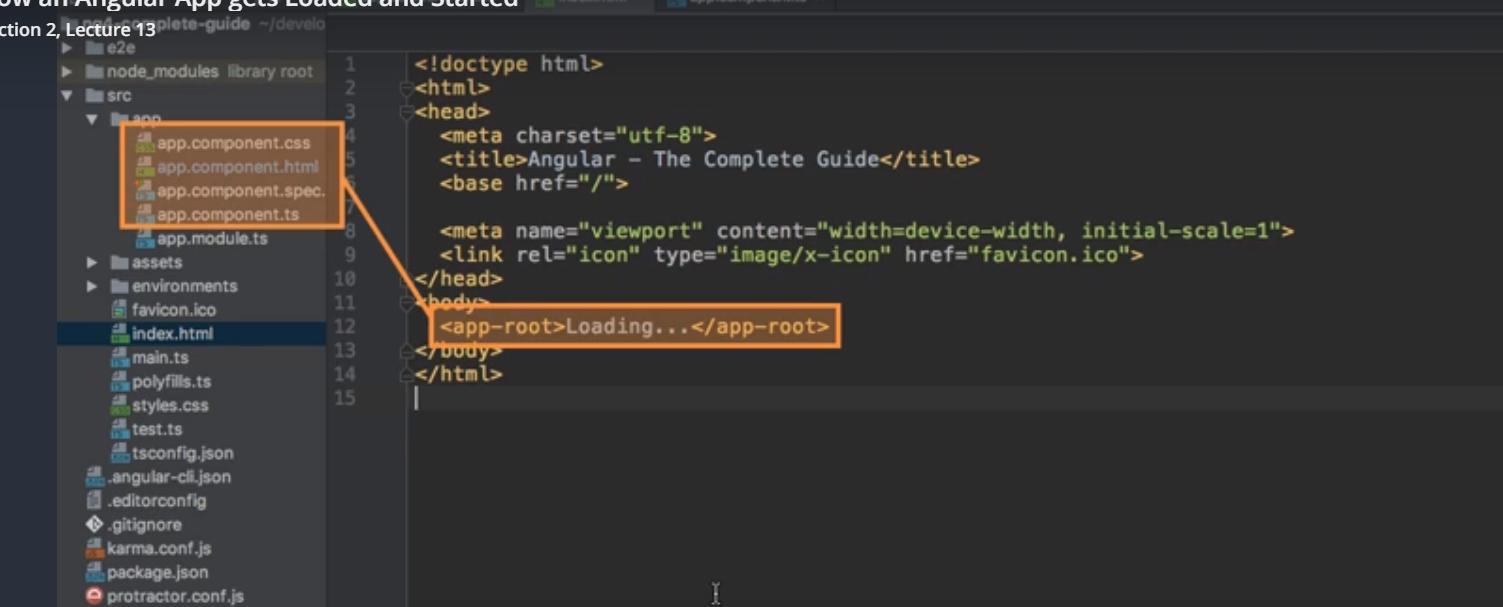
<app-root>Loading...</app-root>

</body>

</html>

It is normal html file but in body of this file we have got, app-root thing with loading… between it. We do not see loading… in our final app.

App-root is not default html element instead it is a selector of one of our components. The cli has created one component for us, the root component of our application, the component which will tie our whole application togather itn the end. So this selector is only information that angular needed to be able to replace this part here in index.html with template of app.component.(that is component having this selector).



Now missing information is how is angular triggered. How is it kicked off to actually run over our body of index.html and answer is ,in the final index.html which is getting served in browser,we can see that we have some script imports that we cannot see in our editor



These are injected by cli automatically. When ever ng serve process rebuilds our project, it will create bundles, javacsript script bundles and it automatically adds the right imports in index.html. these script bundles will contain our own code too. Therefore these script files are executed and they are actually first code to be executed. First code to be executed is that we write in main.ts. main.ts-

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

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';

import { environment } from './environments/environment';

if (environment.production) {

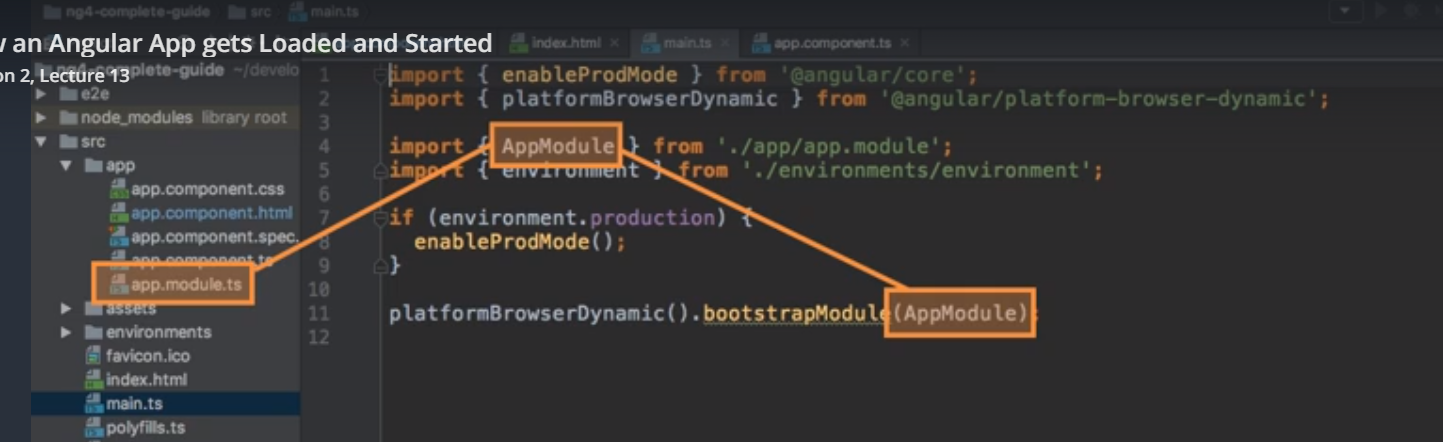
enableProdMode();

}

platformBrowserDynamic().bootstrapModule(AppModule)

.catch(err => console.log(err));

Here we have some imports then we see whether we are in production mode or not. You basically turn off some warning messages I can tell you that. Most important line is last one. This bootstrap(starts) our angular application by passing app.module to this method.



In app.module, we have bootstrap array which basically list all componens which should be known to angular at point to time it analyses our index.html. here we circle closes because here we refrences our app.component. so angular knows the selector of app.component when it analyses the index.html file. therefore it is able to handle selector of app.component in index.html file. so it inserts the html of app.component in place of selector of app.component.

App.module.ts-

import { BrowserModule } from '@angular/platform-browser';

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

import { FormsModule } from '@angular/forms';

import { HttpModule } from '@angular/http';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

FormsModule,

HttpModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

this is how angular application gets started, this is what we see why we see.

15)Creating a new Component

Components are key feature in angular. You build your whole applicaton by composing it from a couple of component which you create on your own. We do start with app.component which is root component you should say, which holds our entire application in the end.

Component is simply a typescript class, so that angular is able to instaniate it to create object based on blueprint, that we setup here you could say. We export the class so that we can use it outside the class. Se create a normal class like this-

export class ServerComponenet { }

now we should add something to it which tells angular that this is not only normal typescript class but instead something special, a component. we do this by adding a special decorator.Decorators are typescript features which allow you to enhance your classes for example, enhance elements you use in your code , it’s not limited to classes. Decorator we use is **@Componennt**  and decorators are always attached by adding **@** sign in front of them. Now this component decorator is not something that typescript knows from start, so we have to import it.

In import we mention that we are importing Component bcoz file from which we are importing it, we can import other things too, so we mention explicitly that we need @Component. we import it from angular package. Angular ships with couple of packages where it groups its functionalities and core package as name implies gives us access to some of core fucntionlities of angular. Now this @Component decorator is known to typescript when it parses the file and compiles it to javascript, it is able to understand it. now we need to pass a object to decorator and configure this component. because without any configuration it’s still not that valuable to angular. But here we can store some important information which will be stored as meta data for this class in background , then which will tell angular what to do with this class.

Code-

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

@Component({

selector: 'app-server',

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

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

})

export class ServerComponent implements OnInit {

constructor() { }

ngOnInit() {

}

}

Here path of html and css are relative to component. This path is needed by webpack because at last everything is bundled by webpack, so we need to tell it where it can find html find.

We can also create component by cli. Run this command-

**ng g c “name of component”**

16)Understanding the role of app module and component declaration

Angular uses components to build webpages and uses modules to basically bundle different pieces, for example components of your app into packages. Right now we have only one module. In bigger project you can split your app into multiple modules.

What is app.module? well its bundle of functionalities of our app and it basically gives angular information which featues my app has and use.

App.moule(or modules in general), they are also normal typescript classes. We transform it into something else by adding a decorator.

App.module-

import { BrowserModule } from '@angular/platform-browser';

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

import {FormsModule} from '@angular/forms';

import { AppComponent } from './app.component';

import { ServerComponent } from './server/server.component';

@NgModule({

declarations: [

AppComponent,

ServerComponent

],

imports: [

BrowserModule,

FormsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

in object that we passed to @Ngmodule we have 4 properties- declarations,imports, providers and bootstrap. We had a look at bootstrap, this was responsible for telling angular which compoents shpuld it be aware of at point of time the whole application starts. So which compoents would you basically recognize in index.html and that was AppComponent. We wnt add new component to index.html, this is not how the angular works.as we are not adding anything to index.html , we wnt touch bootstrap array. still we added a new component and this new componeent needs to be registered in app.module so that angular knows that this component exists because and this is important, by default angular will not scan all your files here. So if you dnt tell it that this new component exists it does’nt know it. Just creating file is not enough. That is why we have to register it here. In ngModule to tell angular, hey part of this module (here our whole app bcoz we have single module) is server component.

We will add our newly created component to **declaration**  array. All components needs to registered in app.module .Before adding you have to tell angular where this component lives. So import component. This is because we need to tell typescript where it can find this class. While importing dnt add .ts extension at the end, this is because it is added by webpack which bundles our project automatically. Now ts knows where to find it.

Now we can use this component.

Now you might wonder what does provider and imports do. Imports allows to add other modules to this module. You can split your app into multiple components and then you can import these modules to basically make this module bit leaner and outsource some stuff into another module. we can also import some modules built into angular. Angular itself is split into multiple modules, BrowserModule for ex gives you all the base functionality we need to start our app, we will come back to forms and http modules later. E come back to provider in services section.

Question-

<https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/questions/5652092>

19)Working with component templates

Instead of defining templates in some other file.

@Component({

selector: 'app-root',

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

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

})

you can define them in same file .

@Component({

selector: 'app-root',

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

template: `<h1>This is multiline</h1>

<h2>using template literals</h2>`,

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

})

You can use either approach but you need to have atleast one template in component. you can omit the selector, you will later in the routing section learn a different way of loading components without using selector and you dnt have to add styles but templates have to be present.

Just like templates we can have inline styles like that-

@Component({

selector: 'app-root',

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

template: `<h1>This is multiline</h1>

<h2>using template literals</h2>`,

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

styles: [`

h3{

color: blue;

}`]

})

We can have one template but w ecan have multiple styles. that is why we use array here.

21) Fully understanding component selector

In @Component decorator we have property called selector. as I told you selector needs to be unique, so that you do not override accidently an already existing element or maybe avalaible by a number of third party package. Right now selector is-

selector: 'app-server',

we are not limited to this type of selector only. this works like css selector. if we want to style h1 tag we will use this selector-

**h1 {**

**}**

In same way above selector is used like this –

<app-server></app-server>

Therefore you are limited to selecting by element. You can out it in in square brackets to use attriburte selector-

selector: '[app-server]',

in css we can select elements by attribute by enclosing attribute in square brackets. now our app wnt recognize this-

**<app-servers></app-servers>**

Now we have to use this-

<div app-server></div>

We can also select by class

selector: '.app-server',

in html-

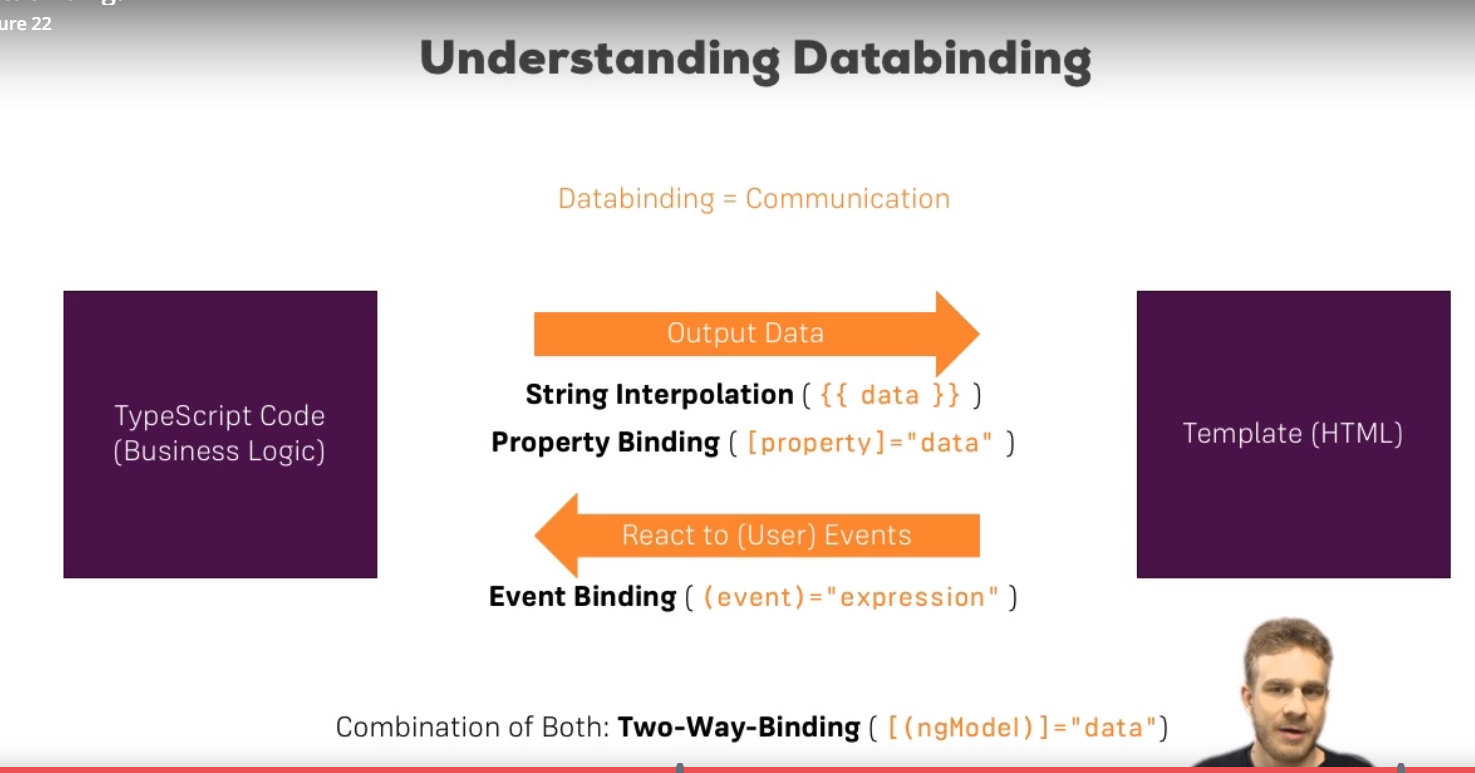
<div class="app-server"></diV>

So these are options that you have. As a sidenote selecting by id wnt work. It is not supported by angular. Psedo selector like hover and so on also wnt work.

Normally we select it by element by compoennts. We will soon learn about directives, another feature, where this is different. but for components you create your own elements and therefore you dnt use class and attribute style.

22)What is Databinding

You could basically translate data binding with communication, communication between your typescript code and html code of component.



Now there are different types if communication. We might want to end data from ts file to html(outout data). Then we might want to react to user events.

Then we also have one additional form of data binding where we combine both directions, 2 way data binding, where we are able to react events and output something at the same time. so these are 4 forms of data binding.

23)String interpolation

Between **{{}}** you could write a type script expression. Any expression that resolves into strong into the end, it is only condition for string interpolation syntax.

<p>my name is {{name}}</p>

It will look for variable name in typescript class.

<p>my name is {{'sumeet'}}</p>

Sumeet is a ts expression that resolves into string so this will work fine.

So what ever you have in between curly braces in the end it has somehow has to return a string. So you could also return a method here which returns a string in the end. Only other restriction is you cannot write multiline expressions in here, you cannot write block expressions in here, so you can’t add a if or for control structure in here. You could use a ternary expression though.

Html-

<p>my name is {{getName()}}</p>

Ts-

getName() {

return 'Sumeet Sood';

}

We can do this-

{{id}}, where id is number. But we said that string interpolation has to resolve to a string in the end, has to get a string in the end. It will work because number can be easily converted into a string. So you have to get string in the end or something which can be converted to a string ,really to be correct here.

24)Property Binding

There are lot of cases in which you can use either property binding or string interpolation. I will show what I mean in a second.

We have button-

<button class="btn btn-primary" disabled="true">Add Server</button>

Here button will be disabled permanently. Lets say we want to disable or enble it depending upon value of a variable. Lets say we have variable called allowNewServer and we want to disable or enable button depending upon value of this variable so we want to bind disabled property to allowNewServer.

To make this disabled functionality dynamic, we can bind to allowNewServer by enclosing it in square brackets.

<button class="btn btn-primary" [disabled]="allowNewServer">Add Server</button>

square indicates to angular that we are using property binding ,that we want to dynamically bind some property and disableb the html attribute in the end, in normal html only sets a specific property on underlying DOM element. You might notice that each html element you use is parsed by the browser and kind of translated into an element on document object model. this is completely unrelated to angular. And therefore we have an element in this DOM and this element has couple of properties. A lot of these can’t even be set thorough attributes on HTML element. Well one of the properties is disabled property and you can set the it through disabled attribute. But here we are not using disabled attribute anymore with the square brackets we are directly binding to this native disabled property, this html element has.

Between “” we have a expression which for this property resolves to a Boolean. For other properties other type might be required, because again you can bind to a lot of properties, basically to all the html element properties. We will soon learn there are also other properties that you can bid too.

So here we pass our property name i.e allowNewServer because it will resolve to true or false.

Now you can see whenever you change value of the variable in typescript, button is enables or disabled. It is because now we are binding to this disabled property, this native element property, we are binding it to own typescript property here and covinient thing is , and this what angular is all about- it will update dynamically. We can made button enabled or disabled by changing value of this variable in class. One of the main reasons why you use angular. It makes it very easy for you to interact with your DOM to change something at runtime, like we do here. That is property binding in place. now as I mentioned besides binding to HTML element property, we can bind to properties of directives and components.

Question-

<https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/questions/5793182>

25)Property Binding vs String Interpolation

<p>{{allowNewServer}}</p>

<p [innerText]="allowNewServer"></p>

Here stringInterpolation and proeprtyBinding are used interchnagibly.

However this is not valid and it will break the app-

<p [innerText]="{{allowNewServer}}"></p>

Between quotation marks of property binding, you can and you must write typescript code, I mean a typescript expression which will return the value this property expects. So just like string interpolation, You may also call a method there but you must not add curly braces there.

<button

class="btn btn-primary"

[disabled]="allowNewServer">Add Server

</button>

It might look that we are using a html attribute there, but we are not. This whole syntax is recognized by angular. This is why between quotation mark you have to write a ts expression, this whole expression is evaluated by angular. Mixing in string interpolation will break it. String interpolation only works in normal template not with in another expression of that template, not within property binding or something like this.

26)Event Binding

Lets say we want to react to user events.

In html-

<button

class="btn btn-primary"

[disabled]="allowNewServer"

(click)="onCreateServer()">Add Server

</button>

In ts-

onCreateServer() {

this.serverCreationStatus = 'Server was created';

}

}

On in beginning ,kind of makes it clear that it will be triggered from template. You dnt have to name it ‘on Somethng’ but it kind of makes it easier to understand who will call this method? Well user will basically, by doing something, some event will call it. That is what I want to symbolize here.

On html element we have onclick attribute on html element here we can execute some js code. Here we used syntax that angular offers. Here we used (), in property binding we used []. Between () we specify the name of the event.

You can basically bind to all events made avalaible by html element , you are placing this on. Like we have onMouseEnter in html, here we have mouseEnter. We can put the code that we want to execute between “”. Most of times we call function but we can also execute coe directly here.

27)Bindable properties and events

How do you know to which Properties or Events of HTML Elements you may bind? You can basically bind to all Properties and Events - a good idea is to console.log()  the element you're interested in to see which properties and events it offers.

**Important**: For events, you don't bind to onclick but only to click (=> (click)).

The MDN (Mozilla Developer Network) offers nice lists of all properties and events of the element you're interested in. Googling for YOUR\_ELEMENT properties  or YOUR\_ELEMENT events  should yield nice results.

28)Passing and using data with event binding

Lets we to show which key user has pressed. We can do it like this-

Html-

<input type="text" class="form-control" (input)="onUpdateServerName($event)">

<p>{{serverName}}</p>

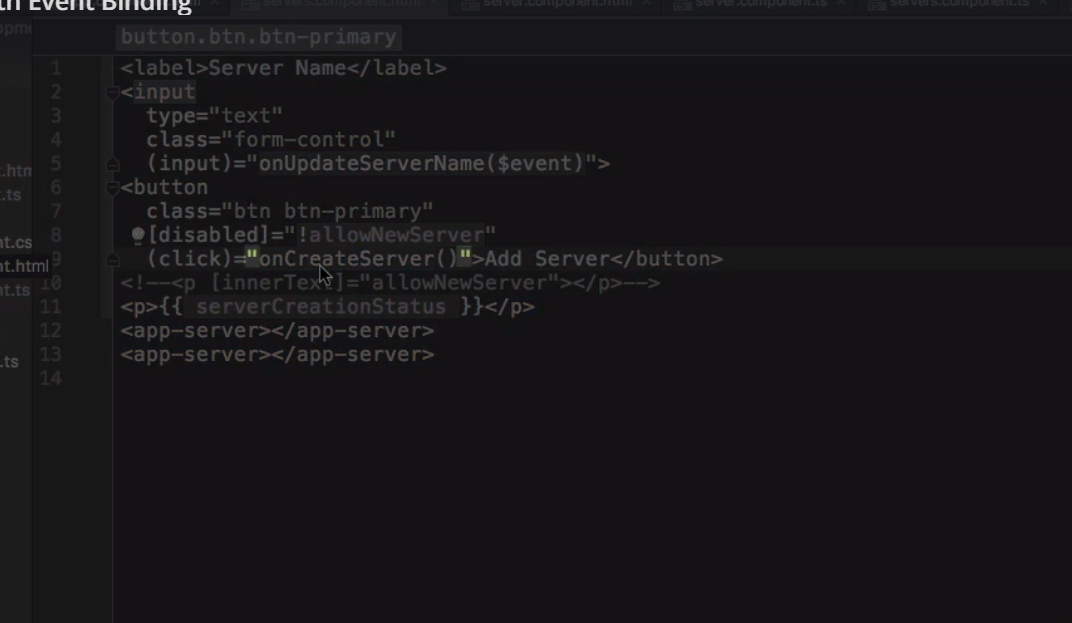
Ts-

onUpdateServerName(event: Event) {

this.serverName = (<HTMLInputElement>event.target).value;

}

$event is a reserved variable name, you can use in the template when using event binding. So for this event, so for only between this quotation marks, $event will be the data emitted with that event. So events provide us with some data when they are fired.



s

On this event we have target property, which represent the html element on which event was fired. **now question is how do we know the type of event will be Event? First assigned type any to event then just print it in console, you will get the type.**

Now value property does not exist on all html elements, our IDE will give error, so we tell it that this element is of type input. This is just needed to inform ts that we know that type of html element of this input event will be html input element. We do this explicit casting by <> and type between these <>.

29)Important: FormsModule is required for 2 way data binding

Important: For Two-Way-Binding (covered in the next lecture) to work, you need to enable the ngModel  directive. This is done by adding the FormsModule  to the imports[]  array in the AppModule.

You then also need to add the import from @angular/forms  in the app.module.ts file:

import { FormsModule } from '@angular/forms';

30)Two-Way DataBinding

Lets combine property and event binding. Syntax of 2 way data binding is also formed by combine syntaxs of event binding and property binding.

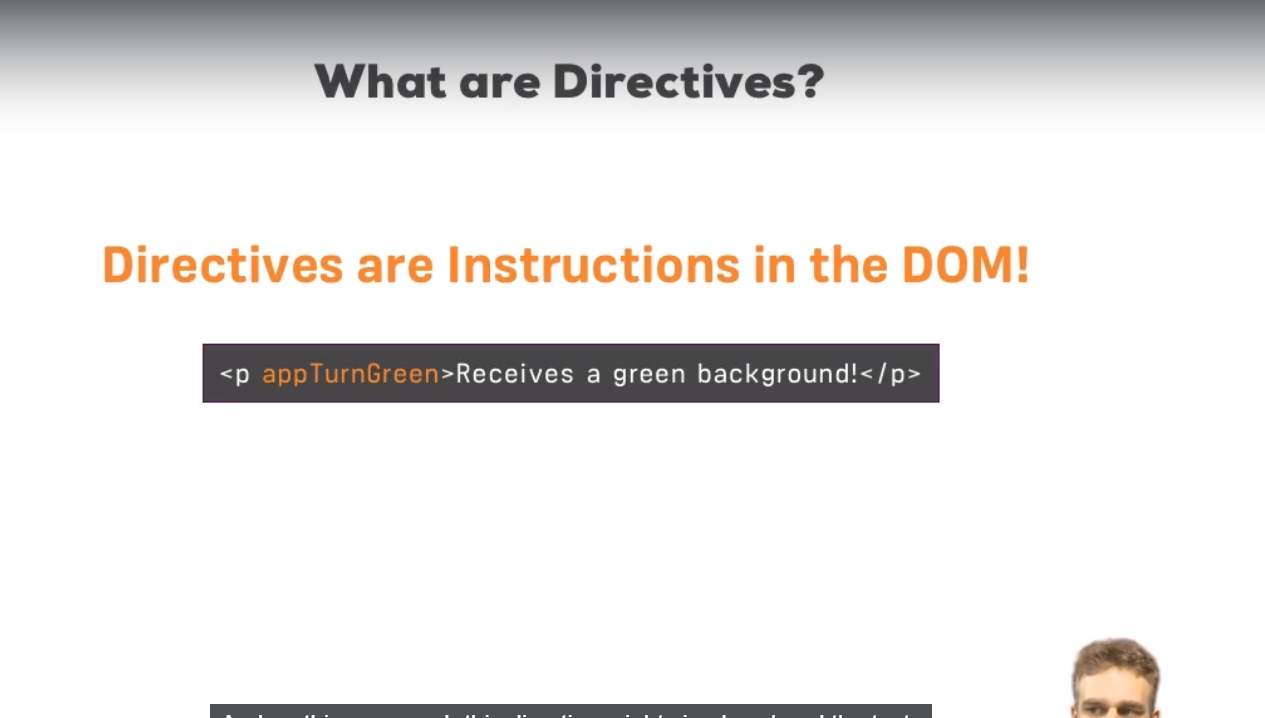
ngModel is a special directive, we make it equal to some property defined in ts class.

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

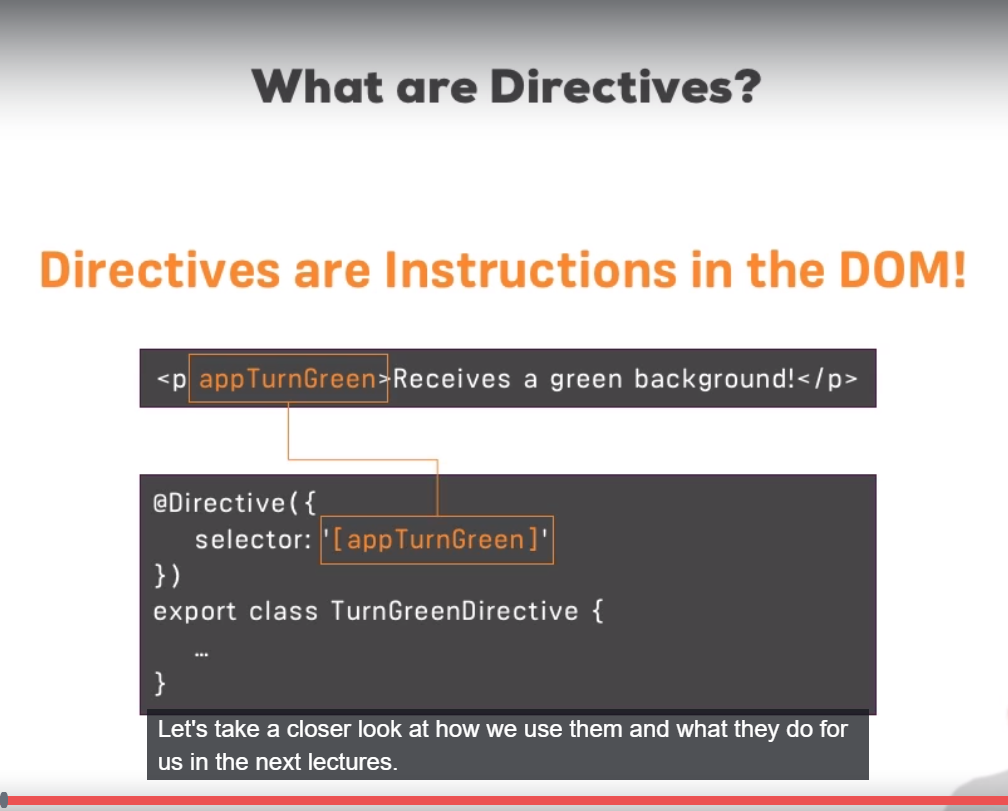
This will do following, it will trigger on the input event and update the value of server name in our component automatically. On the other hand, since it is 2 way binding it will also update the value of input element when, if we change server name somewhere else.

32)what are directives

Directives are instructions in the DOM and we already use directives without knowing it. Components are such instructions in the DOM, once you place the selector of a component somewhere in our template, at this point of time, we are instructing angular to add content of our component template and the business logic of our ts code in this place where we use the selector. this was our instruction. indeed components are directives, but directives with the template. There are also directives without the template. Example would be-



Here **appTurnGreen** is a directive. It would be a custom directive that we could build. we typically add directives with the attribute selector. but technically selector of directive can be configured just like selector of a component. so you could use css classes or element style but again typically we use the attribute style. And on this paragraph, this directive might simply color the text green you could say.



So angular would find this instruction, here we would have defined our directive with @Directive decorator to inform angular that this class holds a directive and there we might have logic to turn the screen.

33)using ngIf to output data conditionally

<p \*ngIf="serverCreated">Server was created, server name is {{serverName}}</p>

\* is required because ngIf is a structural directive which means it changes the structure of the DOM, it either adds this element to DOM or it does’nt add it. So that;s just extra information for angular. The directive itself is just ngIf but \* is required without it, it will not work correctly.

Between “” marks we can setup our condition. For ngIf, it has to be any condition returning true or false. Deciding whether this should be added or not.

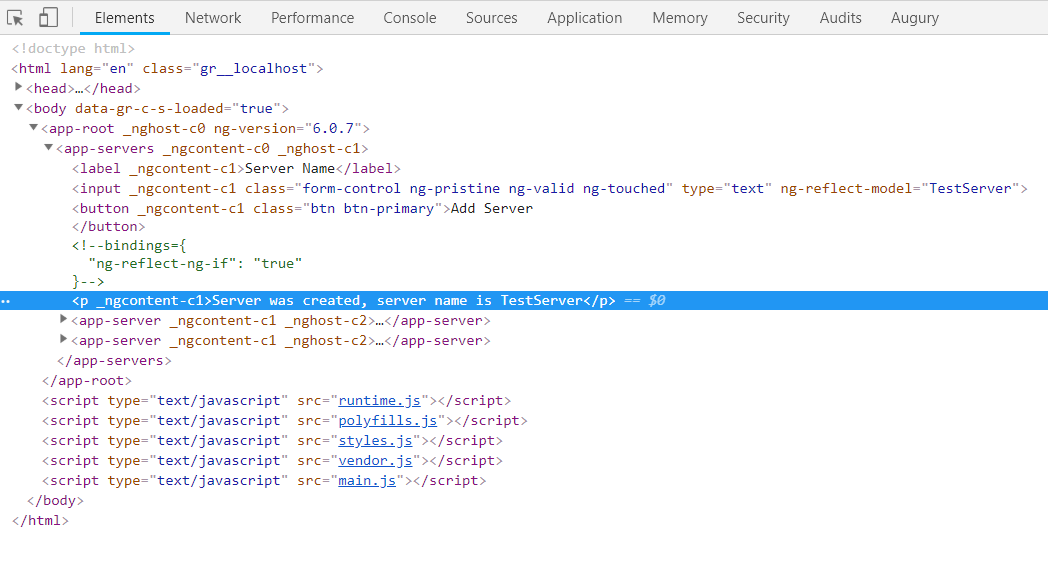
this element will be displayed if expression passed to \*ngIf is evaluated to true**. You can also call a function here, we can also evaluate a expression here**. here we our expression is just variable name.

Note that it is not hidden. Either it is added to DOM or removed from.

When element is not displayed-



When it is displayed-



The comment you see is kind of hook which angular created to know where it should be displayed. If we remove \*ngIf from there then this comment wnt be there.But important thing is it’s really added or removed to or from the DOM. It’s not there all the time, it’s not hidden, it’s just not there. Which is super important to understand.

34)Enhancing ngIf with an else condition

<p \*ngIf="serverCreated; else noServer">Server was created, server name is {{serverName}}</p>

<ng-template #noServer>

<p>No server is created</p>

</ng-template>

We contain p tag in ng-template tag. That is component, directive shipping with angular which you can use to mark places in DOM. Then we place local refrence on it. With local refrence we have marked certain spot in dom which we want to show conditionally. Then we place this refrence in \*ngIf syntax.

35)Styling Elements Dynamically with ngStyle

Other type of directive is attribute directives which are called so because they really just look like normal html attribute without a star basically. Unlike structural directives, attribute directives don’t add or remove elements. They only change the element they were placed on.

<p [ngStyle]="{'background-color': getColor()}">{{'Servers'}} with ID {{serverId}} is {{getServerStatus()}} </p>

Ts file-

getColor() {

return this.serverStatus === 'Online' ? 'green' : 'red';

}

Here square brackets are used because we want to pass value to variables of directive, so we used property binding.[] are not part of directive name. they indicate that we want to bind to a property on this directive and this property name also happens to be ngStyle. As a value this directive exepects the key value pairs. Key is style name and value is value of that style. Note here we used background-color and ‘-‘ cannot be used in property name so we have to use ‘’. Alternative is to use camel case notation- backgroundColor. Alternate is to use this-

<p [ngStyle]="getColor()">{{'Servers'}} with ID {{serverId}} is {{getServerStatus()}} </p>

In ts-

getColor() {

return this.serverStatus === 'Online' ? { 'color': 'green' } : { 'color': 'red' };

}

However this wnt work-

<p [ngStyle]="{getStyle(): getColor()}">{{'Servers'}} with ID {{serverId}} is {{getServerStatus()}} </p>

Ts code-

getColor() {

return this.serverStatus === 'Online' ? 'green' : 'red';

}

getStyle() {

return this.serverStatus === 'Online' ? 'color' : 'color';

}

}

So ngStyle allows us to dynamically update the styles.

36)applying css dynamically with ngClass

It allows us to dynamically add or remove classes. This directive also takes a js object as value. keys are name of class, value is js expression evaluating to true or false. If expression is true, then class is attached otherwise it is not attached. You have to wrap class name in ‘’ if class name is not valid property name in js.

<p

[ngStyle]="{'background-color': getColor()}"

[ngClass]="{online: serverStatus === 'Online'}">

{{'Servers'}} with ID {{serverId}} is {{getServerStatus()}} </p>

Css-

.online {

border: 2px solid black;;

}

Just like ngStyle, this wnt work-

<p

[ngStyle]="{'background-color': getColor()}"

[ngClass]="{getClassName(): serverStatus === 'Online'}">

{{'Servers'}} with ID {{serverId}} is {{getServerStatus()}} </p>

In ts-

getClassName() {

return 'online';

}

But this will work-

Html-

<p

[ngStyle]="{'background-color': getColor()}"

[ngClass]="getClassObject()">

{{'Servers'}} with ID {{serverId}} is {{getServerStatus()}}

</p>

In css-

getClassObject() {

return { 'online': this.serverStatus === 'Online'};

}

37)Outputting Lists with ngFor

<app-server \*ngFor="let server of servers"></app-server>

It is also structural directive, changing the DOM.so we use \*. The ngFor base syntax looks like this , we define a temporary variable for inside a loop with let , can give it any name like here we gave it name server then we have ‘of servers’. Servers is property that we define inside typescript class, and this will now loop through all elements in this array and assigned the individual element to the dynamic server variable. This is normal thing in loop.

Now this server variable can be sued in template. Here our app-server element will be repeated as many times as number of element in array. So if we add new element, we will have new server component.

If we want to get access to index of loop it can be done like this-

<app-server \*ngFor="let server of servers; let i=index"></app-server>

Here I is also local variable for this loop.