CLI Deep Dive & Troubleshooting

Section 1, Lecture 4

In the next lecture, we're going to build our first little app!

If the CLI prompts you to**answer some questions**(some versions do that), you can simply hit **ENTER** for all questions. This will accept the default settings which are fine for this course.

The CLI generates a different welcome screen than you're going to see in my video though. No worries, you'll still be able to follow along without issues! Just make sure to code along **so that your code equals mine** - Angular itself didn't change a bit :)

Depending on the CLI version you're using, you might also need to add the FormsModule  to the imports[]  array in your app.module.ts  file (add it if you don't see it there). You might not fully understand what that all means but we're going to cover that in this course, no worries.

If you don't have FormsModule  in imports[]  in AppModule , please do add it and also add an import at the top of that file: import { FormsModule } from '@angular/forms';

-----

If you want to **dive deeper into the CLI** and learn more about its usage, have a look at its official **documentation**: <https://github.com/angular/angular-cli/wiki>

**You encountered issues during the installation of the CLI or setup of a new Angular project?**

A lot of problems are solved by making sure you're using the latest version of NodeJS, npm and the CLI itself.

**Updating NodeJS:**

Go to nodejs.org and download the latest version - uninstall (all) installed versions on your machine first.

**Updating npm:**

Run [sudo] npm install -g npm  (sudo  is only required on Mac/ Linux)

**Updating the CLI**

[sudo] npm uninstall -g angular-cli @angular/cli

npm cache clean

[sudo] npm install -g @angular/cli

**Here are some common issues & solutions:**

1. **Creation of a new project takes forever (longer than 3 minutes)**  
   That happens on Windows from time to time => Try running the command line as administrator
2. **You get an EADDR error (Address already in use)**  
   You might already have another ng serve process running - make sure to quit that or use ng serve --port ANOTHERPORT  to serve your project on a new port
3. **My changes are not reflected in the browser (App is not compiling)**  
   Check if the window running ng serve  displays an error. If that's not the case, make sure you're using the latest CLI version and try restarting your CLI



**Commands**

1. npm install -g @angular/cli
2. ng new my-first-app
3. ng serve



1. npm install –save bootstrap@3
2. ng generate component servers

ng g c servers

ng generate pipe filter

ng g p filter

ng build –prod --aot

If you want to output something in your template print some text to it use string interpolation, if you want to change some property be that of an angular element or as you will later learn of a directive or a component.

No {{}} braces for property binding

Bindable Properties and Events

**Section 2, Lecture 27**

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.

**Important: FormsModule is Required for Two-Way-Binding!**

**Section 2, Lecture 29**

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';

**Directives**

star is required because ng-if is a structural directive which means it changes the structure of our DOM. It either adds this element or it doesn’t add it.

Unlike structural directives, attribute directives don’t add or remove elements. They only change the elements they are placed on. They look like normal HTML attributes without a star basically.

**Installing Bootstrap Correctly**

**Section 3, Lecture 41**

In the next lecture, we set up the course project. For that, we'll install the Bootstrap CSS Framework.

In this course, we use version 3 of the framework, install it via npm install --save bootstrap@3 => The @3 is important!

Additionally, when using a project created with Angular CLI 6+ (check via ng -v ), you'll have an angular.json file instead of an .angular-cli.json file. In that file, you still need to add Bootstrap to the styles[] array as shown in the next video, but the path should be node\_modules/bootstrap/dist/css/bootstrap.min.css , NOT ../node\_modules/bootstrap/dist/css/bootstrap.min.css . The leading ../ must not be included.

Also see this lecture - I do show the complete setup process there: <https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/t/lecture/6655614/>

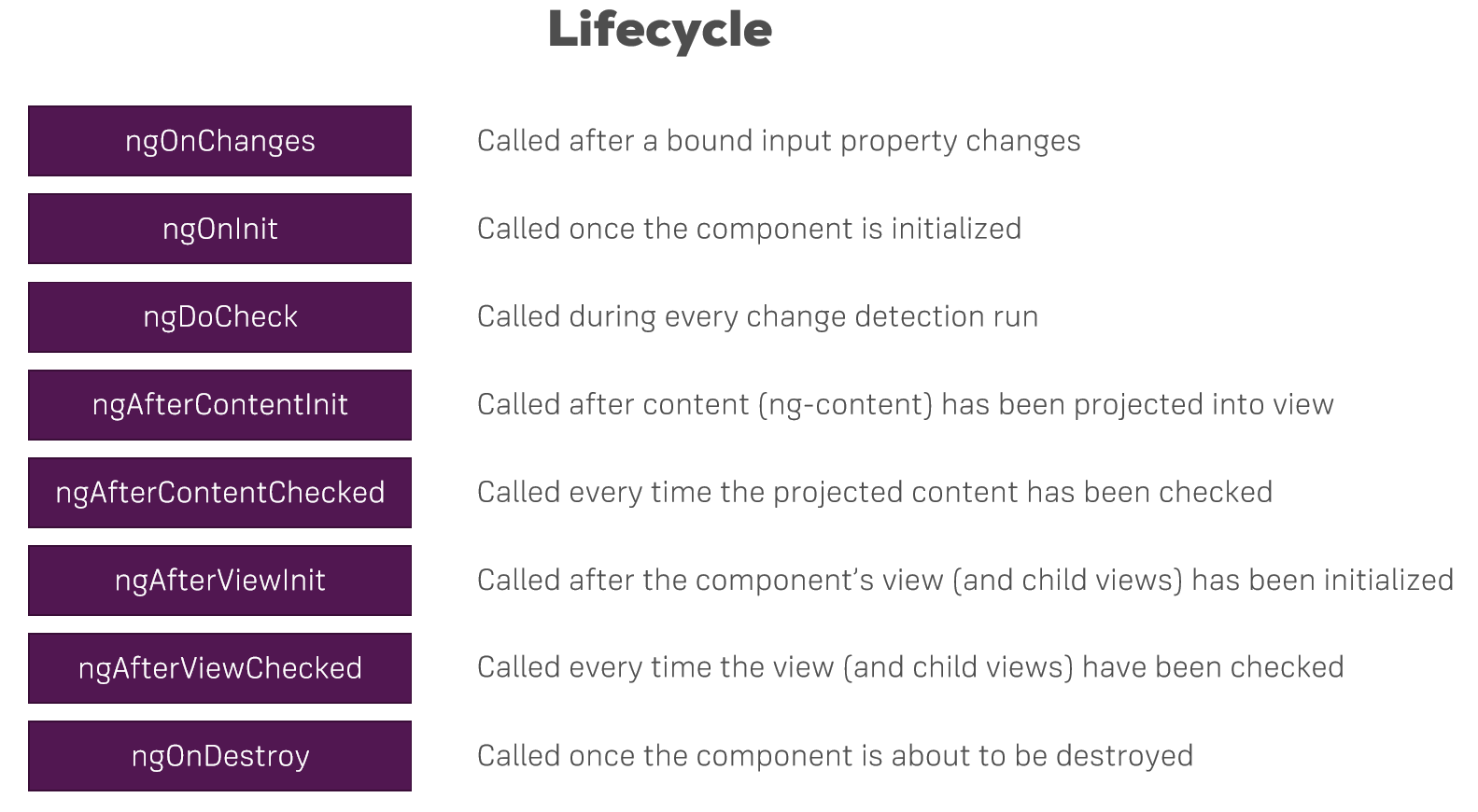
**Alternative Non-Collapsable Navigation Bar**

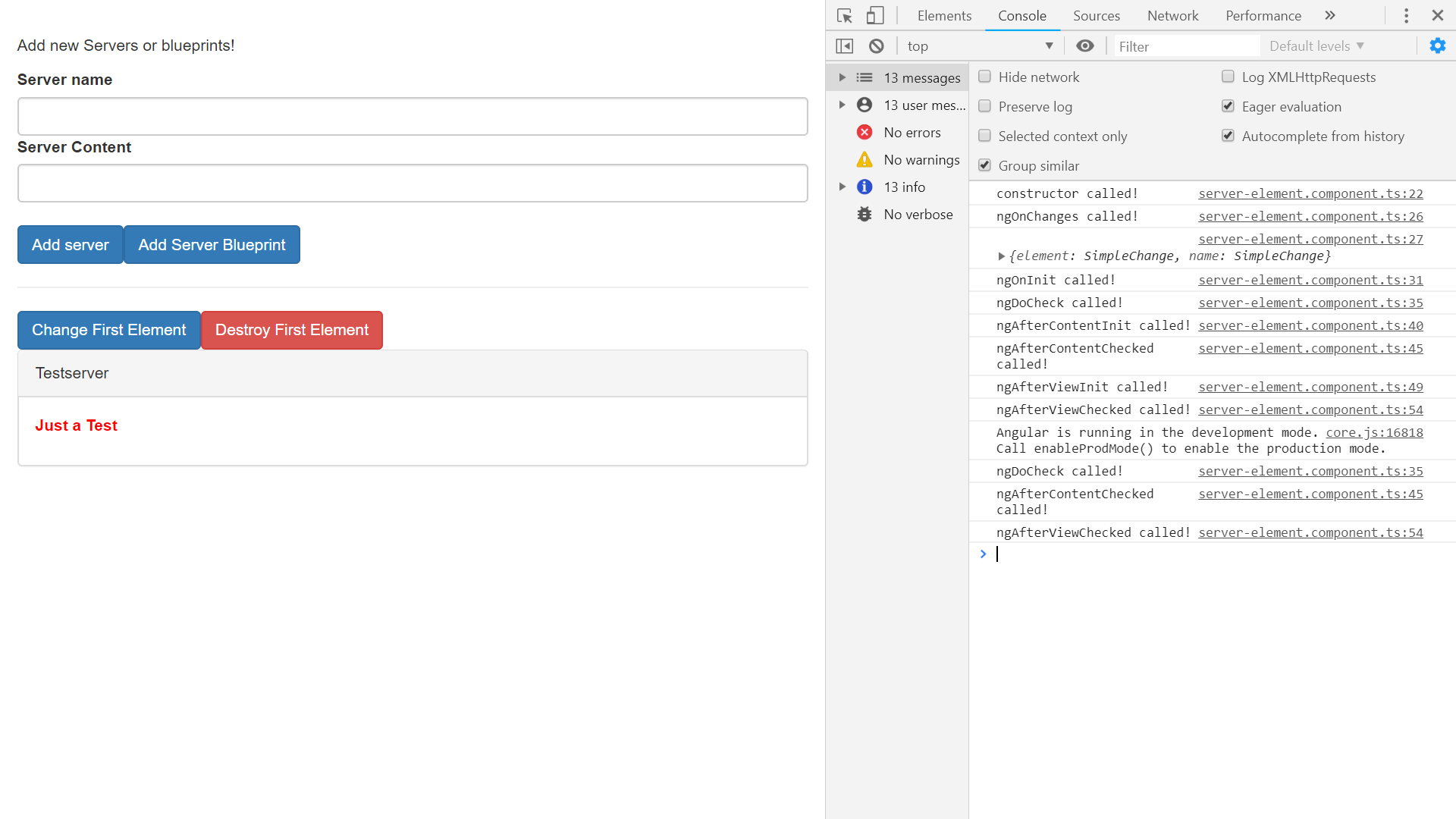
**Section 3, Lecture 46**

The way we added it, the Navbar will collapse on smaller screens. Since we didn't implement a Hamburger menu, that means that there's no way of accessing our links on smaller screens.

You can either add such a menu on your own, or you replace collapse navbar-collapse with just navbar-default.

**Angular Lifecycle hooks**





If a new component is created in angular. And of course angular is responsible for creating these components when it finds one of our selectors. for example it will instantiate a new version of that component and add it into the dom. So once a new component is instantiated angular goes through a couple of different phases in this creation process and it will actually give us a chance to hook into these phases and execute some code. We can hook into these phases by implementing some methods angular will call if they are present.

NgOnChanges

The first phase to first hook we can hook into is N.G. on changes and this may actually be executed multiple times it's executed right at the start when a new component is created but thereafter it'salso always called whenever one of our bound input properties changes.And with that I mean properties decorated with @input.So whenever these properties received new values.

NgOnInit

This method gets executed once the component has been initialized. This does not mean that we cannot see it.It has not been added to the DOM yet so to say it has not been displayed yet but Engler finished a basic initialization.Our properties can now be accessed and initialized for example.So the object was created you could say.And if you're interested ngOnInit would run after the constructor

NgDoCheck

This will also run multiple times.Actually this method will be executed a lot because this will run whenever change detection runs.Now change detection simply used a system by which anguler determines whether something changed on the template of a component or inside of a component or should say.So whether it needs to change something in the template so whether some property value changed from 1 to 2 let's say.And that property is output in the template.Well of course angular needs to re render that part of the template and ngDoCheck is a hook executed on every check angle or makes now important on every check.So not just if something changed.A lot of times ngDOCheck do check will run because you clicked some button which doesn't change anything.But still it's an event and on events Engler has to check if something changed because how else would it know. Now this might sound very inefficient anguler does it in a very efficient way. So change direction angler works pretty great and doesn't cost a lot of performance and you do check. It is a great method to use if you want to do something on every change detection cycle like maybe manually inform angular about some change and would not be able to detect otherwise though that is a very advanced use case.

ngAfterContentInit

This is called whenever the content which is projected by the ng-content has been initialized. So not the view of the component itself but instead you could say the view of the parent component especially the part which will get added to our component through ng-content.

ngAfterContentChecked

This check is executed whenever change detection checked this content we're projecting into our component

ngAfterViewInit

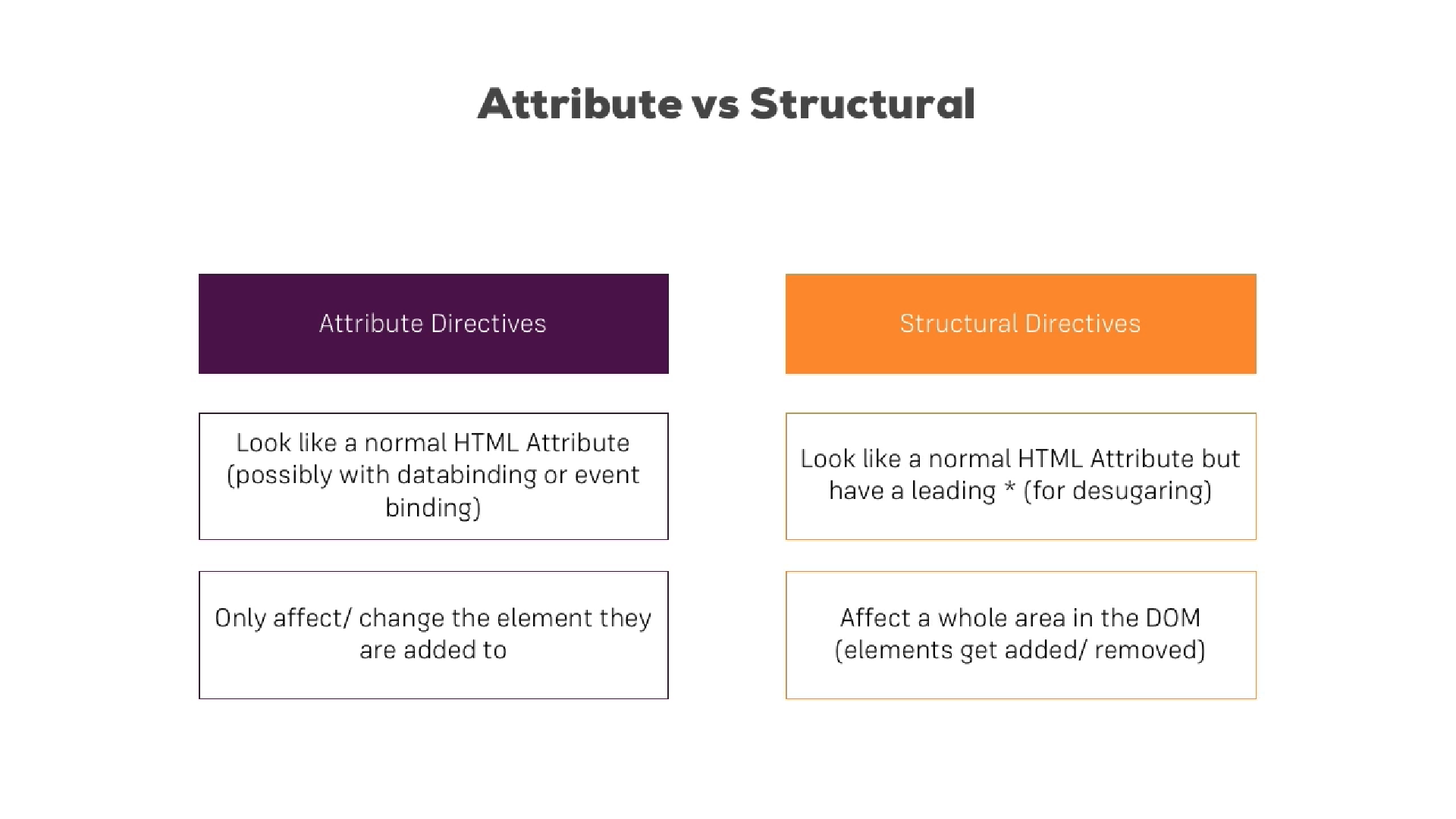
ngAfterViewInit is then reached once the view of our own component has been finished initializing. So once our view has been rendered you could say

ngAfterViewChecked

Same after view checked what that is called whwnever our view has been checked. So once we are assured that well ever all changes which had to be done were displayed in the view or no changes were detected by angular.

ngOnDestroy

And finally if you destroy a component for example if you placed ngIf on it and this then set to false and therefore it removes it from the dom ngOnDestroy is called and here's a great place to do some clean up work because this is called right before the object itself will be destroyed by Angular.

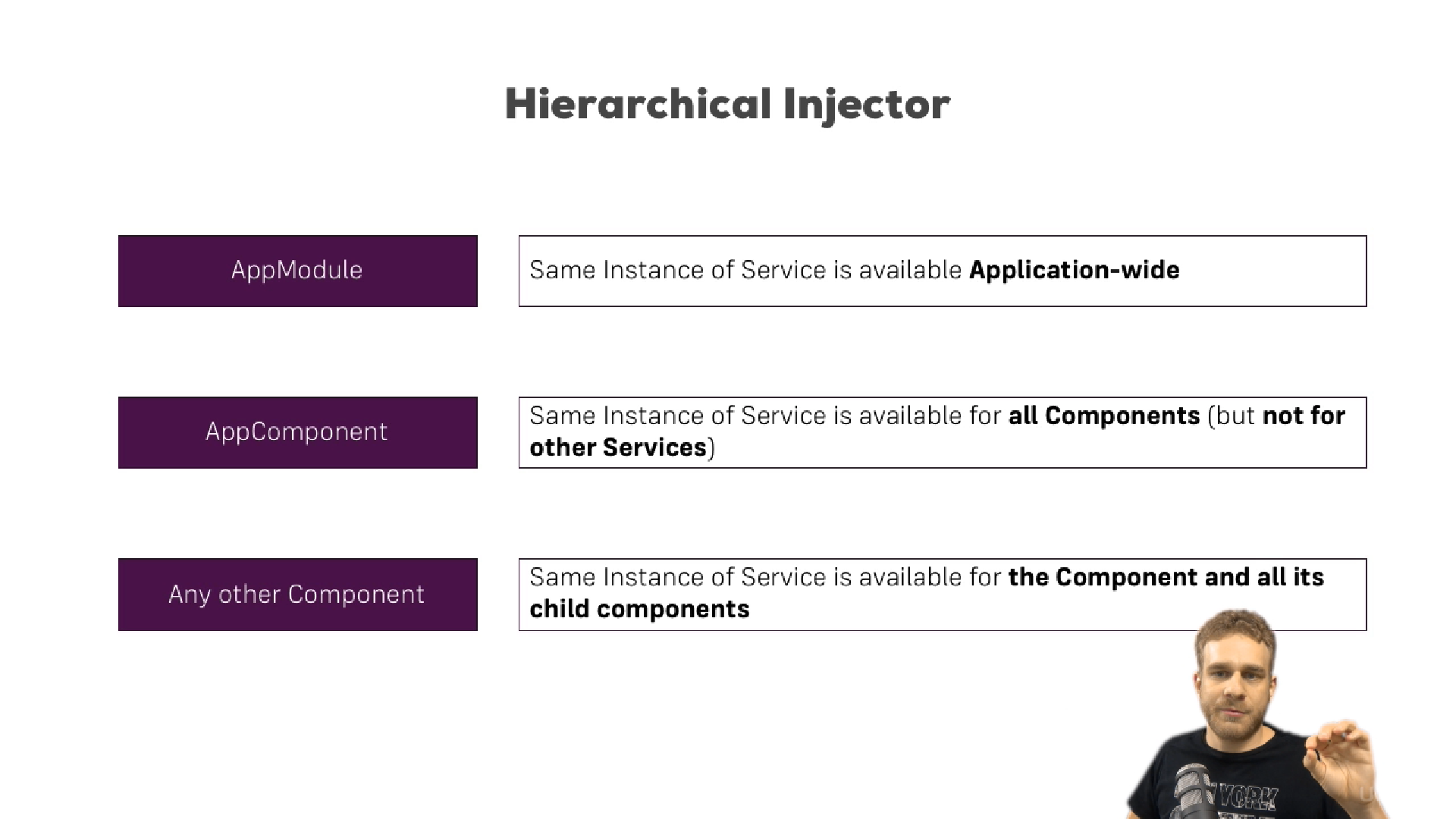


**More about the Renderer**

Section 7, Lecture 88

In the last lecture, we used the Angular Renderer class to change the style of a HTML element. As explained in that lecture, you should use the Renderer for any DOM manipulations. Of course, you can do more than simply change the styling of an element via setStyle(). Learn more about the available Renderer methods here.

**Services**



**Services in Angular 6**

Section 9, Lecture 105

If you're using Angular 6+ (check your package.json to find out), you can provide application-wide services in a different way.

Instead of adding a service class to the providers[] array in AppModule , you can set the following config in @Injectable() :

@Injectable({providedIn: 'root'})

export class MyService { ... }

This is exactly the same as:

export class MyService { ... }

and

import { MyService } from './path/to/my.service';

@NgModule({

...

providers: [MyService]

})

export class AppModule { ... }

Using this new syntax is completely optional, the traditional syntax (using providers[] ) will still work. The "new syntax" does offer one advantage though: Services can be loaded lazily by Angular (behind the scenes) and redundant code can be removed automatically. This can lead to a better performance and loading speed - though this really only kicks in for bigger services and apps in general.

**Understanding the Example Project**

Section 11, Lecture 116

In our app, we got three sections:

Home

Servers

View and Edit Servers

A Service is used to load and update Servers

Users

View Users

This app will be improved by adding routing but definitely feel free to play around with it - besides routing, everything should be working fine.

**Important: Redirection Path Matching**

Section 11, Lecture 134

In our example, we didn't encounter any issues when we tried to redirect the user. But that's not always the case when adding redirections.

By default, Angular matches paths by prefix. That means, that the following route will match both /recipes and just /

{ path: '', redirectTo: '/somewhere-else' }

Actually, Angular will give you an error here, because that's a common gotcha: This route will now ALWAYS redirect you! Why?

Since the default matching strategy is "prefix" , Angular checks if the path you entered in the URL does start with the path specified in the route. Of course every path starts with '' (Important: That's no whitespace, it's simply "nothing").

To fix this behavior, you need to change the matching strategy to "full" :

{ path: '', redirectTo: '/somewhere-else', pathMatch: 'full' }

Now, you only get redirected, if the full path is '' (so only if you got NO other content in your path in this example).

**Forms**

**Built-in Validators & Using HTML5 Validation**

Section 15, Lecture 181

Which Validators do ship with Angular?

Check out the Validators class: https://angular.io/api/forms/Validators - these are all built-in validators, though that are the methods which actually get executed (and which you later can add when using the reactive approach).

For the template-driven approach, you need the directives. You can find out their names, by searching for "validator" in the official docs: https://angular.io/api?type=directive - everything marked with "D" is a directive and can be added to your template.

Additionally, you might also want to enable HTML5 validation (by default, Angular disables it). You can do so by adding the ngNativeValidate to a control in your template.

**Angular 6 and Http**

Section 18, Lecture 237

Angular 6 is currently the latest version of Angular and it deprecates the Http-access method taught in this module.

What does this mean?

It means that the method still works, still is secure - you can use it! But there is a better Http module to use now: HttpClient.

I added a module (section 23) on that new client months ago, even before Angular 5 was released. You'll meet it later in the course and we'll easily update all our Http calls with the new client there.

So for now, follow along with this module here - the core concepts taught here will still apply (i.e. how it works etc).

And later in the course, we'll revisit this solution and update it to HttpClient.

Important

To use Http as shown in this module, you need to install Angular's HttpModule

npm install --save @angular/http

and then, in AppModule, add the HttpModule to the imports[]:

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

...

imports: [..., HttpModule]

**Catching Errors without rxjs-compat**

Section 18, Lecture 248

Are you using Angular 6 (and therefore RxJS 6+) and you're NOT using rxjs-compat (npm install --save rxjs-compat - you may ignore this lecture then, use the code as shown in the videos!)?

You then have to use the catch() operator you'll see in the next lecture a bit differently.

Instead of

....catch(error => {

return Observable.throw(...)

})

use

....pipe(catchError(error => {

return throwError(...)

}))

And make sure to import it:

Instead of

import 'rxjs/Rx';

and

import { Observable } from 'rxjs/Observable';

use

import { catchError } from 'rxjs/operators';

and

import { throwError } from 'rxjs';

**Module Import Order**

Section 21, Lecture 276

In the next module, we'll add routes to a feature module. For this to work, you need to ensure that you get the import order (i.e. the order in which you add all modules to the imports[] array in the AppModule ) right.

To be precise - unlike as shown in the video - you need to position your RecipesModule prior to the AppRoutingModule .

So imports[] should look like this (in AppModule ):

imports: [

..., // Other modules

RecipesModule,

AppRoutingModule

]

This is required to ensure that the Catch-all/ wildcard routes work correctly.

Why does it work in the video (even though I DON'T use that setup there)? Because I recorded this without the wildcard route - a mistake from my side.

**Protecting Lazy Loaded Routes with canLoad**

Section 21, Lecture 286

What if you want to use route protection (canActivate to be precise) on lazily loaded routes?

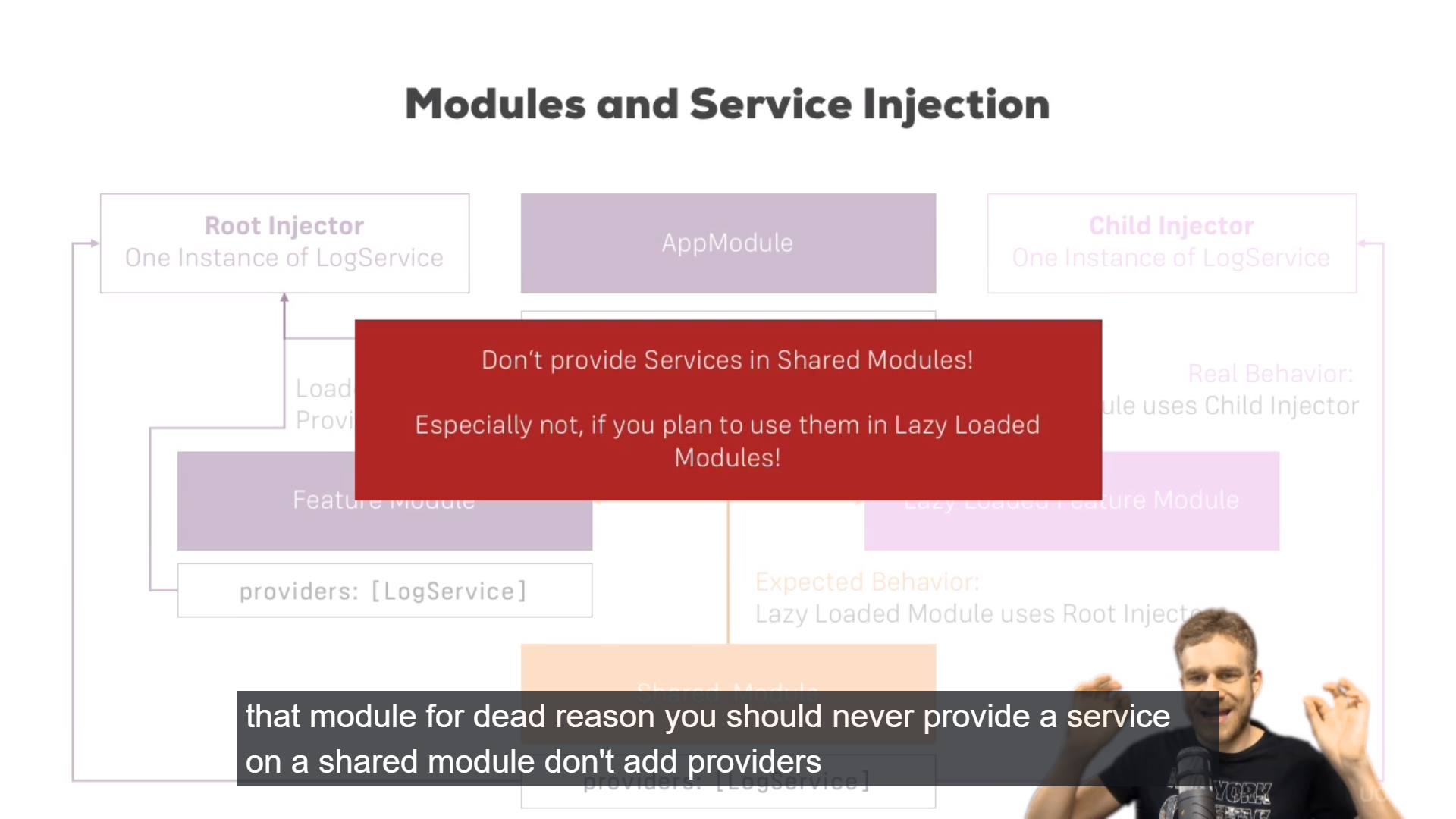
You can add canActivate to the lazy loaded routes but that of course means, that you might load code which in the end can't get accessed anyways. It would be better to check that BEFORE loading the code.

You can enforce this behavior by adding the canLoad guard to the route which points to the lazily loaded module:

{ path: 'recipes', loadChildren: './recipes/recipes.module#RecipesModule', canLoad: [AuthGuard] }

In this example, the AuthGuard should implement the CanLoad interface.

**Don’t add Providers on the shared module**



**Template Interaction & Production Builds**

Section 21, Lecture 291

In the next lectures, we'll build our project for production. This will perform a couple of optimizations and with the current course project, it would also yield some tiny errors. Here's how to fix them!

In the header.component.ts file (and its template file), you need to change the way you access the "is the user authenticated?" information in the template.

Instead of using

\*ngIf="authService.isAuthenticated()"

in the header.component.html file, you should replace that with a method call, like that:

\*ngIf="isAuthenticated()"

That of course means, that this method needs to be added to your header.component.ts file:

isAuthenticated() {

return this.authService.isAuthenticated();

}

So you basically moved the service access from the template to the TypeScript code.

Something similar has to be done for the recipe-edit.component.ts and .html files.

Instead of using

\*ngFor="let ingredientCtrl of recipeForm.get('ingredients').controls; let i = index"

you should use

\*ngFor="let ingredientCtrl of getControls(); let i = index"

in the template.

Again, this getControls() method now needs to be added to the recipe-edit.component.ts file:

getControls() {

return (<FormArray>this.recipeForm.get('ingredients')).controls;

}

The <FormArray> type conversion needs to be added here to avoid compilation errors from TypeScript's side.

You find all these changes included in the optimizations-final.zip project snapshot (which is attached to the last lecture of this module).

