Agular Complete: Notes

9/4/2019

# 1.3 Angular Versioning

A screenshot of a cell phone

Description automatically generated

# 1.4 CLI Deep Dive & Troubleshooting

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:**

**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

**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

**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

# 1.7 Course Structure

A picture containing businesscard, text

Description automatically generated

# 1.11 Where to Find the Source Code

**Do you get some strange error? Are you stuck?**

**Have a look at the source code of this course** - by comparing it to your code you should be able to quickly find out where your code deviates and what causes the issue!

You can find the source code of each section attached to the **last lecture of that section**!

Each ZIP file holds code files which you can use to compare your code to it. Or you execute my projects:

1) Extract

2) npm install in the extracted folder

3) ng serve

Got an error related to TypeScript thereafter?

Run npm install --save-dev-exact typescript@3.4.5 in the same project folder and re-try ng serve.

**Got any other problems with the code**(e.g. error messages when running it or you don't know how to use it)? Check this thread created by Jost: <https://www.udemy.com/the-complete-guide-to-angular-2/learn/lecture/6709112#questions/8079942>

# 2.1 Angular App Load Sequence (vid. 13)

The Angular application load process uses the following files (in sequence) to bootstrap the application.

1. main.ts
2. app.module.ts
3. app.component.ts
4. index.html

## Section Two, Angular Basics:

* Creating Components with the CLI & Nesting Components

ng generate component <component-name>

ng g c <component-name>

* Working with Component Templates
* Understanding the Component Selector

By default, the component selector is created by default when a component is generated by the Angular CLI, in the form of app-name. The selector, however, works like a CSS selector, which is why, by default, the selector, app-name, renders where there's an <app-name> tag. You could use a selector formatted as an attribute selector, [app-name]. In this case then the Angular would render in any tag with the app-name attribute regardless of the tag, e.g., <div app-name>.

You could also use a selector that targets a class or id. You merely format the component selector property in the component's class decorator as a CSS class or id selector, e.g., .app-name or #app-name.

* What is Databinding?

A screenshot of a cell phone

Description automatically generated

* String Interpolation
* Property Binding
* Property Binding vs String Interpolation

You can use string interpolation to bind values to html element

properties in the component template:

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

* Event Binding
* Bind-able 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.

* Passing and Using Data with Event Binding
* The FormsModule and Two-Way Data Binding

Important: For Two-Way-Binding 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';

* Combining all Forms of Databinding
* Understanding Directives

Directives are simply instructions within the DOM. Components are Directives with a Template.

A screenshot of a social media post

Description automatically generated

* Using ngIf for Conditional Output
* Enhancing ngIf with an Else Condition

An Else condition may be combined with an \*ngIf directive through the use of a "local reference."

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

<**ng-template #noServer** >

    <p>Server was created, and the server name is {{serverName}}</p>

</**ng-template**>

* Styling Elements Dynamically with ngStyle

To conditionally add styles to a DOM element, Angular provides the ngStyle directive. The ngStyle directive is set using property binding. The ngStyle attribute on the DOM element is set to an expression that sets styles for the HTML element.

<p **[ngStyle]="{'background-color':getColor()}**"> Server {{serverId}} is {{getServerStatus()}}.</p>

* Applying CSS Classes Dynamically with ngClass

To conditionally add classes to a DOM element, Angular provides the ngClass attribute directive. It takes an object with key-value pairs. The key is the class to apply, and the value is an expression that resolves to a Boolean.

<p **[ngClass]="{online: serverStatus === 'online'}**"> Server {{serverId}} is {{getServerStatus()}}.</p>

* Outputting Lists with ngFor
* Getting the Index with ngFor

# 3. Course Project

## Planning the App

A screenshot of a cell phone

Description automatically generated

### Installing Bootstrap Correctly

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/>

### 3.8 Alternative Non-Collapsible Navigation Bar

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 (see below), or you replace collapse navbar-collapse  with just navbar-default.

Adding a Hamburger Menu:

Alternatively, if you want to make the navigation bar responsive, please replace these lines in header.component.html:

<div class="navbar-header">

<a routerLink="/" class="navbar-brand">Recipe Book</a>

</div>

<div class="collapse navbar-collapse">

with these lines:

<div class="navbar-header">

<button type="button" class="navbar-toggle" (click)="collapsed = !collapsed">

<span class="icon-bar" \*ngFor="let iconBar of [1, 2, 3]"></span>

</button>

<a routerLink="/" class="navbar-brand">Recipe Book</a>

</div>

<div class="navbar-collapse" [class.collapse]="collapsed" (window:resize)="collapsed = true">

and add this line to header.component.ts:

collapsed = true;

# Section 5: Components and Data Binding Deep Dive

## 5.3 Property & Event Binding

A screenshot of a cell phone

Description automatically generated

## 5.12 @ViewChild in Angular 8

In **Angular 8**, the @ViewChild() syntax which you'll see in the next lecture needs to be changed slightly:

Instead of:

**@ViewChild('serverContentInput') serverContentInput: ElementRef;**

use

**@ViewChild('serverContentInput', {static: true}) serverContentInput: ElementRef;**

The same change (add { static: true } as a second argument) needs to be applied to ALL usages of @ViewChild() (and also @ContentChild() which you'll learn about later) If you plan on accessing the selected element inside of ngOnInit().

If you DON'T access the selected element in ngOnInit (but anywhere else in your component), set static: false instead!

This is a temporary adjustment which will NOT be required anymore once Angular 9 is released!

## 5.14 Component Lifecycle

A screenshot of a cell phone

Description automatically generated

## 5.18 @ContentChild() in Angular 8

In **Angular 8**, the @ContentChild() syntax which you'll see in the next lecture needs to be changed slightly:

Instead of:

@ContentChild('contentParagraph') paragraph: ElementRef;

use

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

The same change (add { static: true } as a second argument) needs to be applied to ALL usages of @ContentChild() if you use the selected element inside of ngOnInit (as we do it in the lectures).

If you DON'T use the selected element in ngOnInit, set static: false instead.

## Section 5 Topic Summary

* Splitting Apps into Components
* Property & Event Binding
* Binding Custom Properties
* Assigning an Alias to Custom Properties
* Binding to Custom Events
* Assigning an Alias to Custom Events
* View Encapsulation
* Using Local References in Templates
* Accessing the Template and DOM with @ViewChild Directive
* Projecting Content into Components with ng-content
* Understanding the Component Lifecycle and Lifecycle Hooks
* Accessing ng-Content with the @ContentChild Directive

# Section 7: Directives

### Attribute vs Structural Directives

The key difference between attribute and structural directives is that while both reside in html elements, like attributes, structural directives change the DOM structure around the directive, e.g., ngFor, ngIf, etc.

A screenshot of a cell phone

Description automatically generated

Using the Renderer

abstract class Renderer2 {

abstract data: {...}

destroyNode: ((node: any) => void) | null

abstract destroy(): void

abstract createElement(name: string, namespace?: string): any

abstract createComment(value: string): any

abstract createText(value: string): any

abstract appendChild(parent: any, newChild: any): void

abstract insertBefore(parent: any, newChild: any, refChild: any): void

abstract removeChild(parent: any, oldChild: any, isHostElement?: boolean): void

abstract selectRootElement(selectorOrNode: any, preserveContent?: boolean): any

abstract parentNode(node: any): any

abstract nextSibling(node: any): any

abstract setAttribute(el: any, name: string, value: string, namespace?: string): void

abstract removeAttribute(el: any, name: string, namespace?: string): void

abstract addClass(el: any, name: string): void

abstract removeClass(el: any, name: string): void

abstract setStyle(el: any, style: string, value: any, flags?: RendererStyleFlags2): void

abstract removeStyle(el: any, style: string, flags?: RendererStyleFlags2): void

abstract setProperty(el: any, name: string, value: any): void

abstract setValue(node: any, value: string): void

abstract listen(target: any, eventName: string, callback: (event: any) => boolean | void): () => void

}

## Section 7 Topic Summary

* ngFor and ngIf
* Creating a Basic Attribute Directive
* Using the Renderer to build a Better Attribute Directive
* More about the Render
* Using HostListener and Host Events
* Using HostBinding to bind Host Properties
* Binding to Directive Properties
* What Happens behind the Scenes on Structural Directives
* Building a Structural Directive
* Understanding ngSwitch

# Section 11: Routing

### Section 11.3: Understanding the Example Project

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.

### Section 11.4: Setting up Routes

The simplest way to configure app routes is in the app.module.ts file. Routes require two modules the @angular/router package.

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

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

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

**import { RouterModule, Routes} from '@angular/router';**

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

import { HomeComponent } from './home/home.component';

import { ServersComponent } from './servers/servers.component';

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

import { EditServerComponent } from './servers/edit-server/edit-server.component';

import { UsersComponent } from './users/users.component';

import { UserComponent } from './users/user/user.component';

import { ServersService } from './servers/servers.service';

**const appRoutes: Routes = [**

**{path: '', component: HomeComponent},**

**{path: 'servers', component: ServersComponent},**

**{path: 'users', component: UsersComponent}**

**];**

@NgModule({

declarations: [

AppComponent,

HomeComponent,

ServersComponent,

ServerComponent,

EditServerComponent,

UsersComponent,

UserComponent

],

imports: [

BrowserModule,

FormsModule,

**RouterModule.forRoot(appRoutes)**

],

providers: [ServersService],

bootstrap: [AppComponent]

})

export class AppModule { }

### Section 11.5: Navigating with Router Links

The router link can be set using with an attribute or property binding. Property binding is used, generally, with more complex route paths.

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

<ul class="nav nav-tabs">

<li role="presentation" class="active"><a **routerLink="/"**>Home</a></li>

<li role="presentation"><a **routerLink="/servers"**>Servers</a></li>

<li role="presentation"><a **[routerLink]="['/users']"** >Users</a></li>

</ul>

</div>

### Section 11.6: Understanding navigation Paths

Router paths can be in two forms: Absolute and Relative.

Absolute path example:

<li role="presentation"><a **routerLink="/servers"**>Servers</a></li>

Relative path examples:

<li role="presentation"><a **routerLink="servers"**>Servers</a></li>

<li role="presentation"><a **routerLink="./servers"**>Servers</a></li>

<li role="presentation"><a **routerLink="../../servers"**>Servers</a></li>

### Section 11.7: Styling Active Router Links

Use the routerLinkActive directive to set a class on an active-link element. Home routes (‘/’) require an additional directive, routerLinkActiveOptions, to require exact matching of routes, else the home route will always get the active class because the path is always a sub-string of any active path:

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

<ul class="nav nav-tabs">

<li role="presentation" **routerLinkActive="active" [routerLinkActiveOptions]="{exact: true}"**><a routerLink="/">Home</a></li>

<li role="presentation" **routerLinkActive="active"**><a routerLink="/servers">Servers</a></li>

<li role="presentation" **routerLinkActive="active"**><a [routerLink]="['/users']">Users</a></li>

</ul>

</div>

### Section 11.8: Navigating Programmatically

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

**import { Router } from '@angular/router';**

@Component({

selector: 'app-home',

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

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

})

export class HomeComponent implements OnInit {

constructor(**private router: Router**) { }

ngOnInit() {

}

onLoadServers() {

**this.router.navigate(['/servers']);**

}

}

### Section 11.9: Using Relative Paths is Programmatic Navigation

The navigate() method of the router is unaware of the current route, so route paths are absolute unlike when using the routerLink directive of an anchor tag. To provide a reference to the navigate() method, the relativeTo parameter is passed, providing the currently activated route.

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

import { ServersService } from './servers.service';

**import { Router, ActivatedRoute } from '@angular/router';**

@Component({

selector: 'app-servers',

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

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

})

export class ServersComponent implements OnInit {

private servers: {id: number, name: string, status: string}[] = [];

constructor(private serversService: ServersService,

**private router: Router,**

**private route: ActivatedRoute**) { }

ngOnInit() {

this.servers = this.serversService.getServers();

}

onReloadPage() {

**this.router.navigate(['../servers'], {relativeTo: this.route});**

}

}

### Section 11.10: Passing Parameters to Routes

const appRoutes: Routes = [

{path: '', component: HomeComponent},

{path: 'servers', component: ServersComponent},

{path: 'users', component: UsersComponent},

**{path: 'users/:id', component: UserComponent}**

];

### Section 11.11: Fetching Route Parameters

The route parameters are available through the snapshot.params property of the ActivatedRoute object

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

**import { ActivatedRoute } from '@angular/router';**

@Component({

selector: 'app-user',

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

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

})

export class UserComponent implements OnInit {

user: {id: number, name: string};

constructor(**private route: ActivatedRoute**) { }

ngOnInit() {

**this.user = {**

**id: this.route.snapshot.params['id'],**

**name: this.route.snapshot.params['name']**

**};**

}

}

### Section 11.12: Fetching Route Parameters Reactively

properties set by the ActivatedRoute.snapshot object are not updated if the route path parameters change if the component is re-loaded. To react to any change in the parameters, you need to subscribe to the ActivatedRoute.params observable.

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

**import { ActivatedRoute, Params } from '@angular/router';**

@Component({

selector: 'app-user',

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

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

})

export class UserComponent implements OnInit {

user: {id: number, name: string};

constructor(**private route: ActivatedRoute**) { }

ngOnInit() {

this.user = {

id: this.route.snapshot.params['id'],

name: this.route.snapshot.params['name']

};

**this.route.params.subscribe(**

**(params: Params) => {**

**this.user.id = params.id;**

**this.user.name = params.name;**

**}**

**);**

}

}

### Section 11.13: Cleaning up Routing Observables

Because an observable is not “closely tied” to the component, its instance lives on in memory even after the component is destroyed. In the case of the ActivatedRoute.params observable, Angular does garbage collection, so it’s not necessary to explicitly destroy the observable instance. It may be good practice, however, to do so anyhow.

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

import { ActivatedRoute, Params } from '@angular/router';

**import { Subscription } from 'rxjs';**

@Component({

selector: 'app-user',

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

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

})

export class UserComponent implements OnInit, **OnDestroy** {

user: {id: number, name: string};

**paramsSubscription: Subscription;**

constructor(private route: ActivatedRoute) { }

ngOnInit() {

this.user = {

id: this.route.snapshot.params['id'],

name: this.route.snapshot.params['name']

};

**this.paramsSubscription =** this.route.params.subscribe(

(params: Params) => {

this.user.id = params.id;

this.user.name = params.name;

}

);

}

**ngOnDestroy() {**

**this.paramsSubscription.unsubscribe();**

**}**

}

### Section 11.14: Passing Query Parameters and Fragments

Angular provides additional properties with the routerLink directive to allow for passing of query parameters and hash fragments. Although they look like directives themselves, they are not. They’re merely bindable properties of the routerLink directive.

<div class="col-xs-12 col-sm-4">

<div class="list-group">

<a

[routerLink]="['/servers', 5, 'edit']"

**[queryParams]="{allowEdit:'1'}"**

**[fragment]="'loading'"**

href="#"

class="list-group-item"

\*ngFor="let server of servers">

{{ server.name }}

</a>

</div>

</div>

These properties are also available when routing programmatically, using the Router.navigate() method:

onLoadServer(id: number) {

this.router.navigate(['/servers', id, 'edit'], **{queryParams: {allowEdit: '1'}, fragment: 'loading'}**);

}

### Section 11.15: Retrieving Query Parameters and Fragments

Query parameters and fragments are available through the ActivatedRoute, through the snapshot of the queryParams and fragment observables.

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

import { ServersService } from '../servers.service';

**import { ActivatedRoute, Params } from '@angular/router';**

@Component({

selector: 'app-edit-server',

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

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

})

export class EditServerComponent implements OnInit {

server: {id: number, name: string, status: string};

serverName = '';

serverStatus = '';

constructor(private serversService: ServersService,

**private route: ActivatedRoute**) { }

ngOnInit() {

**console.log(this.route.snapshot.queryParams);**

**console.log(this.route.snapshot.fragment);**

**this.route.queryParams.subscribe((params: Params) => {**

**console.log(params);**

**});**

**this.route.fragment.subscribe((fragment: string) => {**

**console.log(fragment);**

**});**

this.server = this.serversService.getServer(1);

this.serverName = this.server.name;

this.serverStatus = this.server.status;

}

onUpdateServer() {

this.serversService.updateServer(this.server.id, {name: this.serverName, status: this.serverStatus});

}

### Section 11.17: Setting up Child (Nested) Routes

Setting up child (or nested routes) is done with the use of the “children” property in an existing route. The router-outlet directive is then included in the parent component which instructs angular where to render the child components

const appRoutes: Routes = [

{path: '', component: HomeComponent},

{path: 'users', component: UsersComponent, **children**: **[**

**{path: ':id/:name', component: UserComponent}**

**]},**

{path: 'servers', component: ServersComponent, **children: [**

**{path: ':id', component: ServerComponent},**

**{path: ':id/edit', component: EditServerComponent}**

**]**}

];