ANGULAR 8

Angular is a Js Framework which allows you to create “Single-Page-Applications” ( SPA’s)

Angular changes the DOM and run it and it’s a single page. One single HTML page, which changes.

#### CLI DEEP DIVE AND TROUBLESHOOTING

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

If the CLI prompts you to “answer some questions” ( some version do that ) you can simple hit enter for all questions this will accep thte default settings which are fine for this course.

The CLI generates a different welcome screens than youre going to see in my video though. No worries youll still be able to follow along without issues! Just make sure to code alonw so that your code equals mine Angular itself didn’t change a bit

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

You encountered issues during the installation of the CLI or set up of a new project?

A lot of problems are solved by making sure youre using the lasts version of Node js and npm and the CLI itself

Updating NOdeJS:

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

Updating npm:

Run npm install –g npm

Updating the CLI

Npm uninstall –g angular-cli@angular/cli

Npm cache clean

Npm install –g@angular/cli

Here are some common issues and solutions:

-Creation of new project takes forever ( longer than 3 minutes )

- that happens on window 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 youre using the lastest CLI version and try restarting your cli

# ANGULAR CLI

Run these on the command line

-npm install –g@angular/cli

-ng new my-dream-app

-cd my-dream-app

-ng serve

Download the latest version of Node js before you run the above command lines.

It will create a folder with the app name and use default setting by hitting NO for angular routing and CSS

Open the IDE and the folder will be “my-first-angular-app”

Angular is, of course not a tool to allow us to write static HTML files. We wouldn’t need a framework for that. It allows us to mix static HTML code and actually what we have here is one of these componenets Angular works with; the app component

We also see title = 'app'. In “app.component.ts”

Now, what's that?

If you go back to the app.component.html file, we also saw title here right; between the curly braces.

So an assumption would be that the title in the Typescript file is related to that.

So let's maybe change this to 'my app' and now if we save this it recompiles and if we go back, we see

'Hi, this my app!'. Now this might not be perfect English, but we can definitely see that something changed.

This is so-called data binding in action;

a concept we'll also dive into deeply in this course. This is how we can output dynamic

content.

This could of course be content that is calculated dynamically or retrieved from a server in our

HTML code.

Now interestingly, if we right-click on the loaded page and inspect the page source we don't actually see

that code there.

We just see a bunch of script imports at the bottom. That is our build code and the Angular framework

code, the head tag here

and then this strange app-root part. Now, app-root is also something we see in the app.component.ts

file here in the selector.

So it looks like this is basically like our own HTML tag we're creating

and that's indeed what's happening here.

The page we're viewing here, to which this source belongs,

actually is this index.html file in the src folder. Here we also see app-root. The script imports

are missing because they are injected dynamically.

Now, what Angular does in the the end is

always load this page.

Then we have these dynamically-injected script imports and these script imports will dynamically replace

app-root with our own component.

And of course we can have more than one component in Angular apps.

Again, these are all things we'll dive into in this course.

Now that's nice, but

let's do something more fancy.

Let's change title dynamically.

So back in the app component, I'll actually get rid of the entire code in here and I'll add an input instead;

a normal HTML input (type="text"). Below that I'll add a paragraph where I want to output, let's say,

a name.

Now, name is not something we've defined yet, we'll do so in a second.

Let's go to app.component.ts and

change title to name and I'll say Max here (because that is my name) and if I save that and go back,

we see the input and we see the name.

Now I want to be able to enter something in the input and automatically change the name. We can do

this with a tool provided by Angular; a so-called directive which is called ngModel.

Now, you'll learn the exact syntax throughout the course.

For now, let's simply add [(ngModel)],

written like that.

Make sure to get the casing correct.

There is no dash in between or anything like that.

So ngModel in square brackets and parentheses on the input and set it equal to name.

So, to the same name you're outputting here.

Now this is a so-called directive and what it does is it basically tells Angular to listen to anything

you enter here and store it in this name property, in this name model, but also on the other hand, output

the value of the name model in this input.

This is what we're doing here. Now,

if we save this we don't see anything on the page and if we open the developer tools, which you can simply

do from the menu or with the shortcut. You reach it under View ->

Developer -> Developer Tools (in Chrome) and I strongly recommend using Chrome for this course since it has

the best developer tools.

So, then we see here in the console-part there is an error.

Can't bind to 'ngModel' since it isn't a known property of 'input'.

So somehow Angular doesn't understand ngModel.

Now that's strange, because as I said it's built-in right? Angular is actually split up into multiple modules;

sub-packages you could say.

We need to add them if you want to use a certain feature from them.

And in this course you will learn about the different packages and which features they contain. To add

such a feature, we go to another file that we haven't had a look at yet;

the app.module.ts file. This is basically where we tell Angular which pieces belong to our app and there we have

to add something to imports to import another package from Angular.

So we need to import it at the top of the file first because Typescript always needs to know where things

are.

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

This is one package from the Angular framework and from that package we can import the Forms Module; written

like this: import { FormsModule } from '@angular/forms';

So make sure to add this import.

This is not an Angular feature; it's a Typescript feature.

As I said Typescript needs to know where things are.

But now with that imported we can add it to this imports array down here (which is not related to the

imports up there).

This is simply a feature understood by Angular since it's part of an Angular module.

And now it tells Angular that we want to import some form features.

And this directive we're using (ngModel) is such a form feature.

So now if we save that, our app will rebuild and once it's done, we can go back and see our input

with Max inside of it.

And if I start typing there, you'll see that the text below it automatically updates.

This is some magic done by ngModel.

We'll dive much deeper into everything we learned here.

But this is now our first app; with some changes.

What does this course offer you? We're just **getting started** and we just built and edited our first Angular

application.

But of course we didn't fully understand what we did there.

So that's why in the next lecture we're going to dive into ***the basics*** of Angular. What are *components*?

What did we do there with this two-way data binding and how does that all work.

This section will answer it and we'll take a very detailed look at it. We'll also explain how all

these files you saw in your project are connected to each other.

Then we're going to look at ***components*** and data binding; two important key features and we actually

saw both already in our first application. Angular apps are built up from components and data binding

is simply how you output data in your DOM in the end and, as you will also learn, react to user events.

Angular has another key feature; **directives**. ngModel, which we used with two-way data binding, actually

is such a directive. You will learn more about the built-in directives in this section and, also very

important,

build your own directives; nice little helpers (instructions)

you can place in your templates, in your HTML code, which will then do something at runtime depending on

the commands you wrote in there. Well, after having a look at this, we're going to learn more about

**services** and dependency injection;

a core feature of Angular which makes it really easy for you to have your different pieces in your app

communicate with each other, to centralize code and to manage the state of your application.

Once we're finished with that it's time to have a look at routing because, thus far, we will only have

been on one page.

Well, we're always on one page since it's a single page application,

but to the user it really looked like one page. With routing, we introduce the management of different

URLs so that, to the user,

it looks like we're switching pages even though technically, we will still remain on that single page.

Sounds great!

It is

and routing shows you how it works.

Then it's time to have a look at Observables; something which will make more sense once you've been through

the routing section.

It is a concept allowing you to work with asynchronous code. Angular embraces it,

it's really powerful

and this section explains how it works.

Then we'll have a look at forms, because handling forms, handling user input is a key task of almost

any application

and this section takes a very close look at it. After we're done with forms

we're going to have a look at pipes; another nice feature which makes it easy for you to transform the

output

(what you display on the template at runtime). Well, we're nearing the end, but not before having a look

at HTTP.

What if you need to reach out to a web server?

What if you need to store some data in a database?

Angular can't connect to a database directly, but it can connect to a server which is able to and the Http Section

shows just that.

Thereafter, we'll have a look at authentication. What does authentication mean in an Angular application?

How does it work? While we're at it, we'll

implement it in an application. Then we're going to have a look at some optimizations we

can put into place and how we can manage different modules in our application;

something you will really understand once we are at this point. Then, we're going to deploy an application,

learn how we can get our Angular application from our local machine to a place in the Internet where

we can view it.

And if that's not enough for you, we're also going to have a look at animations and testing Angular applications.

Besides that, throughout the whole course, we're going to build a real project putting all these things you'll

learn in the individual sections into practice and, therefore, see how they are used in a real project.

So with that, I feel very confident you're going to be an Angular Master after finishing this course

and I can't wait to start this journey together with you.

HOW TO GET MOST OUT OF IT

-watch the vides and read the transcript

-do the assignments

-do the course project

-use the Q&A section

What is TypeScript

IT offers more features than vaniall JS like classes interfaces and very important which gives it the name types: strong typing

So you define in Typescript if a certain variable is a number a string or something else.

You don’t do this in vaniall js. There you have a dynamic typing

You can have a string variable and then you can assign a number that’s totally fine. That wont work in Typescript. It will give you an error and therefore it allows you to write much more robust code which gets check at the time you write it and not just at the time you run it. Typescript doesn’t run in the browser so it is compiled to JS. At the end of the course you can go through the Typescript.

BASIC PROJECT SETUP

So this is the page as we see it right now in the project we created at the end of the last section,

an empty page here, Angular, The Complete Guide, we reach it at localhost:4200, this is where our development server hosted by the CLI or spun up by the CLI will host our Angular application.

Now the content we see here can be changed by going to the ***app.component.html*** of file in our app folder

and in here, we can output some text let's say I'm the app component here in a h3 tag, a normal

HTML tag therefore. If we save this, we will see it here and it automatically has the bootstrap text style and so on because we added bootstrap at the end of the last section too.

But whilst it might seem kind of obvious that we see it here, it is also kind of strange, how does our browser or how does the server hosting our app know that it should render the content of app.component.html here?

You could argue it's the only component we have right now and we will take a closer look at component

Soon but that is not the reason and actually, this is not the file served by the server, instead the index.html file here is served by the server and remember that I told you that Angular is a framework which allows you to create single page application, this is the single page which is served, the index.html file.

Now if you have a look at this file, we see this is a normal HTML file, we define the title we could see on the tab in the browser here and the body of this file is very interesting. Here we get this app-root (index.html file) thing with loading.... in between. Now clearly we don't see loading... here( on the local hos), so somehow this index.html file seems to have changed and it did, app-root here here of course is not a default HTML element, instead this is one of our own components. We will soon dive into how we create our own components but the CLI created one for us, the root component of our application, the component which will tie together our whole application in the end and all the files in the app folder here which have component in their name, so these files are related to this component.

I will go into more detail about these files in one of the next lectures, for now let's take a closer

look at the app.component.ts file, the TypeScript file here. Here you can see that we have this @component decorator, this seems to be important but more importantly right now, you'll see that there, we have this selector property which assigns a string as a value and this string holds

app-root. Now this clearly is the same text as in our index.html file and this actually is the information Angular needed to be able to replace this part here in this index.html file with the template of this component, the component having this selector and the template on this component simply is the content here in this app.component.html file. Now whilst I will dive deeper into how to create components and how to configure them, this is what basically happens at the startup, though the missing information is how is Angular triggered, how is it kicked off to actually run over our body here of this index.html file? And the answer is in the final index.html file, getting served in the browser and we can verify this by inspecting the source code here, we got a couple of script imports at the end. These are injected by the CLI automatically, so that is why we don't see it here in the raw index.html file, here we don't have any script imports but whenever this ng serve process rebuilds our project, it will create bundles, Javascript script bundles and automatically add the right imports in the index.html file, a little convenience functionality for us. So in the final file, these script imports here are present and these script imports will contain our own code too. So these script files are therefore executed and they're actually the first code to be executed and that is just something you have to keep in mind, is the code we write in our main.ts file, that is why it's called main, this is the first code which gets executed. Let's have a closer look at it then. Here you see that we get a couple of imports, then we check if we are in production mode or not

to basically turn off some warning messages I can tell you that but most importantly here, this line, this now bootstraps starts our Angular application by passing an app module to this method and app module refers to this file here. Now if we have a look at this file, here we actually see that somehow we kind of like with the component have this strange @ thing here, I will dive into this in a couple of lectures too but most importantly for now, here we get this bootstrap array which basically lists all the components which should be known to Angular at the point of time it analyzes our index.html file and here the circle closes

because here, we reference our app component. So Angular gets started, this main.ts file gets started, there we bootstrap an Angular application and we pass this module as an argument. In this module, we tell Angular hey there is this app component which you know when you try to start yourself and Angular

now analyzes the app component, reads the set up we pass here and therefore knows this selector, app-root and now Angular is able to handle app-root in the index.html file and it knows all right this is the selector I know, you told me that I should know it because it was listed in this bootstrap array in the app module, this component. So now I know that here I should insert the app component and the app component happens to have some HTML code, a template attached to it which is this h3 tag and this is how the Angular application starts here, this is why we see what we see. Now that was a load of information regarding the app startup but it's key to understand how this starts, with this information we can now proceed and dive deeper into how components actually work, what they are, how we can create our own ones and what this app module here does in detail.

So let's move on.

#### COMPONENTS ARE IMPORTANT



In the last lecture, we had a close look at what happens at the point of time we visit localhost:4200

here in the browser. We understood that the index.html file is served, that it will contain a bunch of scripts here which get executed which then basically start the Angular app, the Angular app gets the important information, that it should know, the app component that it should analyze it with that information the Angular code is able to parse this here, this app-root component here understand it and insert our Angular application at this point and that is why we don't see loading... which would still be in the source code of this page as you can see but instead why we see this because Angular overwrites this at runtime because that is what it does, what its job is.

So now that we had a close look at this startup process, let's have a closer look at this component and

or @components in general. Components are a key feature in Angular, you build your whole application by composing it from a couple of components which you create on your own. Now we do start with this app component, the root component you should say which holds our entire application basically in the end. So this root component, this app component will be the component where we later nest or add our other components too, so to this template, this HTML file of the app component, this is where we will later add our other components. Typically, we might have a web page like this, with a header, with home and users, then a main area and a side bar maybe. Now if we have a look, this would be a fitting component, the header and maybe also the items on the header could be their own component, could be, you don't have to, you will learn throughout the course how I approach the splitting up of an application into separate components. We also have of course our main area and the side bar here as components and that is the core idea behind Angular, build components, each component has its own template, its own HTML code, maybe its own styling and more importantly also, its own business logic and this is the great benefit. It allows you to split up your complex application, your complex webpage into reusable parts, you may use a component more than once and that allows you to easily replicate that business logic, replicate that styling or in general, make a finely controlled piece in your application without having to crunch everything into one single script file, one single HTML file, instead it's very easy to update, very easy to exchange and again re-usable. So let's have a closer look at how we actually create a component and what this @component thing does in the next lecture.

#### CREATING NEW COMPONENT

Components are important right, that is why we start with one, with the app component. It on the one hand is a normal Angular component but on the other hand, it's also kind of special because it serves as our root component, it's in the end listed here in the app module in this bootstrap array which tells Angular, hey this is a special component, you should bootstrap the whole application with that component being the root component. So all other components we create will not be added to the index.html file, there selectors will not be added here, there selectors will be added to the app.component.html file because this is now the root component of our app where we add the other parts. Enough talking about adding components, let's add one. Let's say we want to output some information about a server. We're building a back-end for our server management application and we want to output some server information. So I will store this in a new folder which is a subfolder of the app folder because generally in a Angular CLI project, all your app related content will go into this app folder. I will name it server because it will hold my ServerComponent and this is a good practice, having the folder name equal your component name and each component typically should have its own folder, though it's not a hard rule but generally, it makes sense. So here in the server folder, I will add my ServerComponent. Now a good naming convention is to have server, so the name of your component first, then a dot and then component, we will later learn that there are other building blocks in an Angular application too, so by using filenames like this, it's very easy to see what's inside the file. And very important, don't forget to add .ts, so for TypeScript because this is what we will write this component in. So now we get an empty file for our new component, how do we create a component now? First of all, a component simply is just a class, a TypeScript class, so that Angular is able to instantiate it to create objects based on the blueprint we set up here you could say. So let's export this class so that we can use it outside of the file too because as I mentioned before, we're going to use our component in the app component for example to add it there. So we will add a class here and this class should now be named ServerComponent. Again the naming convention here, have the name of the component first, server and then also the description, component because later we will also learn about other parts which might then be named differently in the Angular app. This is our class and right now, it's a normal TypeScript class, nothing special about it, we can't use it like this, Angular doesn't have all the information it requires. So we should add something to it which tells Angular that this is not only a normal TypeScript class but instead something special, a component. We do this by adding a special decorator, decorators are a TypeScript feature which allow you to enhance your classes for example, enhance elements you use in your code, it's not restricted to classes but here we will use a class decorator. It's the component decorator and decorators are always attached by adding an @ sign in front of them. Now this component decorator is not something TypeScript knows from the start, so we have to import it. We have to add an import and this import now needs to give us access to component, so we need to import component between curly braces because from the file where we're going to import this, we could import other parts too, so we have to specifically pick the component by using this syntax and then it's from and now the package where we import component from is @angular/core. Now Angular ships with a couple of packages where it basically groups its functionalities and the core

package as the name implies gives us access to some of the core functionalities of Angular. So with that, we imported component, now this @component decorator is known to TypeScript, so when it parses this file and compiles it to Javascript, it is able to understand it, now we need to pass a Javascript object to this component decorator to configure it because without any configuration, it's still not that valuable to Angular. But here, we can set up some important information which will be stored as metadata for this class in the background then which will tell Angular what to do with this class and one important information piece is the selector. So basically, the HTML tag by which you're able to use this component later in your other components templates, we will see this in action soon. The selector should be a string

and here you may set up any name you want but you should make sure that it is a unique selector, that you don't accidentally overwrite a default HTML element or something like this. So typically, you prefix it with app- and then a fitting name, like server because it's a ServerComponent. This is my own selector by which I can now later use this component in my other components HTML files. So with that set up, the other important piece we need to have is the template and here let's reference another external file, I'll come back to another alternative to this soon and this external file of course needs to be created. So in the same folder, I will create a server.component and now .html file, this will hold the template, the HTML code of my component here. So here, I can now basically add any content I want,

like for example the Server Component to start simple and back to the TypeScript file, I can now point to this HTML file. We need a relative path for this, in the end this will all get bundled by webpack, so we need to tell it where to find the HTML file. Well and relative to the TypeScript file, the path simply is ./server.component.html, like this, this is pointing to this file. With this we created our first component,

now to use it we need to dive into app module and understand what this does because we need to change something here to use our own component.

Let's do this in the next lecture.

#### UNDERSTANDING THE ROL OF APPMODULE AND COMPONENT DECLARATION

In the last lecture we created our own component, the server component here. It's almost ready to be used but to use it, we need to change something about our app module, what is an app module? Angular uses components to build web pages and uses modules to basically bundle different pieces, for example components of your app into packages. Now this is kind of an advanced feature but later in the course, I will have a whole section about modules. For the majority of this course, we will only use the app module which is absolutely fine for a lot of projects, only in bigger projects you might think about splitting up your app into multiple modules. But what does this app module do, what is a module then? Well it's as I said bundle of functionalities of our app and it basically gives Angular the information which features does my app have and use. As you can see, it also is just an empty TypeScript class, like our component and as the component, we transform it into something else by adding a decorator, here it's the @NgModule decorator which is also imported from @angular/core. Now in there, we see four properties we set up on the object we passed to @NgModule, declarations, imports providers and bootstrap. We had a look at bootstrap, this was responsible for telling Angular hey which component should you be aware of at the point of time the whole application starts, so which component would you basically recognize in the index.html file and that was the app component. Now I already mentioned that we won't add any more component selectors to the index.html file, that's just not how it works, therefore we won't touch the bootstrap array. Still, we added a new component to Angular and this new component now has to be registered here in the 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 don't tell it that the server component exists, it doesn't know it, just creating the file is not enough. That is why we have to register it here in the @NgModule to tell Angular hey part of this module and therefore right here of our whole app since we only have this module is the server component. We do register new components in this declarations array, we already registered the app component here because even though it's also used in bootstrap, it also is a part of our app, now we also need to add the server component here. Adding it like this is not enough though because now Angular would kind of know that this is part of the app but TypeScript wouldn't know where to find this type, where to find this server component. So when we try to bundle our code, we would get an error which you can actually see if you save this, triggering the recompilation, you see that we get an error in the terminal down there, that it can't find server component because and this is not Angular, this is TypeScript, it doesn't know where this file is, where to find this class. So we need to add an import at the top manually pointing to this. So we add server components here as an import from ./server and then here in the server component and you omit the .ts, you don't add .ts to your import, it's just server.component, so the file name without the extension, the extension is added by webpack which bundles our project automatically. So with this and it's just so important,

this is not Angular, this is a TypeScript feature, now TypeScript knows where to find this, therefore everything can be bundled, so therefore now Angular when it runs knows that we have a server component and that it is part of our app and with this, we can now finally use our component. We will do this in the next lecture but maybe you're also wondering what imports and providers does here in the app module. I'll come back to providers later in the services section, imports simply allows us to add some other modules to this module because I told you, you can split up your app into multiple modules and then you can import these modules to basically make this module a bit leaner and outsource some stuff into another module. Now I told you that we are only going to use one module, so what are we importing here? Some modules built into Angular because Angular itself is split up into modules, so the browser module for example gives us all the base functionality we need to start our app and forms and HTTP module, that is something I'll come back to in the forms or HTTP section of the course. So with this, the basics about this module should be clear, again we'll dive deeper into this later in the course. We also registered our own component, let's now use it in the next lecture.