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