**Angular vs Angular 2 vs Angular 7**

So we now learned what angle is.

Maybe you also came across the many words of angular that you can find there is angular A7 now.

We also had six.

Couple of words to an angular one.

Now angular one was initially released in 2010 and it provided a revolution to the way we can build

front end user interfaces in the browser because it was a huge step forward.

When it came to using browser side javascript to reach rendered a Dom to update the dom at runtime and

therefore provide highly interactive user experiences without reloading the page angular or was a complete

rewrite of Angler one an angle or two was initially released in 2016.

It has nothing in common with angular one except for the core team that developed it.

And therefore if you know angular one unfortunately that knowledge will not help you much.

On the other hand you also don't need to know angular one to get the most out of this course.

Now what's up with angler's six seven and all yabber versions.

Now this entire set on the left here is now referred to as just angular.

It's one in the same framework which totally differs from angular one but which doesn't not differ a

lot within the separate versions.

So it is just angular on the hand angular one is now called angular J.S. to make it clear that these

are two totally separate frameworks now two different words.

We have four angular So four angular two plus are really just incremental improvements.

No complete rewrites.

Not at all to certain texts we use an angle or 7 and therefore in discourse is the same as we used an

angle or two.

There were just some bug fixes minor improvements some new features but generally decent access to same

discourse was updated a couple of times to incorporate these changes and you'll learn the latest angular

syntax in this course.

Therefore because I always and I will do that in the future do update lectures or whole sections or

the whole course even if it is required to meet the latest syntax.

But again it hasn't changed a lot almost hasn't changed at all since angular to an angler JSH or one

as I mentioned is not related to angle or two and it's not part of discourse.

We focus on this part on angler and you'll learn later syntax in this course.

You'll learn Anglo's 7 here but behind the scenes you also learn angler 2.

Because again it's really just the same just that angular sevenths like improved to work.

Of angular 2.

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

Transcript:

So now with the definition of angular and the different word is out of the way let's simply dive into

it and let's build our first angular app for that.

We'll use the official angle or a command line interface Seelye and dad ISDE recommended and best way

of creating angular projects because Angua projects are actually a bit more elaborate regarding their

build workflow.

There are a couple of files that need to be worked out before they can run in the browser and to Seelye

does all of that and also heavily optimized our code so that we ship a highly optimized code word in

to the browser.

Once we finally deploy our app which will all go into this course and the end of course.

So the angler's Seelye and you can see all the steps you need to run on the left here.

And these steps are the same for Windows and Mac.

Now for them to work you need one additional tool and that's note J.S. note.

J.S. is a server side language and you might be wondering what are we doing with it in this course.

It's saying a workhorse not a node course and you'd be right we won't write any note as code here but

note JS will be used behind the scenes by the Seelye to bundle and optimize our project and we'll use

NPM the node package manager to manage two different dependencies and angle or project has dependencies

are things like the angular framework itself but also some of our libraries that FREYBERG uses.

So you can simply download the latest word here by clicking on that button.

In my case it's 10.0 but that might differ for you and you will get an installer for which you can walk

which will conveniently install notes.

Yes.

And does NPM tool on your machine once you you've finished installation we can run these commands and

we run them in our terminal or command prompt.

Now here I'm on a Mac Therefore I used an old terminal.

If you were on windows you would open a normal command prompt possibly open that as an administrator

by right clicking on the executable and clicking run as administrator to rule out some potential issues.

But in the end you should be able to simply run NPM using that note package aventure install dash g

to install it globally on your machine at angular slash Seelye at latest at latest.

It's optional at angular slash Selye is not.

So you absolutely need to run this now on Windows.

This should just work on Mac or Linux you probably need to add a pseudo in front of this to give yourself

the right permissions.

Hit enter and now I'm prompted to enter my password.

You may or may not be.

And once you confirm everything this should now download the angular Selye from this node package manager

repository and install it on your machine.

Now as you can see you might get some errors in-between like I'm getting here.

You may ignore them as long as it's successfully installed TC lie as you can tell by the output here

at the bottom.

So it added to or it updated two packages or added them if you installed it the first time and you CDC

and the Word installed.

Now the word can differ depending on the time when you install it as by the way the general functionality

will not differ.

Now the wording can differ depending on the time when you installed it by way the general functionality

will not differ.

So this did work though with that we can create our first project and we do this with this engine you

command here which is now available since we installed DCL I forget.

Navigate into a folder where you want to create that project and once you navigated there with the cd

command you can simply run Engy new and then maybe my dash for a stash app.

This name is totally up to you.

So N-G new is a set command to create a new project.

The name is up to you.

It must not be test or something like that though because that actually is a researched word which will

work.

Now depending on the word you're using you might be asked two questions here a couple of questions.

The first one is would you like to add angular routing.

Now you can just type N here or hit enter to use the default which is also no.

And then here which Stahl's You'd for me you want to use and there you also want to go with the default

for discours which is already selected so you can hit enter here too.

So essentially the result of this hit Enter two times and now will create a new project.

And now this will create a new folder with a couple of files and dependencies and the entire build workflow

setup in there.

And as I said we need as more complex set up because for example angular uses typescript a superset

of Javascript.

So basically a language that looks a bit like javascript and is compiled down to Javascript in the end.

Bited workflow but which also offer some extra features and to do that compilation as well as a couple

of other optimization steps.

We need this more complex set up with all these dependencies.

Now once it is finished successfully and if it didn't check out the lecture prior to this video here

I do share some common gotchas and fixes there.

So once it is finished successfully we can navigate into that with the cd command and then the name

of the project in my case it's my dash for a stash app.

And once you're in there you can run Engy serve to basically bring up a development server that will

run you are built and for development optimized app so that you can see it in the browser.

The server runs on localhost hundred.

By default you'll see the output here.

So you can simply go to the browser enter localhost four thousand two hundred.

And you should see something like this.

Is this just a simple starting screen provided by the Hasil AI project.

Welcome to app.

And then some useful links to the official docs and so on.

Awesome.

Now this is our first application.

Now we set this up.

Of course not too much is happening here so let's edit this first step and do something more fancy with

it.

**Editing the First App**

Transcript:

In the last lecture, we created our first Angular app but we haven't changed anything there.

So it's time to do that. For that you need an IDE or editor where you can write and edit your code.

Now I'm using WebStorm here. Webstorm is a great IDE,

great for Angular development; but it's not free. A great free alternative would be Visual Studio Code which

you can find on https://code.visualstudio.com/

This is a great IDE and it is free.

You can simply download and install it both for Mac and Windows

and then you can start developing Angular with that. Whichever IDE or editor you install,

you'll always have to open a new folder (in WebStorm's case it's just called Open) and then simply navigate

to the folder you created.

So in my case here it's my-first-app.

Select that folder and open it and now it will load that folder and all the files into the ID on your

system.

So in my case, it now looks like that.

Now this is the project loaded into the ID.

It's still indexing it here but it'll be done in a second.

These are all the folders and files

The Angular CLI created for you. This is your entire Angular project.

Now as I said, this might look intimidating because you've got so many files in there.

Most of these files are just doing some configuration work and you don't really need to touch them.

One interesting file is the package.json file.

Here you can see all the dependencies of your project like Angular 6 and these are third-party

packages

your project needs to run correctly.

All devDependencies are only required for development;

for this build workflow I was talking about. But we're here to edit our code right?

So let's jump into the src folder because that is where our code is.

e2e is for end-to-end testing (we'll ignore this) and node\_modules is where all these dependencies you

see in the package.json file actually were installed.

But we're interested in the source code so let's go to src.

Then we got a bunch of other configuration files and

then here, we've got this app folder and in this app folder we see some other files.

Of course I'll dive into what they all do in detail in this course.

For now let's open the app.component.html file.

Here we indeed see something which looks deceivingly like what we saw in the browser.

Do you remember? It's Welcome to app! and so on.

We see that here too. By the way,

make sure you keep the localhost:4200 process running

(the ng serve process you started here in the terminal).

Make sure to keep that running, but if you're done with development for the day

You can quit it with Ctrl-C and you'll get out of there, but as long as you are developing you should

keep it running because it will automatically watch your files and rebuild your project whenever you

change and save something.

Now since I quit it

I'll need to restart it and I will actually restart it here in my IDE.

In here there is a built in terminal but it's the normal system terminal.

So here I can also run ng serve to bring that back up.

But back to the app.component.html file, we're

outputting something here

and if we change anything here, like,

'Hi, this' (instead of 'Welcome to') and we save it with ng serve running, then if we go back,

you'll see it automatically updated;

Hi, this app!

Now, one strange thing we see is this 'app' here, but

we actually only see these curly braces and title in the app.component.html file and therefore,

we can already see some of the work Angular does here.

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 dynamic things we want to output in that code and actually,

what we have here is one of these components

Angular works with; the app component. A component always has a template, the HTML code, possibly has some

styling in the CSS file (though it's empty here as you can see)

and importantly, it has a Typescript (.ts) file.

If we enter this, this is Typescript and this is now the definition of the component. This is what will

be converted to normal JavaScript by the build workflow.

And in this file, we see a couple of interesting things like @Component. I'll come back to that and what

in detail is happening here in the next module.

We also see title = 'app'.

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.

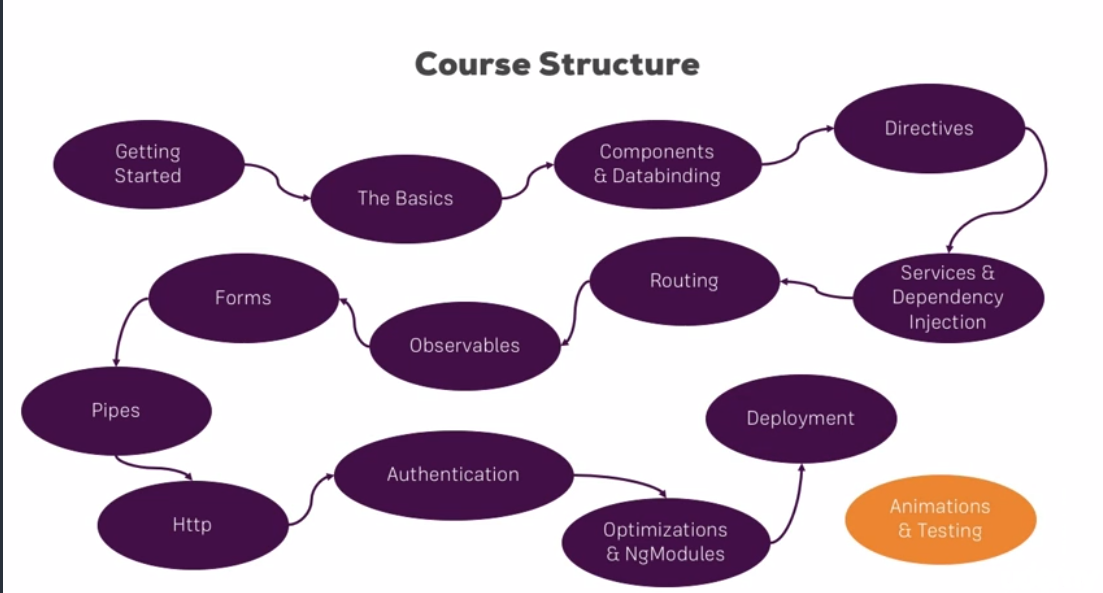
This is some magic done by ngModel.

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

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

**Course Structure**

1. Getting Started
2. Basics
3. Components & Databinding
4. Directives
5. Services & Dependency Injection
6. Routing
7. Observables
8. Forms
9. Pipes
10. Http
11. Authentication
12. Optimizations & NgModules
13. Deployment
14. Animations & Testing



**Transcript:**

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 the Most out of the course**

* Watch the Videos
* Do the Assignments
* Do the Course Project
* Ask in Q&A and answer in Q&A!
* DOCS + Google

**What is Typescript?**

Typescript, really is just a superset of JavaScript.

It offers more features than vanilla JavaScript, 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 vanilla JavaScript.

There, you have dynamic typing.

You can have a string variable and then you can assign a number and that's totally fine.

That won't work in Typescript.

It will give you an error and therefore it allows you to write much more robust code which gets checked at the time you write it; and not just at the time you run it. This is a great enhancement.

However, Typescript doesn't run in the browser, so it is compiled to JavaScript in the end.

This compilation is handled by the CLI; one of the reasons why we need the CLI, why we need a project management tool like the CLI.

Now, this compilation is really fast and therefore, in the end, in the browser JavaScript is going to run.

We're not writing the Angular app in JavaScript though because, whilst technically possible, that wouldn't be much fun.

A lot of the features really only exist in Typescript and Angular is meant to be used together with Typescript.

**Components**

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 and how our entire application basically in the end.

So this route component this app component will be the component where we later nest or add our other components too.

You will learn from the Course how I approach the splitting up of an application into separate components.

Each component has its own template its own age all killed maybe its own styling.

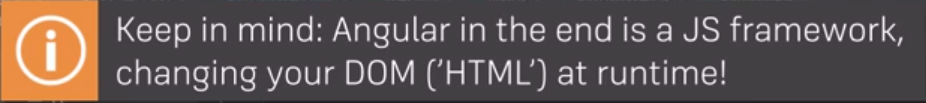
And more importantly also its own business logic and business to great benefit.

It allows us to split up your complex application your complex web page 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 file instead.

It's very easy to update very easy to exchange and again re-usable.



WHAT IS 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.

You know 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 app modules

But what does this app will do what is a module then.

Well it's as I said module is a 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 all is just the empty typescript class like component and asked component we transform it into something else by adding a decorator here.

It's the NgModule decorator which is also imported from an angular core.

Now in there we see four properties we set up on the object we passed to NgModule :

1. Declarations, 2. Imports, 3. Providers and 4. Bootstrap

**Directives**

What are Directives?

**Directives** are the most fundamental unit of Angular applications. As a matter of fact, the most used unit, which is a ***component***, is actually a directive. Components are high-order directives with templates and serve as building blocks of Angular applications.

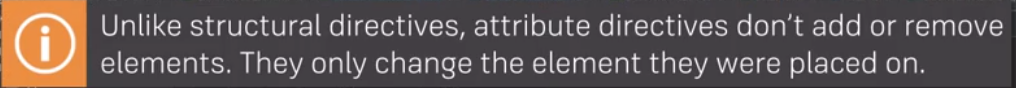
Directives are Instructions in the DOM!

There are three kinds of directives in Angular:

1. Components—directives with a template.
2. Structural directives—change the DOM layout by adding and removing DOM elements.
3. Attribute directives—change the appearance or behavior of an element, component, or another directive.

**Components** are the most common of the three directives. You saw a component for the first time in the [Getting Started](https://angular.io/guide/quickstart).

Structural Directives change the structure of the view. Two examples are [NgFor](https://angular.io/guide/template-syntax" \l "ngFor) and [NgIf](https://angular.io/guide/template-syntax" \l "ngIf). Learn about them in the [Structural Directives](https://angular.io/guide/structural-directives) guide. *Structural Directives* always need to start the prefix ‘\*’ (star) indicating it as a *Structural Directive.*

Attribute directives are used as attributes of elements. The built-in [NgStyle](https://angular.io/guide/template-syntax" \l "ngStyle) directive in the [Template Syntax](https://angular.io/guide/template-syntax) guide, for example, can change several element styles at the same time

NOTE: Directives must be declared in Angular Modules in the same manner as components.