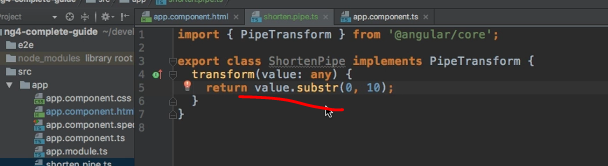
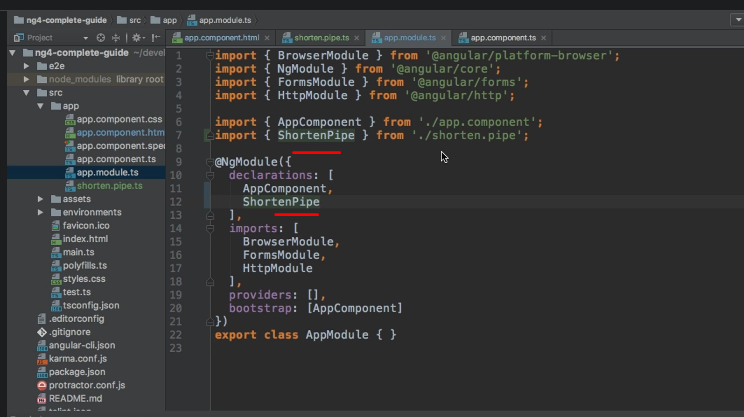
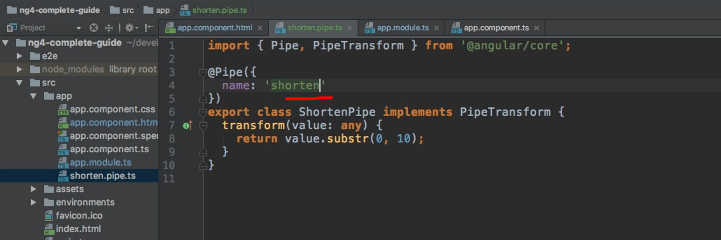
* -: Thus far, we already learned a lot about the built-in pipes or on pipes in general.
* You learned how to use them, where to find out more about them, how to pass parameters or possibly multiple parameters and also, how to chain pipes? This is great, but sometimes, you need some functionality which just isn't built in.
* In this case, you need to create your own pipe.
* And thankfully, this is super simple.
* Now, ***creating our own pipe is as simple as creating a new file in your app folder and then giving it a descriptive name.***
* For example, here I want to create a pipe which **shortens my, my words**, the input, because let's say I want to shorten this name of my server, so the testing environment server somehow gets cut off.
* Now for this, I will create a shortenPipe.
* So I will name the file "shorten.
* pipe.
* ts" to keep this naming convention of being descriptive about what's in a file.
* Now, this newly created TypeScript file should simply hold a class, which I export, of course, which will name "shortenPipe.
* " Now this shortenPipe, this class needs to have one special method to be usable as a pipe.
* And while it's **not strictly necessary,** it's a good practice to implement a certain interface which requires you to implement that method.
* The interface is called "**PipeTransform**" and you need to import it from "@angular/core.
* " So make sure to add this import at the top, "pipeTransform from @angular/core.
* " And with that implemented, you see that I now get an error in my IDE.
* You would also get an error if you now try to compile this, because what we need to do here is, we need to apply or we need to provide the **transform** method.



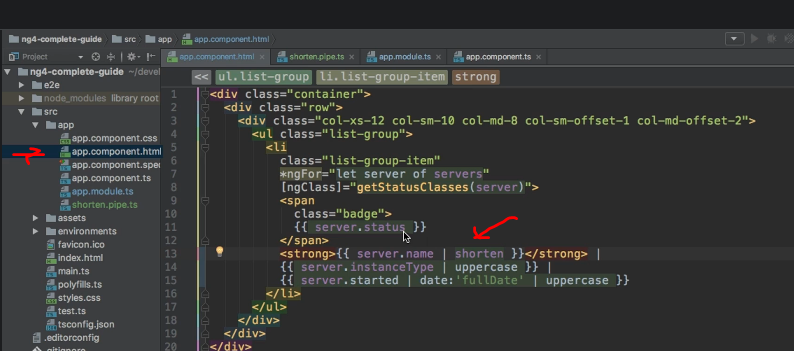
* Now, adding the transform method like this still is not 100% correct.
* Transform needs to receive the value which should get transformed.
* So we need to get the value here, which is of type "any.
* " And then, we would receive a list of arguments.
* Now, for now, our pipe here doesn't take any arguments, so I will omit any other arguments here in the transform method.
* So we only receive the value.
* Now, to shorten this value, a quick and easy way to do this would be to return, and this is important.
* Transform always needs to return something because a pipe is just, you put something in, you get something out.
* So you need to get something out, otherwise it won't work.
* So here we want to return a new string, and this new string should be the old value, but somehow shortened.
* And we could simply use the substring method, "Substr", this is a built-in method JavaScript offers, where you can define, how long this string, the substring you want to extract should be.
* So you want to start at index zero at the start of the value.
* And then, let's say we want to have it 10 characters long.
* This would only give us back the first 10 characters and we can see this, if we save it like this, returning the substring of values, so that we only get the first 10 characters.
* **Now, to use this pipe, we need to go to app module and add it to declarations here to shortenPipe.**
* So just like components and directives, you also need to add pipes to declarations.



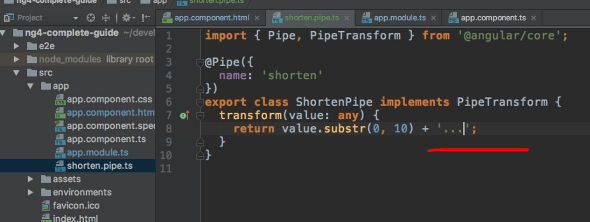
* Make sure to also add the import pointing to the shorten.
* pipe.
* ts file, of course, as always, without the .
* ts at the end here.
* ***Now with this added, we make this pipe available***, and then we can go to our app component, and use it I'd say, ***but how would we use it?*** Well, we have to go back to the shortenPipe first.
* This class is implementing the interface we need to implement, but it lacks one important thing.
* ***We need to add a special decorator, @Pipe decorator which is also important from @angular/core.***
* In the pipe decorator, you can specify the name for the pipe by simply adding name, and then for example, "shorten.



* " Now, by setting it up like this, you can now go to your template, and on the server instance type, which is, excuse me, on the server name.



* Here, we simply add a pipe symbol, and now shorten our own pipe.
* Now, if we do this and save, you see, it gets cut off, and if we count characters, this looks like 10 characters.
* So now our shortenPipe isn't used.
* It's not super pretty though.
* So we can improve our pipe.
* And this is now something which is not really related to pipes.



* I'm simply adding a new string to my shortened string and this will simply be free dots and maybe a blank space in front of it.
* So now, the shortening should look a bit prettier because now, we cut it off, though we always see the free strings.

Table

Description automatically generated with medium confidence

* So another improvement might be to first check if the character, excuse me, if the string is really longer than 10 characters.
* So here, I first check if value length is greater than 10 characters, because only if it is greater, only in this case, I want to shorten it, otherwise I will return the unchanged value.

Text

Description automatically generated

* So now in this case, you see if there's, reloads, it shortens all these names.
* But if we were to remove the server here and the blank space on the first server, so that it is just called "Production", therefore it just has 10 characters.

A picture containing text

Description automatically generated

* You see now, we don't have the free dots here, because we improved our pipe, but this is all pure JavaScript tuning, not really related to pipes.
* The important part about pipes is that you "A" at this pipe decorator, "B", make sure you have the transform method possibly enforced by adding the pipe transform interface, which is a good idea, which is a good practice.
* And "C", by adding your pipe to the declarations array in your app module.
* And with that, you're good to use your own pipe and to create your own pipes and adjust them to your needs.
* Now let's improve this pipe in the next lecture.