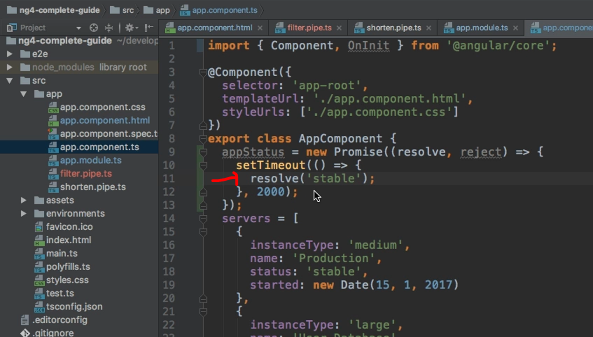
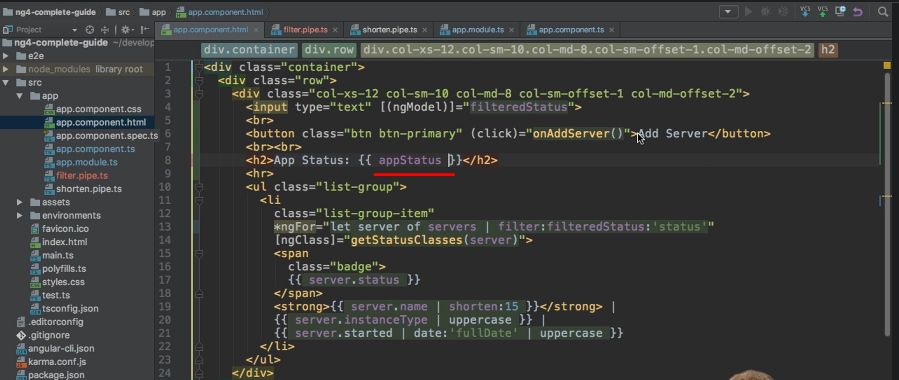
* -: In this module, we learned a lot about pipes,

1. what they are,
2. how to use them,
3. how to parameterize them,
4. how to chain and
5. how to create our own pipes, and
6. how to maybe create an impure pipe forcing update on each data change.

* Now there's one built-in pipe I want to have a closer look at which does something different than all the other pipes.
* ***It helps us with handling asynchronous data.***
* And to demonstrate how this works let's say we have another property right here at the top which says app status, or which holds our app status which could also be offline, critical, or whatever, but here, this will not be a string like offline.
* Instead, this should get loaded after, let's say, two seconds.
* And to simulate this, I will set this equal to a promise.
* And you can imagine this data being returned from a http call from a server, something like this.
* So in this promise, I will initialize this promise with a call back method passed to the constructor where this method itself takes two arguments resolve and reject, the two functions we can execute inside of this promise to resolve or reject the promise.
* And in the promise in this callback function I will then set up a timeout.
* Let's say it triggers after two seconds So after 2000 milliseconds.



* And then this method here will get executed.
* This method, I passed to the set time out method.
* Now here in this function I want to simply set my app status.
* So I want to resolve to, let's say, stable whatever the status should be.
* **So what this will do is it will set app status to stable but only after two seconds.**
* So if we try to output this app status here, let's add a line break, or another line break to have a blank line in between, and then a heading where I say app status.
* And then I try to output app status with string interpolation.

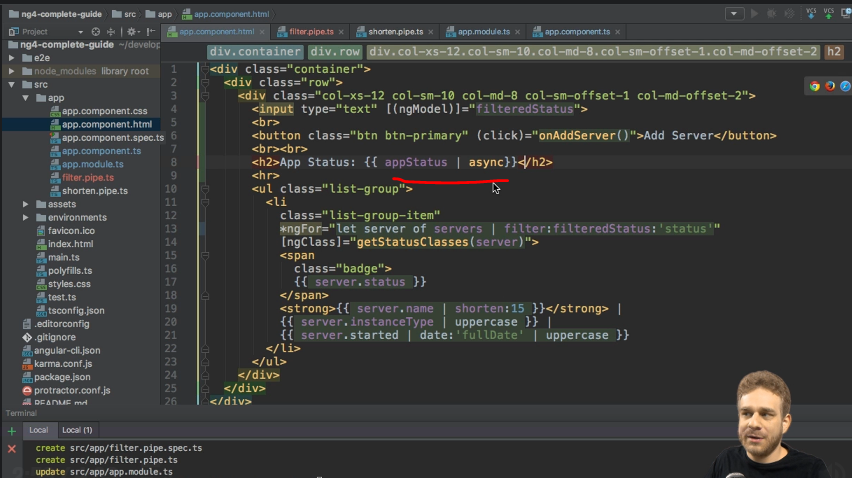


* If we do this, you see object Object.

Graphical user interface

Description automatically generated with medium confidence

* And this is correct because it is our object.
* It's a promise, and a promise is our object.
* But after two seconds, we know that this is no longer an object.
* It now is a string but angler doesn't know because it doesn't watch our object.
* It doesn't see if this object actually transforms to something else, or if this returns as a value.
* It just knows, it's a promise, I'm done.
* And it's good that it behaves like this.
* Saves us performance.
* We should be explicit about how our app behaves.
* But thankfully there is a nice little pipe we can use here to make the transformation of this data easier.
* We know that it will be a string after two seconds and we want to output this string, or it will resolve a string after two seconds, I should say, and we want to output this string so we can add **the pipe symbol and then async.**



* This is a built-in pipe and by adding it, well, watch for yourself.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

* You see, I will reload the app.
* There's nothing there at the beginning but after two seconds you see stable.
* And this is what async does.
* ***It recognizes that this is a promise.***
* ***And as a side note it put all the work with observables there.***
* *It would subscribe automatically, and after two seconds it will simply recognize that something changed, that the promise resolved*, ***Or in the case of an observable that data was sent through the subscription and it will print this data to the screen.***
* And this is the pipe with which I want to conclude this section about pipes.
* Now you should feel really comfortable about using them, building your own pipes, and, as you see here working with async data with pipes.
* Now we will see this async pipe being used in the http section later, but for now let's move on to some exercises regarding pipes.