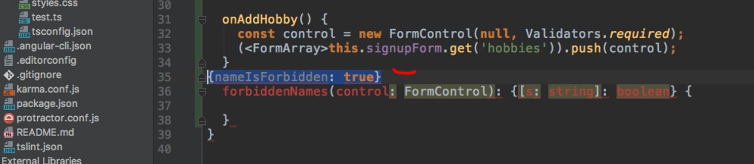
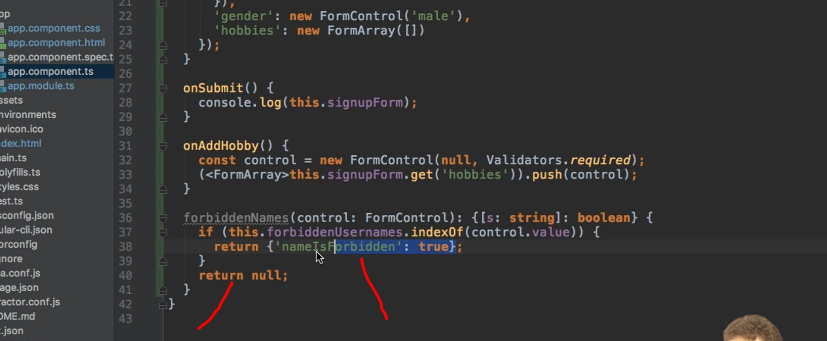
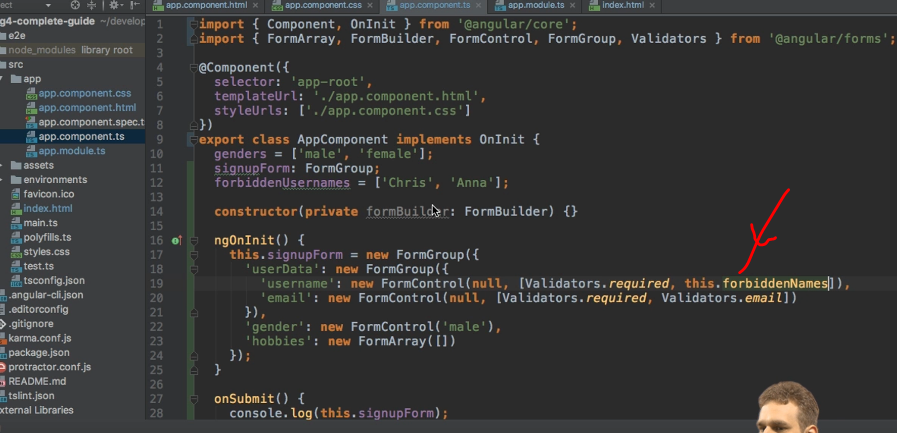
* -: So in the last lectures we learned a lot about the creation and configuration and usage of our own forms.
* Now let's use another cool feature we can easily implement when using the reactive approach.
* **Adding our own validators**.
* Thus far we always use to build in ones.
* And to be honest, with all the built in ones you should be able to cover most of your use cases.
* But let's say we have some usernames you don't want to allow the user to use.
* In this case I'll go to my TypeScript file and here I'll add a new property for forbidden usernames for example.
* And in this array I'll add Chris and Anna as forbidden usernames if written like this.
* So now I want to create my own validator which checks whether the username the user entered is one of the two usernames specified here.
* Now a new validator can be added pretty easily.
* **A validator in the end is just a function which gets executed by angular automatically when it checks the validity of the form control and it checks that validity whenever you change that control.**
* So a validator is just a function.
* Let's name it forbidden names.
* *Now a validator to work correctly needs to receive an argument* ***which is to control it should check.***
* So this will be of type form control.
* ***A validator also needs to return something*** for angular to be able to handle the return value correctly.
* There's something should be a JavaScript object.
* And now the following notation might look strange.
* It should have any key which can be interpreted as a string.
* *And this is just types of syntax for saying, hey, we want to have a key value pair where the key again can be interpreted as a string which is true for a key in an object in general.*
* More importantly***, the value of that key key value pair that should be a boolean.***
* So this function here should return something like let's say an object where we have name is forbidden.
* This would be the key name which is interpreted as a string and it could be true.



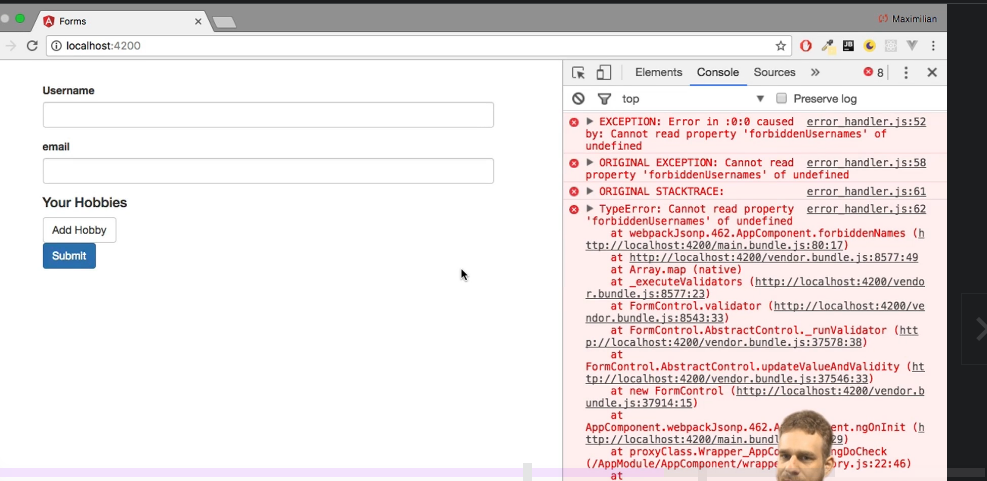
* So this is what this function should return in the end.
* For example, the message itself, the key name is up to us and therefore we can now add the logic in this function.
* In our case here, I want to ***check if the value of the control is part of this forbidden user names array.***
* So I want to check if this forbidden user names index off.
* So if it contains a certain element and the element is the value of our control we pass here the control we're checking.
* If this is the case I want to return an object where I say name is forbidden.
* Any short error code you want is true.
* Now in the other case I want to return null.



* And now this is important.
* ***If validation is successful you have to pass nothing or null.***
* ***You should not pass this object with false.***
* This might sound counterintuitive but that's just how it works.
* *It should be null or you simply omit the return statement.*
* This is how you tell Angler that the form control is valid.
* Well, and this is all we created our own form validator or own validator here, important.
* It receives the control and it returns this object with a arrow code and then true, for example if it is invalid.
* So now to add this, I will go to my username and actually we now want to assign an array of validators.
* So I'll change this appropriately and I'll now add a reference to my forbidden names function.
* Again, don't execute it, only pass a reference.



* And now if we save this, we get an error.

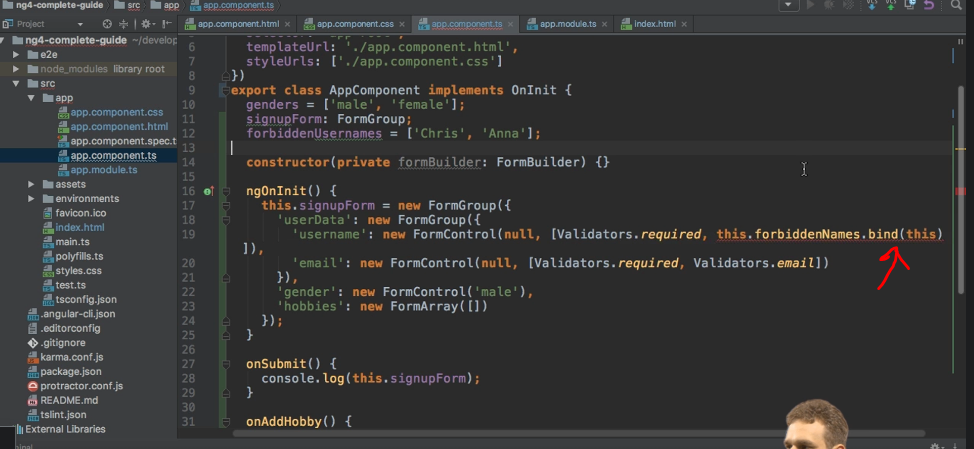


* Now this can be a tough one to spot.
* What's going wrong here? This error has something to do with the way JavaScript handles this.
* In forbidden names I'm accessing this here.

Text

Description automatically generated

* Now this might look all right because I'm in this class and I access this forbidden user names.
* But think about who is calling this forbidden names.
* We're not calling it from inside this class Angler will call it when it checks the validity.
* At this point of time this will not refer to our class here.
* So to fix this, I actually need to bind this the good old JavaScript trick to make sure that this refers to what we want it to refer to.



* So with this in place if we now let this recompile, the error is gone.

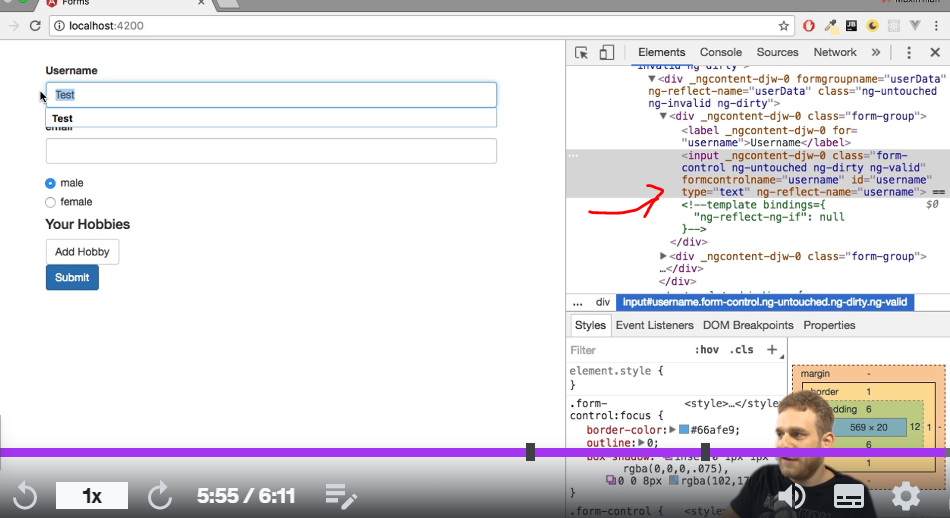
Graphical user interface, text, application

Description automatically generated

* Let's now check the validity of this input.
* Here it is invalid, but if I under test here it stays invalid.
* The reason for this is that right now I'm checking if control value is part of this array, this will return minus one if it is not part.
* Well minus one is interpreted as true though.
* So we need to check if this is not equal to minus one.
* So if this is not equal to minus one, that means we did find it.

Text

Description automatically generated

* That means it is invalid.
* So now with this in place, if we go back, inspect this again and type test here, you see it becomes valid.
* 
* If I type Anna here, it becomes invalid.

Graphical user interface, application, email

Description automatically generated

* So our own validation here is working fine.
* Now in the next lecture I will have a look at this strange error code and how we can use that.
* information alert Schedule learning time Learning a little each day adds up.
* Research shows that students who make learning a habit are more likely to reach their goals.
* Set time aside to learn and get reminders using your learning scheduler.