* -: In the last lecture, we saw how we can use these error codes and we also saw that we can add our own validator.
* We checked if the name is invalid, the username but typically you might need to reach out to a web server to check this.
* That, however, is a asynchronous operation because the response is not coming back instantly.
* Instead, it just takes a couple of seconds.
* So we also kind of ***need asynchronous validators***, which are able to wait for a response before, well, returning, True or false, is it valid or not? Turns out we can create such asynchronous validators.
* So let's create one here in the app component.
* I'll name it, "forbiddenEmails.
* " This **asynchronous validator** ***also takes the control as an argument.***
* Now we also need to return something here *but this will not be an object with an error code and a bullion.*
* Instead***, this will a be a promise which wraps anything or an observable which wraps anything.***

Text

Description automatically generated

* For the observable, you need to import it from rxjs/Observable.
* And, these are of course two constructs which handle asynchronous data.
* That's exactly what this validator is about.
* So here, I now want to create a promise in this validator.
* A new promise it will be which returns us anything.
* And as all promises, this receives a function with resolve and rechecked as arguments, we can execute in that function, and in that function, I now want to set a timeout.
* After one and a half seconds, I want to return a response to simulate the fact that we have a asynchronous task like reaching out to a server.
* So this will be the function which gets executed after one and a half seconds, this anonymous function here.
* And here, I will simply check if control value equals test@test.
* com.
* Of course, we could also check if it's part of an array, I will simply check for this single email address.
* If this is the case, validation failed, and as in the synchronous validator case this is when I will return an object with a key value pair with this error code.
* Here, of course, since we are in a promise I don't return, I resolve, and again, I resolve this object where I say, "emailIsForbidden," like this, set this to true, and if we pass this check here so in the else case, that we have a valid input, that we have a valid email address, I will simply resolve null.

Text

Description automatically generated

* Now of course, we need to return this promise in the end.
* So with this we set up our asynchronous validator.
* Now we can add it.
* Let's add it to the email.
* And here you don't add it in this array of normal validators.
* Instead we use deferred argument we can pass to form control.
* This is asynchronous validator or an array of such validators, just like the normal validators but reserve for the asynchronous ones.
* So here I will use my forbidden emails validator.
* Again, don't execute it, simply pass the reference.
* And you need to bind this.

Text

Description automatically generated

* If you plan on using this in there, I don't, so I don't need to do it here.
* Now let's check if it works.

Graphical user interface, text, application

Description automatically generated

* If we compile this and we inspect the email, it's invalid.
* If I enter something here, you see it stays invalid.
* If I enter test two, closely, watch ng-invalid.

Graphical user interface, text, application, Word, email

Description automatically generated

* Did you see that? It changed to ng-pending because it was evaluating the input? Then it changed to ng-valid.

Graphical user interface, text, application, email

Description automatically generated

* Now if I remove the two and therefore I will enter an invalid email address because that's the one we're checking for.
* It again, switches to pending and then to invalid.
* So this is now how the asynchronous validator works and of course, this would work with code where you reach out to the web, for example, and we correctly therefore, of course, see our error message.
* So, these are asynchronous validators.
* Really easy to add when using the reactive approach, as you saw.