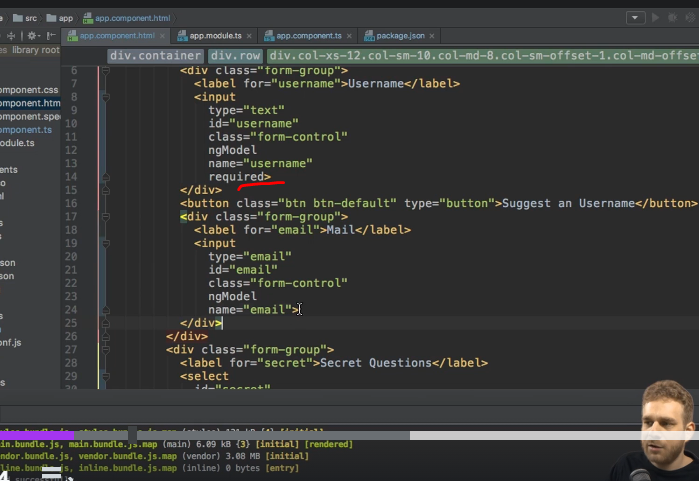
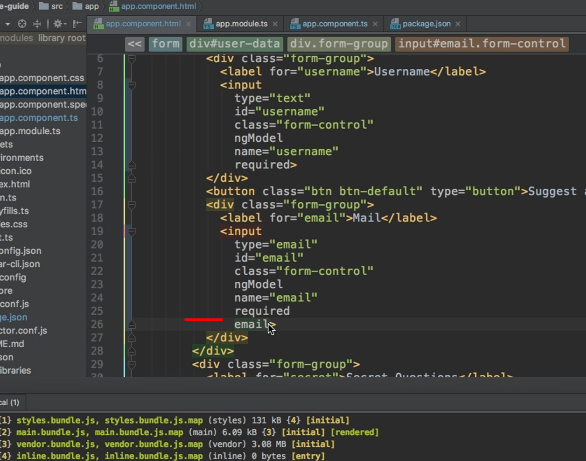
**190. Adding Validation to check User Input:**

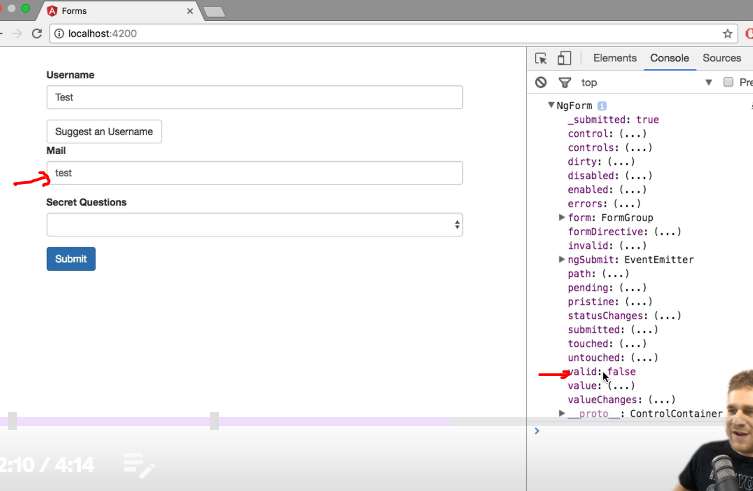
* -: So, as mentioned at the end of the last lecture.
* Would be nice if we could validate the user input.
* Something very important in any app you build.
* Whilst you should still validate input on the server, as the front, and can always be tricked.
* you can greatly enhance user experience, by also validating the input here.
* *For example, you want to make sure, that none of the fields here is empty, and that the email address, actually is a valid email address.*
* So, let's go back to our app component, to the HTML file, to the template, and see how we can add such validators.
* *Now, since we use the template driven approach.*
* *We can only add them in the template.*
* ***Required:***
* And here we can, for example, add "***required***" to our username input.
* Now, "**required**" is a default HTML attribute, You can add to a input.
* Here, However, Angular will detect it, so it acts as a selector for a built indirective shipping with Angular.



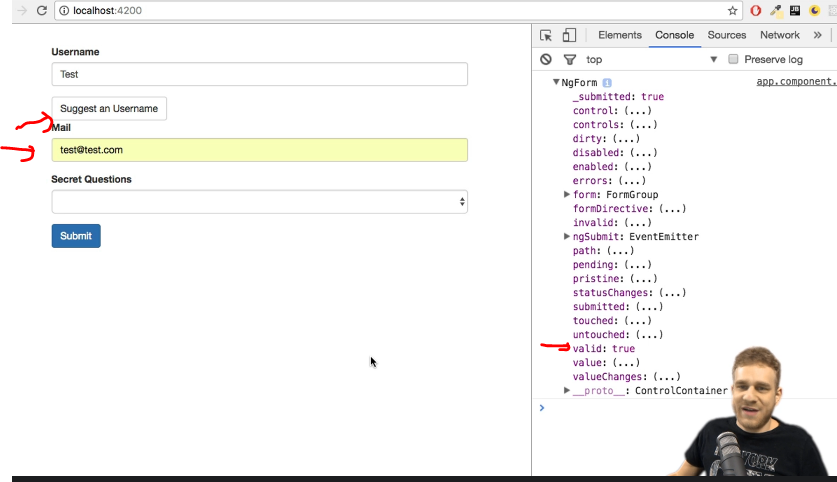
* And it will automatically configure your form you could say, to take this into account.
* To make sure that now, this will be treated as invalid, if, well, it is empty.
* And on the email, we can for example therefore, also add, "required".
* **Email directive:**
* And there also is a email directive you can add.

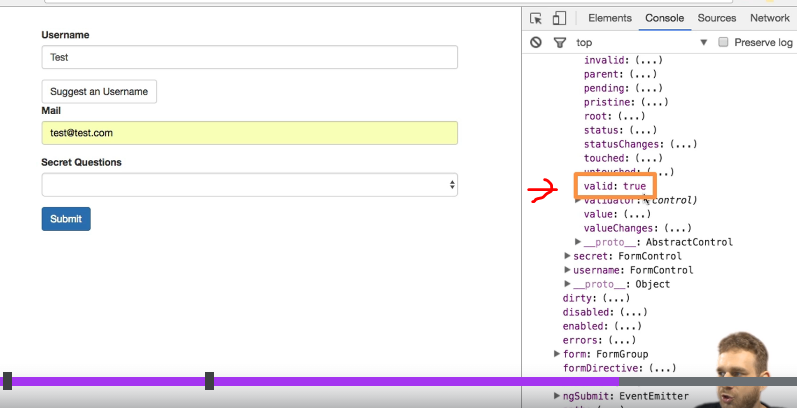


* Now, email is not a built-in HTML attribute.
* It still is a directive.
* And keep in mind, required is only treated as a selector for a angular directive here.
* And email is simply another directive made available by Angler, which makes sure that this is actually a valid email address.
* So, now if we save this, and we have a look at our form here, and I submit it.
* I can still submit it, because we haven't set up anything which would prevent us from doing so.
* But if we have a look at it and check the valid attribute, you see it as false.
* And if I enter something here, I enter something here.
* So, it's filled, but the email address is still not valid.

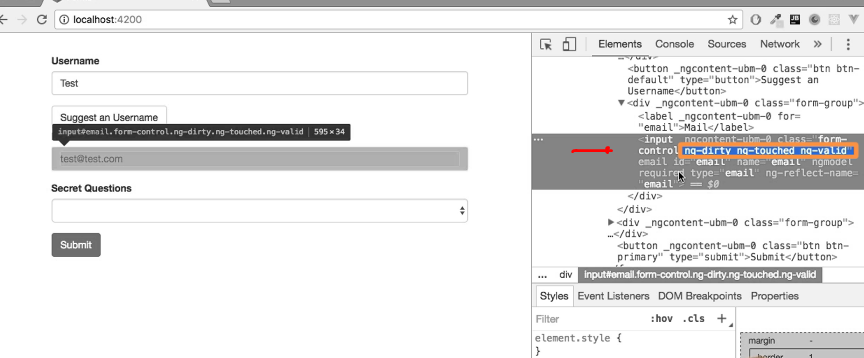


* You see, that's still valid as false.
* Only, if I turn this into a valid email address here.
* only in this case.
* Now, if we submit it, now you see valid as true.
* So now, Angler tracks the state of this form, and correctly informs us, or gives us a chance of querying it.
* Whether this form is valid or not.
* And actually this does not only work on form level.
* If we dive into the actual controls.
* You see that on the email control.
* We also have a valid attribute, which is true.





* ***So, it tracks this on a per-control level, and then also on a form level.***
* Now, there also, is a number of place where it tracks this, and helps us.
* If we inspect this email element here, in the HTML code.



* You see, that it adds a couple of classes, because the form control class here is by us.
* It's a bootstrap class to give it some styling.
* But, ng dirty, ng touched and ng valid? these are not classes added by us.
* And the ng at the beginning makes it pretty clear, who is responsible for adding these classes.
* Now watch these classes.
* If I remove the @ sign here.
* you saw that? ng invalid was added, and ng valid was removed.
* So, Angler dynamically adds some classes.
* Giving us information, some CSS classes.
* Giving us information about the state of the individual control here.
* whether it is dirty or not.
* So, whether we did change the initial value.
* whether it is touched, or not.
* So, whether we clicked into it, or not.
* And whether it is valid, or not.
* Now, with that information.
* We can style these inputs conditionally.
* So, let's the next lecture take advantage of the fact, that Angler tracks the state of the validity, and of the form overall.
* And change the styling and the behavior, the user experience with that form.