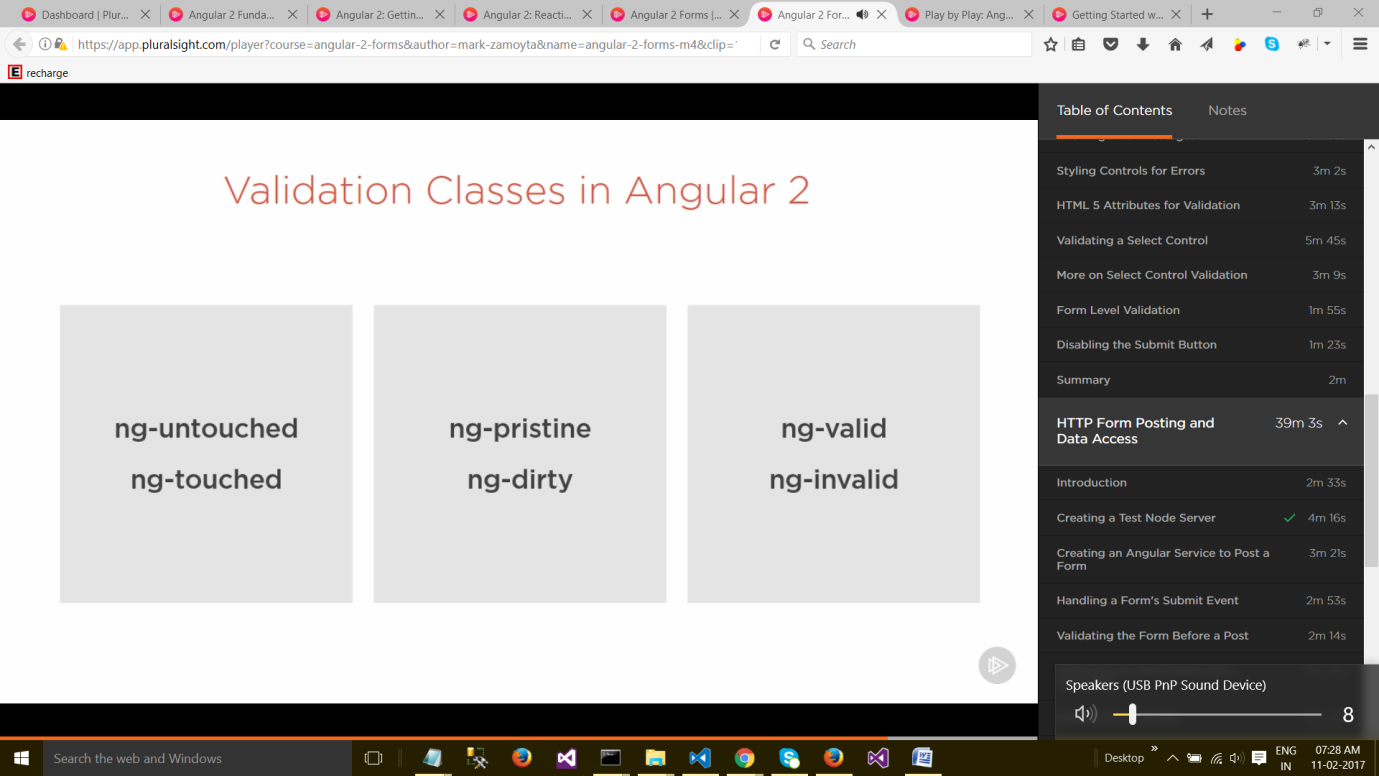
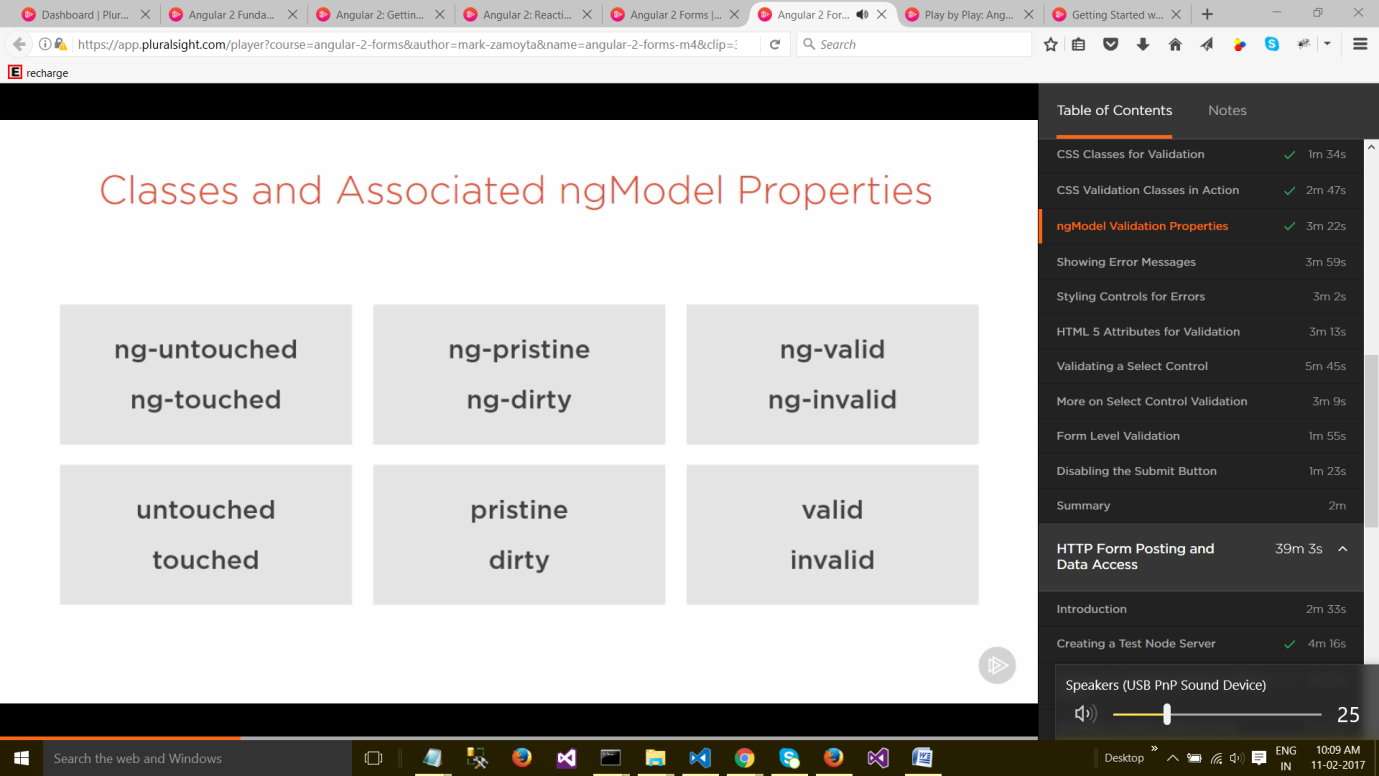


**CSS Classes for Validation**

**Let's take a look at validation classes in Angular 2. By validation classes, we mean real CSS classes that Angular is going to attach to a field on the form.** Once a field has a class, we're able to style that field as a developer. But, for now, let's just take a look at what these classes are.

* When a form is first created, all the fields are marked as **ng-untouched**. And, this is the CSS class that gets attached to each field.
* And, as we tab off of fields, the class changes to **ng-touched**, meaning that the end user actually had a chance to enter information, and whether they did or not, they still left that field. So now, the field is labeled **touched**.
* There are two other classes: **ng-pristine**, and **ng-dirty**, which monitor whether or not the user actually entered any information, or left the field in its initial state.
* So, pristine is the starting value. **Pristine means it hasn't been changed or touched**. But, once the user changes something, it immediately gets set to **ng-dirty**. And, also, for these classes, such as ng-touched or ng-dirty, once these are applied, they're applied until you reset the form.
* And, there's one final set of classes. There's an **ng-valid**, which tells us whether or not the field is valid. And, there's also an **ng-invalid** class.
* So, depending upon how you set up your validation, right from the get-go, it might be set as ng-valid. Or, if you have something like a required field, that would be set to ng-invalid. So, let's see how these work in a real form.

****

****

**change the border, etc on the basis of state of control**

**HTML 5 Attributes for Validation**

Let's take a look at validation for a text input. We'll look at some of the attributes that we can use in the HTML markup. So far, we've been working with required, and, we've seen how that works. We can tab out of the field and we get our error message. But, there are some other common ones that get used. Let's try maxlength. I'll set a maximum length of three characters. I'll start to type a name, and it stops me at three characters. So, we're not getting an error; the browser, itself, is preventing us from entering any more data. And, that's fine. Some browsers might work in a different way, but, for our case, this is good. Now, certain browsers also implement a minlegth, a minimum length, and Chrome is one of them. So, I'll save this, and I'll start to type a name, but, I'll leave it at two characters. And, I'll tab out, and, we get our error message. Now, our error message isn't much use to us; "First Name is required." So, we would have to fix this up, as we add validation to our field. So, I'll try it again, and we get our message. But, the problem is that minlegth isn't widely accepted. It's not an official HTML 5 attribute for an input. I'm here at caniuse.com, and let's take a look at minlength. If we look at the chart, we can see that it's fine in Chrome, and the latest version of Firefox, but, for things like IE and Edge and Safari, it's not going to work. And, we can compare that to maxlength. That's implemented, pretty much, across the boards. So, another option we have is, we can specify a regular expression for validation. Let's do that. The attribute we use is pattern, and let's just set a simple pattern to start with. We'll make sure that the first name begins with a Q, just an arbitrary rule for testing. A refresh. And, let's try typing a Z. And, I'll tab out of it. We get our error message; it doesn't begin with a Q. I'll type Quenton, and, we still get our error message; that's because there's a problem with our regular expression. If we have just Q, we're fine. So, let's specify anything that follows Q dot star. And, now we can type Quenton, and we're fine. And, as we delete it, we'll delete the Q, type a Z, and we get our error. Now, we saw earlier that minlength wasn't supported, and, we can go ahead, and add our regular expression for that. You can put, let's say, a minlength of three. So, we'll put three characters, and, the last character will accept one or more of those. So, I'll refresh it. And, let's see if it accepts a single character. I'll type a Z, and tab out, and we get our error. I'll type two characters, we still get our error. And, on the third, we're fine. So, regular expressions work well. You can actually search the Internet for regular expressions for email addresses, URLs, phone numbers, or anything. And, the pattern attribute is well supported for regular expressions. **If we look at Can I Use, let's search for "pattern," we can see that it's accessible, pretty much, everywhere. So, Angular works well, in validation, when it comes to looking at the native attributes of an input field. We saw it work with minlength, maxlength, pattern, and, of course, required.**

Form Level Validation

* So far, we've been looking mainly at validation at the field level, for inputs of type text, or our select, in options.
* But, Angular also keeps track of validation at the form level.
* If you remember, earlier in the course, when we specified our form tag, we created a **template reference variable** called #**form**, and, we set it to **ngForm**. Now, you might be wondering where **ngForm** comes from. **Well, whenever you have a form in Angular, Angular will automatically add a directive, called ngForm.** And, that directive gets exported as the string, ngForm, and that's what we see here**. So, we can access this ngForm directive, now, with our #form template reference variable.** So, I'm back to having just a single firstName field. And, let's scroll down to the bottom, and after our Ok button, which is of type submit, this eventually will submit a form; we don't have that set up yet. But, let's just go ahead and use our template reference variable form, and, let's take a look at some of the validation properties that we had on fields. The same ones exist on the form, itself. So, let's look at pristine. And, I'll just wrap this, so we get it looking a little bit bigger. So, we get true for pristine. If we enter a field and enter some data, pristine becomes false, so, that's fine. And, let's also take a look at valid. You'll see we get true for valid. And, that's a little deceptive. Let's go up and look at our text field again. We specified this pattern from our previous video. But, for HTML 5, we also need to specify the required attribute. That way, the field is required, and the pattern will be checked. So, I'll save this, and it'll refresh. So, now, valid is false. And, we can go ahead and check any one of these six validation flags. Valid is valid, touched, untouched, and dirty and pristine.

Disabling the Submit Button

So, using these validation properties on the form itself, on the ngForm, they're mainly useful and things like a submit handler. If you are about to submit a form, you'd want to make sure it was valid, and that everything was good to go. But, one place where you do see things like this used, outside of a submit function, would be to disable a button, such as the submit button. So, let's do that. I'll make some room here. And, we can bind to the disabled property. Again, properties need these square braces, to perform a property binding. And, we'll just specify form.invalid. So, if the form is invalid, the button will be disabled. I'll remove this test message, and let's refresh. So, our Ok button is disabled. If I mouse over it, you can see we get the red circle and line through it. There's no way to submit the form this way. If I go in, and I enter a valid value, the button becomes enabled. Again, if I back up a bit, and I don't meet the requirements of a minimum length of three, we get the disabled button again. So, that's pretty much all we're going to cover in this module. We still need to cover submitting a form, and validation goes along with that, too. But, we're going to be covering that in the next module of this course, where we cover handling submit, making sure everything's good, and then, actually posting it to a server, and getting a response, whether successful or unsuccessful.