





**A Simple Form and ngForm**

We're all set up, so let's create a form. We'll take a look at AppModule, and we can see that for our bootstrap property the module is going to boot up with AppComponent. So we'll look at app.html, and this is the markup that we see in our browser. And we want to cut this out. So all we have is a main tag and our router outlet. We'll just keep the routing in this application, even though we're not going to be using it. You can add your own routes later, or you can just cut it out on your own.

But, the default route is going to be home. So, let's take a look at home.ts. And essentially all this is is a CSS file and our template, home.html. And this is where we'll be working in this course. We'll create our form here. We'll make a heading of Angular 2 Forms, and let's create a normal HTML5 form. So, I created a form tag, and it's not doing much. It just has a text field. Input type of text. And it has an OK button. So I'm just going to save everything, and here we have our form. It's got the title Angular 2 Forms. And the text field. Now, nothing's happening in this form. If we click OK nothing happens.

We don't have any kind of posting set up. So it's pretty useless right now, but there is one very important thing going on. **When we use a form tag in an Angular template, Angular 2 is going to add a directive automatically to this form. And the way you access a directive is with a template reference variable.**

So let's create a variable called **#form,** and we'll set that equal to **ngform**. So the directive that Angular adds to form is called **ngform**. And it exports itself with that same name. So this template reference variable, **#form**, is the actual instance of a directive.

And we can also access that directly from this template. **We'll bind to form.pristine. And pristine will just tell us whether or not the form has been touched or not by a user.** If any data has been entered. We'll cover this more in the module on validation. But, for now, let's save this and see what we get.

So we get **true** showing underneath the Name field. **So the form is pristine, it hasn't been touched yet. But, you'll notice if we do enter characters into this field, Angular thinks the form still is pristine.** And the reason for this is that we have to let Angular know which fields it should be interested in.

Right now, it's not aware of our text field at all. So, let's go to our text field, and we'll give it a **name**. **And we'll also let Angular know that it should be aware of its value.** And we do that with the **ngModel** directive. I'll save it. Now when I start to type in a name, pristine should go from true to false, and it does. And even if I backspace, pristine stays as false.

The form has been modified to some degree. So this ngModel directive, that we've placed on the input, this is part of the forms module. And that's why we have access to it. In Angular 1, ngModel is used for data binding. And it's used in Angular 2 for data binding as well. But even though we're not binding any data here, Angular is still becoming aware that we want it to control this field. So we can specify ngModel without specifying any data to bind to.

**And also when using ngModel, we want to make sure that the field has a name. Let's see what happens if we take the name out. I'll save it. And we get an error.** I'll increase the size here. If ngModel is used within a form tag, either the name attribute must be set, or the form control must be defined as stand alone. So for Angular to work with a field, you need to remember to add that Name attribute. And the form works again fine.

**Shutting Off Browser Form Validation**

One of the things we want to do when we're working with forms is we want to turn off browser validation. There's going to be a whole module on validation, but it's important to get this set up up front, in case you run into it. Let's say our input field, our text field, is required. Required is a valid HTML attribute. So, I'll save this. And I won't enter a name, but I'll click the OK button. The browser pops up this warning message.

Please fill out this field. Now, this message if going to be different on every browser. Like here we can see it went away automatically. Let's see what that same message looks like on Edge. So here's an Edge browser window, and I'll click OK without entering a name, and we get a very different experience. The name is surrounded in red, and the message doesn't have that cool look that you get in Chrome. And this is referred to as browser validation. It's done by the browser itself, and you have no control over how it looks, or how it behaves. So we'll see later in this course how to get control over that. **But for now, we want to shut off this browser validation. We go to the form tag, and we add an attribute called novalidate. I**'ll save it. And now when we hit the OK field without entering a name, we don't get the message from the browser. It still doesn't do anything, but at least later on in this course, we'll get control over how to actually show a validation error. And this is important, because most likely, the users of your application are going to be looking at it in different browsers, and you want to give them all the same experience. So, remember to turn off browser validation by specifying the novalidate attribute directly on the form. And this is part of the HTML standard, it's not an Angular 2 feature.

**Styling Forms and Text Input Using Bootstrap**

So we have some basic forms working, but before we go too far, I just want to make sure our forms are looking good. So I'm here at getbootstrap.com, and we'll use Bootstrap to style our forms.

I'll click on the Getting Started link, and if we scroll down a bit, we can see the Bootstrap CDN section. I'm not interested in a theme, or the JavaScript part of Bootstrap, just the CSS. So, I'll copy this link right here. Now, there are lots of ways to get Bootstrap into our application. One would be through Bower or NPM, and then it would have to tie it in with our package manager, whether it's Webpack, System JS, Browser FI, and there are a lot of options. But, in order to get started quickly, I'll just take this link and paste it into our index.html page. And I'll just paste it as a link at the bottom of our head section. So now we'll get Bootstrap directly off the internet from Bootstrap CDN.

Now, in order to use Bootstrap, let's go to our home.html page. And the first thing we want to do is to warp our form and its header inside of a container. So that should give us some simple styling now. I'll save these two files. And the browser refreshes, and you can see that we have our Ariel font and our form has a margin too.

Let's look into styling the rest of the form. For the button we can add some Bootstrap classes. Btn will give it regular button styling. And we can say btn primary to give the button emphasis. So we have a Bootstrap styled button. I'll take this form.pristine tag out, and let's also style our text field. We can wrap it and its label in a div. We'll give it a class of form group. And we'll add a label here. I'll take out the placeholder. The name will be First Name, and we'll also give our input a class of form control. And I'll save it, we'll see how this looks. So there we have it. A form control takes up the full width of the viewport. And that's the kind of styling we get with Bootstrap. And because we're using a form group, we'll have our OK button on its own line now. One thing I should probably mention is it's standard in HTML to add a for attribute to a label. And this will assign the label to the input pointed to by for.

**So, you would put, basically, your firstName ID here, and then the input tag would have an ID itself, called firstNameID. And that simply links up the label with the input**.

If you were to touch or click on the label, that input would get focus. But, more commonly, this is just left out, and I'm going to leave it out as well. To get focus on a field, you can just tap or click on the field. There's already enough going on with the input, in that we're giving it a name right here. And also, needing to give every field an ID is just too much unless you absolutely need it. So, now that we have this, let's also quickly create a Last Name field. So now we have our First and Last Name fields, styled with Bootstrap.

**Browser Inconsistencies for Input Types**

I'm here at the Mozilla Developers Network. And I want to take a look at the input tag, especially the Type attribute, in more detail. Here's our input tag, and if we look down a little bit, we can see the Type attribute. And we've been working with checkboxes, and radio for radio buttons. And of course, we've been working with text, right here. But, as you can see, there are various other inputs that we can use.

**And the main problem with using these is inconsistency among the browsers.** **I'll scroll up a bit, and let's take a look at the date field right here.** Let's add this to our form, but we'll see how it behaves in different browsers. Again, I'll just copy the First Name field. I'll go down to the bottom, and I'll paste it and we'll make a date out of it. Let's just call it Date Hired, and type will be date. We'll take out this other information for now. I'll save it and let's see what we get. So, we have Date Hired. It's looking for a month, day and year. And we can go ahead and type in the values, or we have a drop-down calendar. And that all works fine on Chrome, but let's see what it looks like on another browser. I have Edge from Microsoft up, so let's look at that one. So we have our Employee Form in Edge, and you can see here the Date Hired looks much different. Let's look at them side by side. So, first of all, the alignment's off. Next, I click on the field, and you can see that the interface is much different. It's almost as if it was meant for mobile. And again, looking at the Chrome interface for date/time, we get the calendar month. Now, which one is better doesn't really matter. The point is your users are going to be getting different experiences from different browsers. So, in cases like this, when using dates, it's better to use third-party controls which will give you consistency across the various browsers. And we'll see that later in the course. And just to check this out again, let's also just play around with the color type. **Color** is another standard HTML5 type for an input. On Chrome, we get a big, black color to start out with. And we get a color dialog. And then Edge, let's refresh this, and we don't see anything. Color doesn't work at all in Edge. So, again, whenever you use these different types for the input tag, make sure they work properly on the browsers that you're using.

