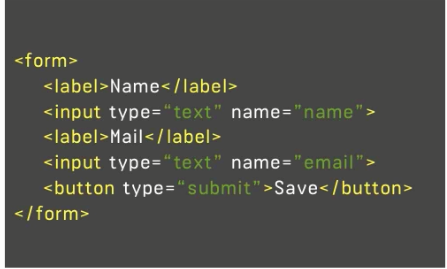
**Section 15: Handling forms in Angular Apps**

**Section 15: Lecture 168//Module Introduction**

1. This course module is about forms i.e. forms in our HTML page and how angular can help us with those forms.
2. We know that a form is something that we can submit to the server and then we must keep in mind that we are creating single page application – So there is no submitting to the server.
3. Instead we would need to handle the form through angular and then if we must submit something to server then we would need to submit is through angular HTTP Service – which we will cover later.
4. So, we must handle our forms through angular and actually angular will help us handling it. It will not only allow us to check the values that user entered but it will also allow us to check if the form is valid.
5. It will allow us to change the way in which form is displayed, put some red borders around invalid – so angular really ships with some powerful tools when it comes to working with forms.

**Section 15: Lecture 169//Why do we need Angular’ s help**

1. Let’s have a closer look at how angular works together with forms. Consider below given form:



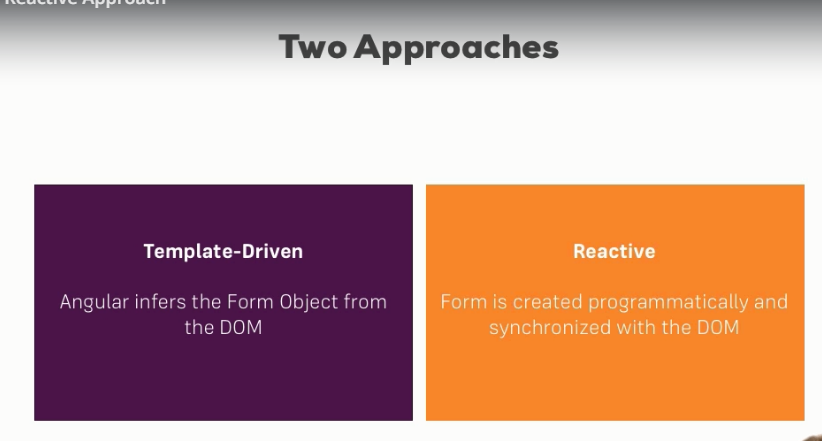
1. This is the HTML code of the form which we might use and there is nothing angular specific in this form. This is normal HTML code which you place in your HTML documents.
2. Angular’ s job here is to allow you to retrieve the values of the fields entered by you here and also check some other things like - is the form valid?; did the user entered the valid information and all that will happen in JavaScript i.e. in Typescript so on the angular 2 side of the App.
3. So, we somehow need to be able to parse the values, the user enters and you somehow need some JavaScript object representation of your form in your typescript code to work with.
4. So, we need to fetch the value of the form in JavaScript code and there we could have the key value pairs i.e. name value which is referred to as key and value as the email here. These all are mentioned as name in the html input type; so we can fetch the value of these in JavaScript by using these names.
5. There can be other properties which would change on the scenarios according to which the values of the input type changes
6. Javascript object representation of our form makes it simple for us to retrieve the values and to see the state of the form and to work with it.
7. In the next lectures we are going to have the detailed look.

**Section 15: Lecture 172//Template-Driven (TD) vs Reactive Approach**

1. Before diving into the code it’s very important to understand that angular actually offers 2 approaches for handling forms

**Template-Driven (Angular infers the form Object from the DOM)**

**Reactive (Form is created programmatically and synchronized with the DOM)**

1. In template-driven – we set up our form in template in HTML code and angular will automatically infer the structure of your form and will infer it i.e. which controls your form has which inputs and makes it easy to get started quickly.
2. There is another approach which is more complex i.e. the reactive approach – there we define the structure of the form in typescript coat. You also setup the HTML code and then manually connect which might sound more complicated than it is in the end and therefore it gives us greater control over it. We can fine tune every little piece about your form.
3. Now, in this module we’re going to look at both approaches in the next tutorials.
4. We will start with template driven approach which is fine for many use cases, before we then dive into the advance reactive approach.
5. 

**Section 15: Lecture 173 //An Example form**

1. `So here we are on the simple example just to get started. The code is attached to this lecture.
2. The form which must be submitted to the server should have action item on its from, but this needs to be submitted to the angular so there is no action item on the form.

**Section 15: Lecture 174 //TD: Creating the form and registering the controls**

1. Let’s understand how angular contains such a JavaScript object representing our form in the template driven approach. The great thing is that we don’t need to do anything.
2. Make sure you add your forms module in your app module and add the imports at the top. This built in module in angular ships with a lot of forms and related functionalities that are actually needed to get this template driven approach to work to get this form creating by angular to work and get this form creation by angular.
3. By default in the cli project it should get imported, with this imported angular will automatically create a form for you and so our JavaScript representations of that form when it detects a form element in our html template like it does here.
4. So, we can think of the form element as the selector for some angular directive which then creates such a JavaScript representation of the code for us.
5. Of course we cannot see that representation as of now and it would be very empty because one thing will not happen automatically – angular will not automatically detect our inputs in this form.
6. And the simple reason for this is that, while you should be able to argue that it should be able to scan your HTML code and detect that you have input here and you have a select dropdown here.
7. You still might not want to add all these elements as controls to our form – so, with control we are referring to – what is in the JavaScript object.
8. And again not every input in your HTML code might be the control you want to have in your JavaScript form. Maybe you have a dropdown for which value only changes something we view on the UI but the input should not be the part of what gets submitted.
9. Maybe you use some 3rd party package which adds some custom form controls which are not labeled input, which don’t use input as the selector and then angular would have no chance of detecting that this is a control of your form – so you still need to register controls manually, you need to tell angular – within the from element what should be the actual control of my form and this is what we are going to do now - tell angular how does our form look like - which controls do we want to have in the template driven approach this is super simple.
10. You simply pick the input which you want to add as a control like this input here. We will now add ng model in the in the user name input type.
11. Now we know the ngModel from the 2 way data binding and it actually is the same directive. Now, we will add ngModel without any parenthesis and without any square brackets hust like this.
12. This is enough to tell angular that this input is actually a control of my form, ngModel in the end is a directive made available in the forms module something we mentioned earlier in this course when we had a look at two way data binding. This is key to understand – we can use it to get two way data binding but it actually is a part of the bigger module with more feature giving you full control of the forms.
13. Now, for this to work i.e. for this to be recognized as the control in our form we need to give angular a one other piece of conformation i.e. the name of this control; right now it will see that this input should part of the JavaScript object representations of this form.
14. So, what ever the user enters here as a value – it should be the value of the control I.e. we need to give that information to angular and we do this by adding the normal HTML attribute name – so, name is nothing angular 2 specific.
15. Name is the default attribute one can add to any HTML control. So, here the name might be the username because that is what we enter in this input.
16. And with this the control will be registered in this JavaScript representation on the form. Now, I’ll do the same for the email – restructure it – so that it is easier to read.
17. In the next lecture we will see how to submit such a form and see these key value paires.
18. app.component.html:
19. <div class="container">
20. <div class="row">
21. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
22. <form>
23. <div id="user-data">
24. <div class="form-group">
25. <label for="username">Username</label>
26. <input
27. type="text"
28. id="username"
29. class="form-control"
30. ngModel
31. name="username"
32. >
33. </div>
34. <button class="btn btn-default" type="button">Suggest an Username</button>
35. <div class="form-group">
36. <label for="email">Mail</label>
37. <input
38. type="email"
39. id="email"
40. class="form-control"
41. ngModel
42. name = "email"
43. >
44. </div>
45. </div>
46. <div class="form-group">
47. <label for="secret">Secret Questions</label>
48. <select
49. id="secret"
50. class="form-control"
51. ngModel
52. name="secret"
53. >
54. <option value="pet">Your first Pet?</option>
55. <option value="teacher">Your first teacher?</option>
56. </select>
57. </div>
58. <button class="btn btn-primary" type="submit">Submit</button>
59. </form>
60. </div>
61. </div>
62. </div>

**Section 15: Lecture 175//TD: Submitting and Using the Form**

1. In the last lecture we configured our form here we added some controls by placing ngModel on the inputs.
2. Now, let’s make this form submit table so that we can actually see that what the user entered – for this we will go to app.component.ts; we already do have a method that we will use later – lets ignore it for now.
3. We will add here a new method i.e. onSubmit(); this should be triggered whenever this form is submitted by the user on submit.
4. In onSubmit we need to output whatever the user entered. Well, first of all we need to call this method.
5. So, back in a template how we can call on submit. Now, you might think that a good place would be on a click listener on this button here at the bottom – because this is the button we want to click when we want to submit the form. However, this is not the best place to do it. Keep in mind that this button here is of the type submit.
6. So, if we click the button as it is placed inside the HTML form element – something else will happen. The default behavior of HTML will be triggered to call it like this.
7. If we have a button inside the form this button will submit the form and will send request normally. But besides that it will also trigger the JavaScript event i.e. the submit event that is built into HTML. However, angular takes advantage of this and gives us a directive that we can place on this form element as a whole. It is called **ngSubmit** and it actually gives us one event that we can listen to so let’s wrap it in the paranthesis.
8. This event made available by the ngSubmit directive will be fired whenever this form is submitted. So, when ever this default behavior is triggered. And here we can call our method i.e. onSubmit().
9. Now, if we save this and let it recompile. Lets go back and open the developer tools.
10. Now, lets see the actual value of the form i.e. the form object and to see it we have to go back to our template because we are in the template driven approach as the rule of thumb for this approach everything you do in this approach is you do in template.
11. Anything you want to change on as functionality – you do it in template. On this form object we want to get the form created by the angular.
12. Now, we learned about the local reference – we can place on the HTML elements to get access to them; so, we can place #f on the form element.
13. Now, we can access this form element as the f reference on our template and we can pass the f as an argument to the on submit method and print it there
14. Now, in the onSubmit method we will receive the form which would be of the type ElementRef.
15. So, we can see form object here, we can also see some of the classes here.
16. But this object is not what we want – its not a JavaScript object created by angular. But this object is stare to get to it and there actually is a trick we call but its not a trick it’s a default way to get access to it.
17. Here, we must set this local reference equal to something. We must keep in mind that a form element is a selector for a directive built into angular which will create this JavaScript object automatically and then it can expose some data we can fetch here on this form element.
18. We can get access to it by writing ngForm in the local variable by putting the quotation marks.
19. It tells angular - please give me access to this form you created automatically. That’s just something we must keep in mind.
20. This is how we get access to this form in this JavaScript object created by angular automatically.
21. So, here where we get this form – now this form will no longer be of element ref but it would be of the type ngForm in typescript file.
22. Now, we can look in the object that we have many properties in the object that is now printed. When we expand it we can see a couple of key value pairs.
23. app.component.ts:
24. import { Component } from '@angular/core';
25. import { NgForm } from '@angular/forms';
26. @Component({
27. selector: 'app-root',
28. templateUrl: './app.component.html',
29. styleUrls: ['./app.component.css']
30. })
31. export class AppComponent {
32. suggestUserName() {
33. const suggestedName = 'Superuser';
34. }
35. onSubmit(form: NgForm){
36. console.log(form);
37. }
38. }

24. app.component.html:

<div class="container">

<div class="row">

<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

<form (ngSubmit)="onSubmit(f)" #f="ngForm">

<div id="user-data">

<div class="form-group">

<label for="username">Username</label>

<input

type="text"

id="username"

class="form-control"

ngModel

name="username"

>

</div>

<button class="btn btn-default" type="button">Suggest an Username</button>

<div class="form-group">

<label for="email">Mail</label>

<input

type="email"

id="email"

class="form-control"

ngModel

name = "email"

>

</div>

</div>

<div class="form-group">

<label for="secret">Secret Questions</label>

<select

id="secret"

class="form-control"

ngModel

name="secret"

>

<option value="pet">Your first Pet?</option>

<option value="teacher">Your first teacher?</option>

</select>

</div>

<button class="btn btn-primary" type="submit">Submit</button>

</form>

</div>

</div>

</div>

**Section 15: Lecture 176//TD: Understanding from State**

1. In the last lecture we learnt – how to submit a form created by angular and how to have access to this object angular created for us. Now, we had a look at the value property which stores the input of user in key value pairs.
2. Now, we had a look at the property value which stores input in the key value pairs.
3. We see that we have many other properties and that’s pretty cool about the JavaScript object about the form handling in angular. It allows us to really understand the state of our form.
4. We can see which controls we registered here on the controls object e-mails, username and the secret and each control is of the type form control of course a type made available where each control then has a couple of properties mostly the same properties we have on the overall form though and therefore lets go back to the overall form – for example properties like dirty, disabled, enabled, errors etc.
5. Most of these properties are pretty self-explanatory – dirty for example is true because we changed something about that form.
6. If we reload the page and submit it now, we will see dirty as false because we didn’t type into any input so therefore of course dirty is false.
7. Disabled would be true if the form was disabled for some reason, invalid id false because we haven’t added any validators – so it isn’t invalid, it is indeed valid. We do have a valid property done here too. So the form is valid right now. We will make sure that we add validators to make sure that a valid email address has to be entered – for example – and we also have touched for example to see did we click into some of the fields the difference to Dirty would be that for dirty we have to change the field, have to change the value of a field for touched simply.
8. For that we have to click simply into it and now it would be touched and we will later learn how these properties can be helpful in well changing the user experience for example disabling the submit button if the form is not valid. We will come back to this later.
9. It’s important to understand that you have all these properties and feel free to dive into the output we logged here and understand which properties you have, how they change which properties the individual controls have and so on.

**Section 15: Lecture 177// TD: Accessing the form with @ViewChild**

1. In the last lecture we learnt how we can register controls and how we can submit our form and also which properties this form has now.
2. Right now we submit the form here by passing the form which we get via entry from here to the onSubmit() method. This is absolutely fine and probably the approach we will use in many use cases.
3. We just need to have attention on one another approach here - we don’t have to submit it here. As in the component section where we learnt about **@ViewChild which allowed us to access a local reference element controlled or which holds a local reference in our typescript code.**
4. In the end we do just have a local reference here - it might not point to an EelementRef but to the ngForm object – it still is local reference now in our template so we can all use @ViewChild here.
5. This decorator we learnt about in the component section and we need to get the reference of the element which has the local reference f on it. So, we pass f as string i.e. ‘f’ as the argument to the @ViewChild decorator.
6. And I can store this in the variable name and sign up for any name you like to which name we like – this signupForm will be of the type ngForm and now onSubmit() method we could output signupform to the console.
7. And you should see that if I now submit this again we still have this form. So, this gives access to us to the very same form without passing it to onSubmit().
8. **This is specially usefull if you need to access the form not just at the point of time when you submitted but also earlier.**
9. Now, let’s determine if the form is valid or not this will check the validity of the form. We will add some validations and take advantage of tools angular gives us here.

**Section 15: Lecture 179//TD: Adding Validation to check User Input**