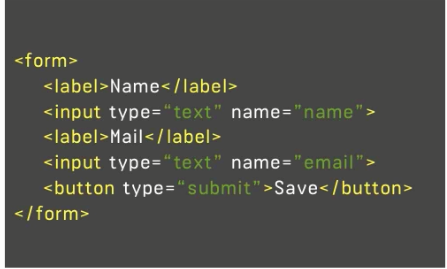
**Section 15: Handling forms in Angular Apps**

**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:



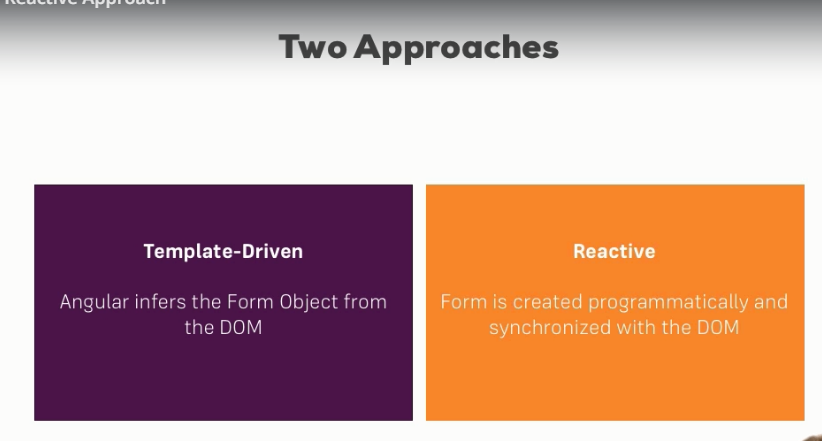
1. This is the HTML code of the form which we might use and there is nothing angular specific in this form. This is normal HTML code which you place in your HTML documents.
2. Angular’ s job here is to allow you to retrieve the values of the fields entered by you here and also check some other things 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**

1. We need to always validate the user input. The validation should be put at both front end and backend as frontend can always be tricked. We will also enhance our experience here by also validating the inputs.
2. We will put validation here that none of the field here is empty and no email address is valid email address.
3. We will now add validators in app.component.html. Now, since we are using the template driven approach so, we can only add the validations in the template.
4. The fields that we don’t want to be empty – there we will add required keyword, which will check if the fields are left empty. Here it acts as a directive for the built in directive shipping with angular and it will automatically configure your form to check if the valid input is entered in the form.
5. In the email input type, we will add required as well as the email directive. Now, email is not a HTML built on directive but still it’s a directive. **We need to keep in mind that the required is only acting as a selector for the angular directive here.**
6. email is another directive by angular which makes sure that email that is entered is valid email address.
7. Now, in the console in the form object the valid will be true if both the fields have a valid value.

Else it will be false.

1. It will check the validations at 2 levels i.e. the form level and the template structure value.
2. Now, if we check the form object in the console we will come to know that there are various classes that are added by angular i.e. ng-dirty, ng-touched, ng-valid; so angular dynamically adds some classes giving us information giving us the information about individual control here.

**Section 15: Lecture 180//Built in validators & using HTML 5 Validation**

1. Which Validators do ship with Angular?
2. Check out the Validators class: <https://angular.io/docs/ts/latest/api/forms/index/Validators-class.html> - these are all built-in validators, though that are the methods which actually get executed (and which you later can add when using the reactive approach).
3. For the template-driven approach, you need the directives. You can find out their names, by searching for "validator" in the official docs: <https://angular.io/api?type=directive> - everything marked with "D" is a directive and can be added to your template.
4. Additionally, you might also want to enable HTML5 validation (by default, Angular disables it). You can do so by adding the ngNativeValidate to a control in your template.

**Section 15: Lecture 181//TD: Using the form State**

1. In the last lecture we found out that the angular tracks the state of each control of the form, weather it valid or not and conditionally also adds there CSS classes.
2. Here we will disable the submit button if the form is not valid, which we will accomplish through property binding.
3. Now, we will check if the local reference is valid then we will let it enabled. We will use here f.valid i.e. local reference object dot valid method.
4. The second improvement is to take advantage of these CSS classes.
5. app.component.css:
6. .container {
7. margin-top: 30px;
8. }
9. input.ng-invalid.ng-touched{
10. border: 1px solid red;
11. }
12. app.component.html:
13. <div class="container">
14. <div class="row">
15. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
16. <form (ngSubmit)="onSubmit(f)" #f="ngForm">
17. <div id="user-data">
18. <div class="form-group">
19. <label for="username">Username</label>
20. <input
21. type="text"
22. id="username"
23. class="form-control"
24. ngModel
25. name="username"
26. required
27. >
28. </div>
29. <button class="btn btn-default" type="button">Suggest an Username</button>
30. <div class="form-group">
31. <label for="email">Mail</label>
32. <input
33. type="email"
34. id="email"
35. class="form-control"
36. ngModel
37. name = "email"
38. required
39. email
40. >
41. </div>
42. </div>
43. <div class="form-group">
44. <label for="secret">Secret Questions</label>
45. <select
46. id="secret"
47. class="form-control"
48. ngModel
49. name="secret"
50. >
51. <option value="pet">Your first Pet?</option>
52. <option value="teacher">Your first teacher?</option>
53. </select>
54. </div>
55. <button
56. class="btn btn-primary"
57. type="submit"
58. [disabled]="!f.valid"
59. >Submit</button>
60. </form>
61. </div>
62. </div>
63. </div>

**Section 15: Lecture 182//TD: Outputting Validation Error Messages**

1. In the last lecture we added form state handled by angular to improve the user experience.
2. Now, we will add some help text which would be visible if some invalid values is entered in the input.
3. Now, we can use a bootstrap class here i.e. help-block. Here we will add a local reference to the input element to display the help text in the case when an invalid value is entered. The local reference will connect input with span.
4. Here we will assign ngModel to form and not ngForm because we don’t want this for whole form but only for input type.
5. app.component.html:
6. <div class="container">
7. <div class="row">
8. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
9. <form (ngSubmit)="onSubmit(f)" #f="ngForm">
10. <div id="user-data">
11. <div class="form-group">
12. <label for="username">Username</label>
13. <input
14. type="text"
15. id="username"
16. class="form-control"
17. ngModel
18. name="username"
19. required
20. >
21. </div>
22. <button class="btn btn-default" type="button">Suggest an Username</button>
23. <div class="form-group">
24. <label for="email">Mail</label>
25. <input
26. type="email"
27. id="email"
28. class="form-control"
29. ngModel
30. name = "email"
31. required
32. email
33. #email="ngModel"
34. >
35. <span class="help-block" \*ngIf="!email.valid && email.touched">Please enter a valid email!</span>
36. </div>
37. </div>
38. <div class="form-group">
39. <label for="secret">Secret Questions</label>
40. <select
41. id="secret"
42. class="form-control"
43. ngModel
44. name="secret"
45. >
46. <option value="pet">Your first Pet?</option>
47. <option value="teacher">Your first teacher?</option>
48. </select>
49. </div>
50. <button
51. class="btn btn-primary"
52. type="submit"
53. [disabled]="!f.valid"
54. >Submit</button>
55. </form>
56. </div>
57. </div>
58. </div>

**Section 15: Lecture 183//TD: Set Default Values with ngModel Property Binding**

1. Now, we will add the default values which will be displayed, now, we will add some property binding also to the ngModel
2. app.component.ts:
3. import { Component, ViewChild } from '@angular/core';
4. import { NgForm } from '@angular/forms';
5. @Component({
6. selector: 'app-root',
7. templateUrl: './app.component.html',
8. styleUrls: ['./app.component.css']
9. })
10. export class AppComponent {
11. @ViewChild('f') signupForm: NgForm;
12. defaultQuestion='pet';
13. suggestUserName() {
14. const suggestedName = 'Superuser';
15. }
16. // onSubmit(form: NgForm){
17. // console.log(form);
18. // }
19. onSubmit(){
20. }
21. }
22. app.component.html:
23. <div class="container">
24. <div class="row">
25. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
26. <form (ngSubmit)="onSubmit(f)" #f="ngForm">
27. <div id="user-data">
28. <div class="form-group">
29. <label for="username">Username</label>
30. <input
31. type="text"
32. id="username"
33. class="form-control"
34. ngModel
35. name="username"
36. required
37. >
38. </div>
39. <button class="btn btn-default" type="button">Suggest an Username</button>
40. <div class="form-group">
41. <label for="email">Mail</label>
42. <input
43. type="email"
44. id="email"
45. class="form-control"
46. ngModel
47. name = "email"
48. required
49. email
50. #email="ngModel"
51. >
52. <span class="help-block" \*ngIf="!email.valid && email.touched">Please enter a valid email!</span>
53. </div>
54. </div>
55. <div class="form-group">
56. <label for="secret">Secret Questions</label>
57. <select
58. id="secret"
59. class="form-control"
60. [ngModel]="defaultQuestion"
61. name="secret"
62. >
63. <option value="pet">Your first Pet?</option>
64. <option value="teacher">Your first teacher?</option>
65. </select>
66. </div>
67. <button
68. class="btn btn-primary"
69. type="submit"
70. [disabled]="!f.valid"
71. >Submit</button>
72. </form>
73. </div>
74. </div>
75. </div>

**Section 15: Lecture 184//Using ngModel with Two-Way-Binding**

1. Now, we want to intently check something when the user enters in the input type.
2. For the question here we will enter a new form group – bootstrap style here. In this div we will enter a textarea, which we will name questionAnswer and will have 3 rows in it and then we will also enter ngModel to get what the user entered as a reply.
3. Now, here we want to output the answer to the above question instantly for which we will use 2 way binding on ngModel and bind it to the answer property here.
4. No binding to tell angular that the input is just a control
5. One way binding to give that control a default value.
6. Two way binding to intently output it or do whatever we want to do with that value.
7. app.component.ts
8. import { Component, ViewChild } from '@angular/core';
9. import { NgForm } from '@angular/forms';
10. @Component({
11. selector: 'app-root',
12. templateUrl: './app.component.html',
13. styleUrls: ['./app.component.css']
14. })
15. export class AppComponent {
16. @ViewChild('f') signupForm: NgForm;
17. defaultQuestion='pet';
18. answer='';
19. suggestUserName() {
20. const suggestedName = 'Superuser';
21. }
22. // onSubmit(form: NgForm){
23. // console.log(form);
24. // }
25. onSubmit(){
26. }
27. }
28. app.component.html:
29. <div class="container">
30. <div class="row">
31. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
32. <form (ngSubmit)="onSubmit(f)" #f="ngForm">
33. <div id="user-data">
34. <div class="form-group">
35. <label for="username">Username</label>
36. <input
37. type="text"
38. id="username"
39. class="form-control"
40. ngModel
41. name="username"
42. required
43. >
44. </div>
45. <button class="btn btn-default" type="button">Suggest an Username</button>
46. <div class="form-group">
47. <label for="email">Mail</label>
48. <input
49. type="email"
50. id="email"
51. class="form-control"
52. ngModel
53. name = "email"
54. required
55. email
56. #email="ngModel"
57. >
58. <span class="help-block" \*ngIf="!email.valid && email.touched">Please enter a valid email!</span>
59. </div>
60. </div>
61. <div class="form-group">
62. <label for="secret">Secret Questions</label>
63. <select
64. id="secret"
65. class="form-control"
66. [ngModel]="defaultQuestion"
67. name="secret"
68. >
69. <option value="pet">Your first Pet?</option>
70. <option value="teacher">Your first teacher?</option>
71. </select>
72. </div>
73. <div class="form-group">
74. <textarea name="questionAnswer" rows="3" [(ngModel)]="answer" class="form-control"></textarea>
75. </div>
76. <p>Your reply: {{answer}}</p>
77. <button
78. class="btn btn-primary"
79. type="submit"
80. [disabled]="!f.valid"
81. >Submit</button>
82. </form>
83. </div>
84. </div>
85. </div>

**Section 15: Lecture 185//TD: Grouping Form Controls**

1. Now, when we submit the form, we want to group somethings. Let’s say we want to group the secret into question answer, and username & email just have some structure in our object because they are very big form.
2. Then we would also must validate the group form input and that is easy to handle in the template driven approach.
3. Now, we will put ngModelGroup directive on the group div. ngModelGroup needs to be assigned to a string. For example, here we will put key name for this group as userData.
4. Now, we will have another object named userData in the form object which contains the email and username.
5. We can get the access to the JavaScript representation by adding the local reference to the tag which holds ngGroupModel directive.
6. app.component.html
7. <div class="container">
8. <div class="row">
9. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
10. <form (ngSubmit)="onSubmit(f)" #f="ngForm">
11. <div id="user-data" ngMoedelGroup="userData" #userData="ngMoedelGroup">
12. <div class="form-group">
13. <label for="username">Username</label>
14. <input
15. type="text"
16. id="username"
17. class="form-control"
18. ngModel
19. name="username"
20. required
21. >
22. </div>
23. <button class="btn btn-default" type="button">Suggest an Username</button>
24. <div class="form-group">
25. <label for="email">Mail</label>
26. <input
27. type="email"
28. id="email"
29. class="form-control"
30. ngModel
31. name = "email"
32. required
33. email
34. #email="ngModel"
35. >
36. <span class="help-block" \*ngIf="!email.valid && email.touched">Please enter a valid email!</span>
37. </div>
38. </div>
39. <p \*ngIf="!userData.valid && userData.touched">User Data is Invalid!</p>
40. <div class="form-group">
41. <label for="secret">Secret Questions</label>
42. <select
43. id="secret"
44. class="form-control"
45. [ngModel]="defaultQuestion"
46. name="secret"
47. >
48. <option value="pet">Your first Pet?</option>
49. <option value="teacher">Your first teacher?</option>
50. </select>
51. </div>
52. <div class="form-group">
53. <textarea name="questionAnswer" rows="3" [(ngModel)]="answer" class="form-control"></textarea>
54. </div>
55. <p>Your reply: {{answer}}</p>
56. <button
57. class="btn btn-primary"
58. type="submit"
59. [disabled]="!f.valid"
60. >Submit</button>
61. </form>
62. </div>
63. </div>
64. </div>

**Section 15: Lecture 186//Handling Radio Buttons**

1. Now, to implement the radio button lets create a new array in the .ts file – for now it will contain 2 values i.e. male and female.
2. We will name it radio and use class as radio to use the bootstrap class for radio. We will place ngModel on it to make it a control and we will prepopulate it to set the value of this button equal to gender i.e. the variable of our \*ngFor loop.
3. app.component.html:
4. <div class="container">
5. <div class="row">
6. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
7. <form (ngSubmit)="onSubmit(f)" #f="ngForm">
8. <div id="user-data" ngModelGroup="userData" #userData="ngModelGroup">
9. <div class="form-group">
10. <label for="username">Username</label>
11. <input
12. type="text"
13. id="username"
14. class="form-control"
15. ngModel
16. name="username"
17. required
18. >
19. </div>
20. <button class="btn btn-default" type="button">Suggest an Username</button>
21. <div class="form-group">
22. <label for="email">Mail</label>
23. <input
24. type="email"
25. id="email"
26. class="form-control"
27. ngModel
28. name = "email"
29. required
30. email
31. #email="ngModel"
32. >
33. <span class="help-block" \*ngIf="!email.valid && email.touched">Please enter a valid email!</span>
34. </div>
35. </div>
36. <p \*ngIf="!userData.valid && userData.touched">User Data is Invalid!</p>
37. <div class="form-group">
38. <label for="secret">Secret Questions</label>
39. <select
40. id="secret"
41. class="form-control"
42. [ngModel]="defaultQuestion"
43. name="secret"
44. >
45. <option value="pet">Your first Pet?</option>
46. <option value="teacher">Your first teacher?</option>
47. </select>
48. </div>
49. <div class="form-group">
50. <textarea name="questionAnswer" rows="3" [(ngModel)]="answer" class="form-control"></textarea>
51. </div>
52. <p>Your reply: {{answer}}</p>
53. <div class="radio" \*ngFor="let gender of genders">
54. <label>
55. <input type="radio"
56. name="gender"
57. ngModel
58. [value]="gender"
59. required
60. >{{ gender }}
61. </label>
62. </div>
63. <button
64. class="btn btn-primary"
65. type="submit"
66. [disabled]="!f.valid"
67. >Submit</button>
68. </form>
69. </div>
70. </div>
71. </div>

**Section 15: Lecture 187//TD: Setting and Patching Form Values**

1. Here we will make our suggest a user name button, it doesn’t work as of now. So, we will make it work for this purpose only we have added the suggest username method in the .ts file, as of now it contains the default username.
2. By clicking this button we will populate our username input on the page. Now, as we know that we have access to that form through @ViewChild in the typescript file.
3. It would be nice if we could set the values of one of our inputs and turns out we can for this we have got 2 different methods.
4. The 1st one would be on our signupForm i.e. this.signupForm.setValue(); - this allows us to set the value of the whole form.
5. And here we need to pass a JavaScript object exactly representing our form.
6. So, we will setup our userData where we have a username field. This is not the bext approach as it can overwrite the value in the userName input.
7. Better would be to access the form object from the signupForm object.
8. Now, when we will click the button, then only username will get populated and all the other values will remain untouched.
9. app.component.ts:
10. import { Component, ViewChild } from '@angular/core';
11. import { NgForm } from '@angular/forms';
12. @Component({
13. selector: 'app-root',
14. templateUrl: './app.component.html',
15. styleUrls: ['./app.component.css']
16. })
17. export class AppComponent {
18. @ViewChild('f') signupForm: NgForm;
19. defaultQuestion='pet';
20. answer='';
21. genders=['male','female'];
22. suggestUserName() {
23. const suggestedName = 'Superuser';
24. // this.signupForm.setValue({
25. // userData:{
26. // username: suggestedName,
27. // email: '',
28. // },
29. // secret: 'pet',
30. // questionAnswer:'',
31. // gender: 'male'
32. // });
33. this.signupForm.form.patchValue({userData:{
34. username: suggestedName
35. }});
36. }
37. // onSubmit(form: NgForm){
38. // console.log(form);
39. // }
40. onSubmit(){
41. }
42. }
43. app.component.html:
44. <div class="container">
45. <div class="row">
46. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
47. <form (ngSubmit)="onSubmit(f)" #f="ngForm">
48. <div id="user-data" ngModelGroup="userData" #userData="ngModelGroup">
49. <div class="form-group">
50. <label for="username">Username</label>
51. <input
52. type="text"
53. id="username"
54. class="form-control"
55. ngModel
56. name="username"
57. required
58. >
59. </div>
60. <button
61. class="btn btn-default"
62. type="button"
63. (click)="suggestUserName()"
64. >Suggest an Username</button>
65. <div class="form-group">
66. <label for="email">Mail</label>
67. <input
68. type="email"
69. id="email"
70. class="form-control"
71. ngModel
72. name = "email"
73. required
74. email
75. #email="ngModel"
76. >
77. <span class="help-block" \*ngIf="!email.valid && email.touched">Please enter a valid email!</span>
78. </div>
79. </div>
80. <p \*ngIf="!userData.valid && userData.touched">User Data is Invalid!</p>
81. <div class="form-group">
82. <label for="secret">Secret Questions</label>
83. <select
84. id="secret"
85. class="form-control"
86. [ngModel]="defaultQuestion"
87. name="secret"
88. >
89. <option value="pet">Your first Pet?</option>
90. <option value="teacher">Your first teacher?</option>
91. </select>
92. </div>
93. <div class="form-group">
94. <textarea name="questionAnswer" rows="3" [(ngModel)]="answer" class="form-control"></textarea>
95. </div>
96. <p>Your reply: {{answer}}</p>
97. <div class="radio" \*ngFor="let gender of genders">
98. <label>
99. <input type="radio"
100. name="gender"
101. ngModel
102. [value]="gender"
103. required
104. >{{ gender }}
105. </label>
106. </div>
107. <button
108. class="btn btn-primary"
109. type="submit"
110. [disabled]="!f.valid"
111. >Submit</button>
112. </form>
113. </div>
114. </div>
115. </div>

**Section 15: Lecture 188//TD: Using Form Data**

1. We will output all the above fields in the bottom of the page once the form is submitted and we will fetch the data to be displayed from our submitted form.
2. Now, we will a new property in the app.component.ts i.e. user
3. Now, we will put \*ngIf in the row div here to check if the form was submitted
4. App.component.html:
5. <div class="container">
6. <div class="row">
7. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
8. <form (ngSubmit)="onSubmit(f)" #f="ngForm">
9. <div id="user-data" ngModelGroup="userData" #userData="ngModelGroup">
10. <div class="form-group">
11. <label for="username">Username</label>
12. <input
13. type="text"
14. id="username"
15. class="form-control"
16. ngModel
17. name="username"
18. required
19. >
20. </div>
21. <button
22. class="btn btn-default"
23. type="button"
24. (click)="suggestUserName()"
25. >Suggest an Username</button>
26. <div class="form-group">
27. <label for="email">Mail</label>
28. <input
29. type="email"
30. id="email"
31. class="form-control"
32. ngModel
33. name = "email"
34. required
35. email
36. #email="ngModel"
37. >
38. <span class="help-block" \*ngIf="!email.valid && email.touched">Please enter a valid email!</span>
39. </div>
40. </div>
41. <p \*ngIf="!userData.valid && userData.touched">User Data is Invalid!</p>
42. <div class="form-group">
43. <label for="secret">Secret Questions</label>
44. <select
45. id="secret"
46. class="form-control"
47. [ngModel]="defaultQuestion"
48. name="secret"
49. >
50. <option value="pet">Your first Pet?</option>
51. <option value="teacher">Your first teacher?</option>
52. </select>
53. </div>
54. <div class="form-group">
55. <textarea name="questionAnswer" rows="3" [(ngModel)]="answer" class="form-control"></textarea>
56. </div>
57. <p>Your reply: {{answer}}</p>
58. <div class="radio" \*ngFor="let gender of genders">
59. <label>
60. <input type="radio"
61. name="gender"
62. ngModel
63. [value]="gender"
64. required
65. >{{ gender }}
66. </label>
67. </div>
68. <button
69. class="btn btn-primary"
70. type="submit"
71. [disabled]="!f.valid"
72. >Submit</button>
73. </form>
74. </div>
75. </div>
76. <hr>
77. <div class="row" \*ngIf="submitted">
78. <div class="col-xs-12">
79. <h3>Your Data</h3>
80. <p>Username: {{ user.username }}</p>
81. <p>Mail: {{ user.email }}</p>
82. <p>Secret Question: Your First {{ user.secretQuestion }}</p>
83. <p>Answer: {{ user.answer }}</p>
84. <p>Gender: {{ user.gender }}</p>
85. </div>
86. </div>
87. </div>
88. app.component.ts:
89. import { Component, ViewChild } from '@angular/core';
90. import { NgForm } from '@angular/forms';
91. @Component({
92. selector: 'app-root',
93. templateUrl: './app.component.html',
94. styleUrls: ['./app.component.css']
95. })
96. export class AppComponent {
97. @ViewChild('f') signupForm: NgForm;
98. defaultQuestion='pet';
99. answer='';
100. genders=['male','female'];
101. user={
102. username: '',
103. email: '',
104. secretQuestion: '',
105. answer: '',
106. gender: ''
107. };
108. suggestUserName() {
109. const suggestedName = 'Superuser';
110. // this.signupForm.setValue({
111. // userData:{
112. // username: suggestedName,
113. // email: '',
114. // },
115. // secret: 'pet',
116. // questionAnswer:'',
117. // gender: 'male'
118. // });
119. this.signupForm.form.patchValue({userData:{
120. username: suggestedName
121. }});
122. }
123. // onSubmit(form: NgForm){
124. // console.log(form);
125. // }
126. onSubmit(){
127. this.user.username = this.signupForm.value.userData.username;
128. this.user.email = this.signupForm.value.userData.email;
129. this.user.secretQuestion = this.signupForm.value.secret;
130. this.user.answer = this.signupForm.value.questionAnswer;
131. this.user.gender = this.signupForm.value.gender;
132. }
133. }

**Section15: Lecture 189 //Resetting Forms**

1. Here we will add code to reset the form once its displayed below, so, we will call reset method on signupForm.
2. It will also reset all the states such as valid, touched etc.
3. This was the template driven approach now for the rest of the module we will use the reactive approach which is even more flexible.
4. app.component.ts:
5. import { Component, ViewChild } from '@angular/core';
6. import { NgForm } from '@angular/forms';
7. @Component({
8. selector: 'app-root',
9. templateUrl: './app.component.html',
10. styleUrls: ['./app.component.css']
11. })
12. export class AppComponent {
13. @ViewChild('f') signupForm: NgForm;
14. defaultQuestion='pet';
15. answer='';
16. genders=['male','female'];
17. user={
18. username: '',
19. email: '',
20. secretQuestion: '',
21. answer: '',
22. gender: ''
23. };
24. suggestUserName() {
25. const suggestedName = 'Superuser';
26. // this.signupForm.setValue({
27. // userData:{
28. // username: suggestedName,
29. // email: '',
30. // },
31. // secret: 'pet',
32. // questionAnswer:'',
33. // gender: 'male'
34. // });
35. this.signupForm.form.patchValue({userData:{
36. username: suggestedName
37. }});
38. }
39. // onSubmit(form: NgForm){
40. // console.log(form);
41. // }
42. onSubmit(){
43. this.user.username = this.signupForm.value.userData.username;
44. this.user.email = this.signupForm.value.userData.email;
45. this.user.secretQuestion = this.signupForm.value.secret;
46. this.user.answer = this.signupForm.value.questionAnswer;
47. this.user.gender = this.signupForm.value.gender;
49. this.signupForm.reset();
50. }
51. }

**Section 15: Lecture 190// Introduction to the reactive Approach**

1. This allows us to configure our form in the greater detail, here we create the for programmatically in typescript
2. 

**Section 15: Lecture 191//Reactive: Setup**

1. So, as we know the form will now be created in our typescript file, so we will start working on our typescript file. Here we will create a new property which will hold our form in the end.
2. So, we will name it here signup form – this would be of the type FormGroup.
3. In the template driven approach we imported ngForm which automatically wrapped a FormGroup in the end, because in the angular in the end is the group of controls and this is what a FormGroup holds.
4. Therefore, the overall form is also in the end a FormGroup.
5. For connecting our programmatically generated form without template form in the end, we need to add something in the app.module.ts
6. We don’t need the FormModule - this is required for the template driven approach, instead we need to use the ReactiveFormsModule.
7. ReactiveFormsModule this contains all the library for us to create the reactive – programmatical forms in our app.
8. In the next lecture will create the actual form.

**Section 15: Lecture 193//Reactive: Creating a Form in Code**

1. So, now we will use onInit() lifecycle hook to write the code for the dynamic form and then will assign to the property we created of that form i.e. signupForm.
2. In this ngOnInit() method we will initialize our form before rendering the template of-course. Here we need to make sure that we use the lifecycle hook which is called before the template is rendered.
3. Now, we have created the signup form in the ngOnInit() lifecycle hook. Now we will add some controls in it. The controls will be the key-value pair in the object that we pass to this overall form group.
4. The username control will be of the type FormControl(), now this accepts a couple of argumrnts. The first argument is the insitial state/value of this control we could say. The second argument would be the single validator or an array of validators, that we want to apply to this control and the third argument would be the potential asynchronous validators.
5. app.component.ts
6. import { Component, OnInit } from '@angular/core';
7. import { FormGroup, FormControl } from '@angular/forms';
8. @Component({
9. selector: 'app-root',
10. templateUrl: './app.component.html',
11. styleUrls: ['./app.component.css']
12. })
13. export class AppComponent implements OnInit {
14. genders = ['male', 'female'];
15. signupForm: FormGroup;
16. ngOnInit(){
17. this.signupForm = new FormGroup({
18. 'username' : new FormControl(null),
19. 'email' : new FormControl(null),
20. 'gender': new FormControl('male')
21. });
22. }
24. }

**Section 15: Lecture 193//Reactive: Syncing HTML and Form**

1. In the last lecture we created our form in typescript, the issue is it’s nice to have it here, but our actual form lays in the HTML template.
2. Now, we need to synchronize our HTML template and our form. As of now HTML doesn’t know about our form.
3. Now, we will use some directives to add our form to HTML template. Now, the first directive which we need to have is formGroup directive we will add this directive in the property binding. i.e. [formGroup].
4. This will inform angular that please use my formGroup, we need to use property binding here because we will pass our form to it using property binding.
5. So, here we should reference our signupForm i.e. the property we created here which stores our form.
6. Now, still we need to inform angular that which controls would be connected to the which input in the template code, for this we get another directive. Here on the input type we add a formControlName directive - this is used to tell angular that what is the name of this input in my form.
7. Now, this input will be connected to the control named username. Now, if we look in the console – we observe that angular internal classes are added on each of the input type.
8. Now, the next step is to submit the form to check how this works.
9. app.component.html
10. <div class="container">
11. <div class="row">
12. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
13. <form [formGroup]="signupForm">
14. <div class="form-group">
15. <label for="username">Username</label>
16. <input
17. type="text"
18. id="username"
19. formControlName="username"
20. class="form-control">
21. </div>
22. <div class="form-group">
23. <label for="email">email</label>
24. <input
25. type="text"
26. id="email"
27. formControlName="email"
28. class="form-control">
29. </div>
30. <div class="radio" \*ngFor="let gender of genders">
31. <label>
32. <input
33. type="radio"
34. formControlName="gender"
35. [value]="gender">{{ gender }}
36. </label>
37. </div>
38. <button class="btn btn-primary" type="submit">Submit</button>
39. </form>
40. </div>
41. </div>
42. </div>

**Section 15: Lecture 194//Reactive: Submitting the Form**

1. In the last lecture we synchronized our HTML form and our typescript form. Now, we will look at how to submit this form.
2. We used ngSubmit in template driven approach, here also we will us the same approach and we will assign it to the onSubmit() method in our typescript.
3. Now the difference in template driven and reactive approach is that we don’t need to get the form via local reference in the reactive approach, that actually will not work because we are not actually using the angular’s order creation mechanism and also we don’t need to get this form because we created this form on our own - so, we already got access to it here in our typescript code (Actually, in every method in the typescript code).
4. app.component.html
5. <div class="container">
6. <div class="row">
7. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
8. <form [formGroup]="signupForm" (ngSubmit)="onSubmit()">
9. <div class="form-group">
10. <label for="username">Username</label>
11. <input
12. type="text"
13. id="username"
14. formControlName="username"
15. class="form-control">
16. </div>
17. <div class="form-group">
18. <label for="email">email</label>
19. <input
20. type="text"
21. id="email"
22. formControlName="email"
23. class="form-control">
24. </div>
25. <div class="radio" \*ngFor="let gender of genders">
26. <label>
27. <input
28. type="radio"
29. formControlName="gender"
30. [value]="gender">{{ gender }}
31. </label>
32. </div>
33. <button class="btn btn-primary" type="submit">Submit</button>
34. </form>
35. </div>
36. </div>
37. </div>

5. app.component.ts:

import { Component, OnInit } from '@angular/core';

import { FormGroup, FormControl } from '@angular/forms';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

genders = ['male', 'female'];

signupForm: FormGroup;

ngOnInit(){

this.signupForm = new FormGroup({

'username' : new FormControl(null),

'email' : new FormControl(null),

'gender': new FormControl('male')

});

}

onSubmit( ){

console.log(this.signupForm);

}

}

**Section 15: Lecture 195//Reactive: Adding Validation**

1. Now, we are able to submit the form, lets also add some validations to it. In the template driven approach we simply add Required simply to make the field required in the template, however, it does not work here.
2. The formControl(), takes another argument i.e. validators – so, we can either only pass one validator here – ex. Valicator.required. Here we can also pass multiple validators by passing an array of validators.
3. app.component.ts:
4. import { Component, OnInit } from '@angular/core';
5. import { FormGroup, FormControl, Validators } from '@angular/forms';
6. @Component({
7. selector: 'app-root',
8. templateUrl: './app.component.html',
9. styleUrls: ['./app.component.css']
10. })
11. export class AppComponent implements OnInit {
12. genders = ['male', 'female'];
13. signupForm: FormGroup;
14. ngOnInit(){
15. this.signupForm = new FormGroup({
16. 'username' : new FormControl(null, Validators.required),
17. 'email' : new FormControl(null, [Validators.required, Validators.email]),
18. 'gender': new FormControl('male')
19. });
20. }
21. onSubmit( ){
22. console.log(this.signupForm);
23. }
25. }

**Section 15: Lecture 196//Getting Access to controls**

1. Now, in the reactive approach if there is some validation error in the input then we want to display a message, we need a bit different approach here.
2. So, now we will have access to the controls in the warning messages which we are adding using span.
3. app.component.css
4. .container {
5. margin-top: 30px;
6. }
7. input.ng-invalid.ng-touched{
8. border: 1px solid red;
9. }
10. app.coponent.html:
11. <div class="container">
12. <div class="row">
13. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
14. <form [formGroup]="signupForm" (ngSubmit)="onSubmit()">
15. <div class="form-group">
16. <label for="username">Username</label>
17. <input
18. type="text"
19. id="username"
20. formControlName="username"
21. class="form-control">
22. <span
23. \*ngIf="!signupForm.get('username').valid && signupForm.get('username').touched"
24. class="help-block">Please enter a valid username!</span>
25. </div>
26. <div class="form-group">
27. <label for="email">email</label>
28. <input
29. type="text"
30. id="email"
31. formControlName="email"
32. class="form-control">
33. <span
34. \*ngIf="!signupForm.get('email').valid && signupForm.get('email').touched"
35. class="help-block">Please enter a valid email!</span>
36. </div>
37. <div class="radio" \*ngFor="let gender of genders">
38. <label>
39. <input
40. type="radio"
41. formControlName="gender"
42. [value]="gender">{{ gender }}
43. </label>
44. </div>
45. <span
46. \*ngIf="!signupForm.valid && signupForm.touched"
47. class="help-block">Please enter a valid data!</span>
48. <button class="btn btn-primary" type="submit">Submit</button>
49. </form>
50. </div>
51. </div>
52. </div>

**Section 15: Lecture 197//Reactive: Grouping Controls**

1. In the last lecture we discussed that get() method also takes the path – it means that we can specify a path here, because we have a nested form.
2. Lets say username and email are a part of the group called the formGroup() is not only used at the outer form but a form group can contain another form group in the inside form.
3. Now, we can add our form controls inside of this group, so, now we have a nested form. Now on this group we place formGroupName directive
4. app.component.html
5. <div class="container">
6. <div class="row">
7. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
8. <form [formGroup]="signupForm" (ngSubmit)="onSubmit()">
9. <div formGroupName="userData">
10. <div class="form-group">
11. <label for="username">Username</label>
12. <input
13. type="text"
14. id="username"
15. formControlName="username"
16. class="form-control">
17. <span
18. \*ngIf="!signupForm.get('userData.username').valid && signupForm.get('userData.username').touched"
19. class="help-block">Please enter a valid username!</span>
20. </div>
21. <div class="form-group">
22. <label for="email">email</label>
23. <input
24. type="text"
25. id="email"
26. formControlName="email"
27. class="form-control">
28. <span
29. \*ngIf="!signupForm.get('userData.email').valid && signupForm.get('userData.email').touched"
30. class="help-block">Please enter a valid email!</span>
31. </div>
32. </div>
34. <div class="radio" \*ngFor="let gender of genders">
35. <label>
36. <input
37. type="radio"
38. formControlName="gender"
39. [value]="gender">{{ gender }}
40. </label>
41. </div>
42. <span
43. \*ngIf="!signupForm.valid && signupForm.touched"
44. class="help-block">Please enter a valid data!</span>
45. <button class="btn btn-primary" type="submit">Submit</button>
46. </form>
47. </div>
48. </div>
49. </div>

5. app.component.ts

import { Component, OnInit } from '@angular/core';

import { FormGroup, FormControl, Validators } from '@angular/forms';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

genders = ['male', 'female'];

signupForm: FormGroup;

ngOnInit(){

this.signupForm = new FormGroup({

'userData': new FormGroup({

'username' : new FormControl(null, Validators.required),

'email' : new FormControl(null, [Validators.required, Validators.email])

}),

'gender': new FormControl('male')

});

}

onSubmit( ){

console.log(this.signupForm);

}

}

**Section 15: Lecture 198//Reactive: Arrays of Form Controls (FormArray)**

1. We will now add a new area on our form – in this page we will add the radio buttons here.
2. Let’s add a new div and in this div we want the user to add dynamically the form controls.
3. A FormArray holds an array of controls, now we need to sync our typescript code with HTML code.
4. This we will do by formArrayName directive. We need property binding when we are passing a variable to the property and we can assign without property binding when we are passing a string.
5. App.component.html
6. <div class="container">
7. <div class="row">
8. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
9. <form [formGroup]="signupForm" (ngSubmit)="onSubmit()">
10. <div formGroupName="userData">
11. <div class="form-group">
12. <label for="username">Username</label>
13. <input
14. type="text"
15. id="username"
16. formControlName="username"
17. class="form-control">
18. <span
19. \*ngIf="!signupForm.get('userData.username').valid && signupForm.get('userData.username').touched"
20. class="help-block">Please enter a valid username!</span>
21. </div>
22. <div class="form-group">
23. <label for="email">email</label>
24. <input
25. type="text"
26. id="email"
27. formControlName="email"
28. class="form-control">
29. <span
30. \*ngIf="!signupForm.get('userData.email').valid && signupForm.get('userData.email').touched"
31. class="help-block">Please enter a valid email!</span>
32. </div>
33. </div>
35. <div class="radio" \*ngFor="let gender of genders">
36. <label>
37. <input
38. type="radio"
39. formControlName="gender"
40. [value]="gender">{{ gender }}
41. </label>
42. </div>
43. <div formArrayName="hobbies">
44. <h4>Your Hobbies</h4>
45. <button class="btn btn-def" type="button" (click)="onAddHobby()">Add Hobby</button>
46. <div class="form-group"
47. \*ngFor="let hobbyControl of signupForm.get('hobbies').controls; let i = index"
48. >
49. <input type="text" class="form-control" [formControlName]="i">
50. </div>
51. </div>
52. <span
53. \*ngIf="!signupForm.valid && signupForm.touched"
54. class="help-block">Please enter a valid data!</span>
55. <button class="btn btn-primary" type="submit">Submit</button>
56. </form>
57. </div>
58. </div>
59. </div>

6. app.component.ts:

import { Component, OnInit } from '@angular/core';

import { FormGroup, FormControl, Validators, FormArray } from '@angular/forms';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

genders = ['male', 'female'];

signupForm: FormGroup;

ngOnInit(){

this.signupForm = new FormGroup({

'userData': new FormGroup({

'username' : new FormControl(null, Validators.required),

'email' : new FormControl(null, [Validators.required, Validators.email])

}),

'gender': new FormControl('male'),

'hobbies' : new FormArray([])

});

}

onSubmit( ){

console.log(this.signupForm);

}

onAddHobby(){

const control = new FormControl(null, Validators.required);

(<FormArray>this.signupForm.get('hobbies')).push(control);

}

}

**Section 15: Lecture 199//Reactive: Creating Custom Validators**

1. Here we will create our own validators. Here, we will take the scenario where there would be some usernames which we don’t want to allow.
2. A validator is just a function which angular executes once its checks the validity of the form control. And it check the validity whenever you change the control.
3. app.compnent.ts
4. import { Component, OnInit } from '@angular/core';
5. import { FormGroup, FormControl, Validators, FormArray } from '@angular/forms';
6. @Component({
7. selector: 'app-root',
8. templateUrl: './app.component.html',
9. styleUrls: ['./app.component.css']
10. })
11. export class AppComponent implements OnInit {
12. genders = ['male', 'female'];
13. signupForm: FormGroup;
14. forbiddenUsernames = ['Chris','Anna'];
15. ngOnInit(){
16. this.signupForm = new FormGroup({
17. 'userData': new FormGroup({
18. 'username' : new FormControl(null, [Validators.required, this.forbiddenNames.bind(this)]),
19. 'email' : new FormControl(null, [Validators.required, Validators.email])
20. }),
21. 'gender': new FormControl('male'),
22. 'hobbies' : new FormArray([])
23. });
24. }
25. onSubmit( ){
26. console.log(this.signupForm);
27. }
28. onAddHobby(){
29. const control = new FormControl(null, Validators.required);
30. (<FormArray>this.signupForm.get('hobbies')).push(control);
31. }
32. forbiddenNames(control: FormControl): {[s: string]: boolean}{
33. if(this.forbiddenUsernames.indexOf(control.value) !== -1){
34. return {'nameIsForbidden': true}
35. }
36. return null;
37. }
39. }

**Section 15: Lecture 200//Reactive: Using Error Codes**

1. Now, here if check in the form control object in the console – as we expend it, we will observe one object named error – on this object we will be able to see the key value pair that our validator is returning as we saw in the previous lecture.
2. Now, we will fine tune this error message in our HTML code
3. app.component.html
4. <div class="container">
5. <div class="row">
6. <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
7. <form [formGroup]="signupForm" (ngSubmit)="onSubmit()">
8. <div formGroupName="userData">
9. <div class="form-group">
10. <label for="username">Username</label>
11. <input
12. type="text"
13. id="username"
14. formControlName="username"
15. class="form-control">
16. <span
17. \*ngIf="!signupForm.get('userData.username').valid && signupForm.get('userData.username').touched"
18. class="help-block">
19. <span \*ngIf="signupForm.get('userData.username').errors['nameIsForbidden']">
20. This name is Invalid!</span>
21. <span \*ngIf="signupForm.get('userData.username').errors['required']">
22. This field is required!</span>
23. </span>
24. </div>
25. <div class="form-group">
26. <label for="email">email</label>
27. <input
28. type="text"
29. id="email"
30. formControlName="email"
31. class="form-control">
32. <span
33. \*ngIf="!signupForm.get('userData.email').valid && signupForm.get('userData.email').touched"
34. class="help-block">Please enter a valid email!</span>
35. </div>
36. </div>
38. <div class="radio" \*ngFor="let gender of genders">
39. <label>
40. <input
41. type="radio"
42. formControlName="gender"
43. [value]="gender">{{ gender }}
44. </label>
45. </div>
46. <div formArrayName="hobbies">
47. <h4>Your Hobbies</h4>
48. <button class="btn btn-def" type="button" (click)="onAddHobby()">Add Hobby</button>
49. <div class="form-group"
50. \*ngFor="let hobbyControl of signupForm.get('hobbies').controls; let i = index"
51. >
52. <input type="text" class="form-control" [formControlName]="i">
53. </div>
54. </div>
55. <span
56. \*ngIf="!signupForm.valid && signupForm.touched"
57. class="help-block">Please enter a valid data!</span>
58. <button class="btn btn-primary" type="submit">Submit</button>
59. </form>
60. </div>
61. </div>
62. </div>

**Section 15: Lecture 201//Reactive: Creating a custom Async Validator**

1. We also need asynchronous validators which are able to wait for the response from the server, before returning true or false I.e. isInvalid or not.
2. So, in this validator we will not return object containing error and a Boolean. Here we will return a promise or an observable which wraps anything.
3. Now, we will not pass this validator in the second parameter array of the FormControl; we will add it at the 3rd parameter.
4. app.component.ts:
5. import { Component, OnInit } from '@angular/core';
6. import { FormGroup, FormControl, Validators, FormArray } from '@angular/forms';
7. import { Observable } from 'rxjs';
8. @Component({
9. selector: 'app-root',
10. templateUrl: './app.component.html',
11. styleUrls: ['./app.component.css']
12. })
13. export class AppComponent implements OnInit {
14. genders = ['male', 'female'];
15. signupForm: FormGroup;
16. forbiddenUsernames = ['Chris','Anna'];
17. ngOnInit(){
18. this.signupForm = new FormGroup({
19. 'userData': new FormGroup({
20. 'username' : new FormControl(null, [Validators.required, this.forbiddenNames.bind(this)]),
21. 'email' : new FormControl(null, [Validators.required, Validators.email], this.forbiddenEmails)
22. }),
23. 'gender': new FormControl('male'),
24. 'hobbies' : new FormArray([])
25. });
26. }
27. onSubmit( ){
28. console.log(this.signupForm);
29. }
30. onAddHobby(){
31. const control = new FormControl(null, Validators.required);
32. (<FormArray>this.signupForm.get('hobbies')).push(control);
33. }
34. forbiddenNames(control: FormControl): {[s: string]: boolean}{
35. if(this.forbiddenUsernames.indexOf(control.value) !== -1){
36. return {'nameIsForbidden': true}
37. }
38. return null;
39. }
40. forbiddenEmails(control: FormControl): Promise<any>|Observable<any>{
41. const promise = new Promise<any>((resolve, reject)=> {
42. setTimeout(()=>{
43. if(control.value==='test@test.com'){
44. resolve({'emailIsForbidden': true});
45. }else{
46. resolve(null);
47. }
48. },1500);
49. });
50. return promise;
51. }
53. }

**Section 15: Lecture 202//Reactive: Reacting to status or value changes**

1. We can also track the form status and value.
2. app.component.ts:
3. import { Component, OnInit } from '@angular/core';
4. import { FormGroup, FormControl, Validators, FormArray } from '@angular/forms';
5. import { Observable } from 'rxjs';
6. @Component({
7. selector: 'app-root',
8. templateUrl: './app.component.html',
9. styleUrls: ['./app.component.css']
10. })
11. export class AppComponent implements OnInit {
12. genders = ['male', 'female'];
13. signupForm: FormGroup;
14. forbiddenUsernames = ['Chris','Anna'];
15. ngOnInit(){
16. this.signupForm = new FormGroup({
17. 'userData': new FormGroup({
18. 'username' : new FormControl(null, [Validators.required, this.forbiddenNames.bind(this)]),
19. 'email' : new FormControl(null, [Validators.required, Validators.email], this.forbiddenEmails)
20. }),
21. 'gender': new FormControl('male'),
22. 'hobbies' : new FormArray([])
23. });
24. // this.signupForm.valueChanges.subscribe(
25. // (value)=> console.log(value)
26. // );
27. this.signupForm.statusChanges.subscribe(
28. (status)=> console.log(status)
29. );
30. }
31. onSubmit( ){
32. console.log(this.signupForm);
33. }
34. onAddHobby(){
35. const control = new FormControl(null, Validators.required);
36. (<FormArray>this.signupForm.get('hobbies')).push(control);
37. }
38. forbiddenNames(control: FormControl): {[s: string]: boolean}{
39. if(this.forbiddenUsernames.indexOf(control.value) !== -1){
40. return {'nameIsForbidden': true}
41. }
42. return null;
43. }
44. forbiddenEmails(control: FormControl): Promise<any>|Observable<any>{
45. const promise = new Promise<any>((resolve, reject)=> {
46. setTimeout(()=>{
47. if(control.value==='test@test.com'){
48. resolve({'emailIsForbidden': true});
49. }else{
50. resolve(null);
51. }
52. },1500);
53. });
54. return promise;
55. }
57. }

`

**Section 15: Lecture 203//Reactive: Setting and Patching Values**

1. Not only we can listen to the updates, we can also update the form itself
2. app.component.ts:
3. import { Component, OnInit } from '@angular/core';
4. import { FormGroup, FormControl, Validators, FormArray } from '@angular/forms';
5. import { Observable } from 'rxjs';
6. @Component({
7. selector: 'app-root',
8. templateUrl: './app.component.html',
9. styleUrls: ['./app.component.css']
10. })
11. export class AppComponent implements OnInit {
12. genders = ['male', 'female'];
13. signupForm: FormGroup;
14. forbiddenUsernames = ['Chris','Anna'];
15. ngOnInit(){
16. this.signupForm = new FormGroup({
17. 'userData': new FormGroup({
18. 'username' : new FormControl(null, [Validators.required, this.forbiddenNames.bind(this)]),
19. 'email' : new FormControl(null, [Validators.required, Validators.email], this.forbiddenEmails)
20. }),
21. 'gender': new FormControl('male'),
22. 'hobbies' : new FormArray([])
23. });
24. // this.signupForm.valueChanges.subscribe(
25. // (value)=> console.log(value)
26. // );
27. this.signupForm.statusChanges.subscribe(
28. (status)=> console.log(status)
29. );
30. // this.signupForm.setValue({
31. // 'userData':{
32. // 'username': 'Sukhmeet',
33. // 'email': 'sukhmeet@test.com'
34. // },
35. // 'gender': 'male',
36. // 'hobbies': []
37. // });
38. this.signupForm.patchValue({
39. 'userData':{
40. 'username': 'Sukhmeet',
42. }
43. });
44. }
45. onSubmit( ){
46. console.log(this.signupForm);
47. this.signupForm.reset();
48. }
49. onAddHobby(){
50. const control = new FormControl(null, Validators.required);
51. (<FormArray>this.signupForm.get('hobbies')).push(control);
52. }
53. forbiddenNames(control: FormControl): {[s: string]: boolean}{
54. if(this.forbiddenUsernames.indexOf(control.value) !== -1){
55. return {'nameIsForbidden': true}
56. }
57. return null;
58. }
59. forbiddenEmails(control: FormControl): Promise<any>|Observable<any>{
60. const promise = new Promise<any>((resolve, reject)=> {
61. setTimeout(()=>{
62. if(control.value==='test@test.com'){
63. resolve({'emailIsForbidden': true});
64. }else{
65. resolve(null);
66. }
67. },1500);
68. });
69. return promise;
70. }
72. }

===========================END OF SECTION=============================