167)Module Introduction

In normal application we submit form to server but here we have single page application we cannot submit form to server, instead we need to handle this form through angular.

168)Why do we need Angular’s help

Consider this form, this is html code of form that your might use-

**<form>**

**<label>Name</lable>**

**<input type =”text” name=”name”>**

**<label>Mail</lablel>**

**<input type=”text” name=”email”>**

**<button type=”submit”>Save</button>**

**</form>**

This is not angular specific. Now angular’s job is to allow you to retrieve values the user entered here and also some other things like is the form valid , does user enter valid information? All this will happen in typescript code, so in angular side of your application. So you somehow need to be able to parse the value that user entered and you somehow need some javascript object representation of your form in your typescript code to work with. So this object code looks something like this. It is more complex in reality . but these are key feature that we need to have – value of form- here we will have key value pairs, where key refers to name of input and value is value ebtered by user. Also we can have some metadata like valid, which represent that form is valid or not.

**{**

**value: {**

**name: ‘Max’,**

**email: ‘test@test.com’**

**}**

**valid: true;**

**}**

So angular gives us javascript representation of form making simple for us to retrieve values, see state of form and work with it.

169)Template Driven vs Reactive Approach

Angular offers 2 approaches when it comes to handling the forms. It offers-

**1)Template Driven approach** – it is called so because here, we setup our form in HTML code and angular will automatically infer the structure of your form, will infer which controls your form has, which inputs and makes it easy for you to get started.

Angular also has more complex approach -

**2)Reactive approach-**  here you actually define structure of form in typescript code. You also setup HTML form then you manually connect both. It gives greater control. You can fine tune every little piece of your form.

Template driven approach is fine for most useCase.

171)TD:Creating the form and Registering the Controls

Lets understand how angular creates js object representing our form in template driven approach. Great thing is you don’t have to do almost anything.

Make sure you have FormsModule in import array of your app.module. this built in module includes a lot of Forms related functionalities and it’s actually needed to get this template driven approach to work, to get this form creation by angular. Now with this import, angular will automatically create form for you, so javascript representation of form when it detects a form element in HTML code.

So you can think of this form element serving as a selector for some angular directive which then creates js representation of form for you. You cannot see that form represeantion as of now and it would be very empty, because one thing cannot happen automatically- angular will not automatically detect your inputs in this forms. You could argue that it should be able to scan html page and detect inputs, selectors etc. but you still might not want to add all these elements as controls to your form, with control I am refering to what is in javascript object. Soo you dnt want to have every input in your html as control in your form. Because of following reasons-

1)may be you have a dropdown which well you only change something you view on UI but input should not be part of what gets submitted.

2)may be you use some third party package which add some custom form controls which are dnt use input as a selector, then angular will have no chance of detecting that this is form control.

So you need to register control manually. This is simple, you pick the input which you want to add as a control, then we add **ngModel**  as attribute to that element. Code-

div class="form-group">

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

<input type="text"

id="username"

class="form-control"

ngModel>

ngModel is directive made avalaible by FormsModule. We also use it in 2-way data bidning.like this-

<input type=”text” [(ngModel)] = “name of variable”>

You can also use it for 2 way data binding but actually it is part of bigger module with more features.

Now to recognized it as control in our js object, we need to give angular one more piece of information. That is name of control. Right now it says ok, this input needs to be part of js object representation of this form. So whatever user enters in this text field should be value of this control. But what’s the name of this control? We need to give that information to angular. We do this by adding html name attribute.

<input type="text"

id="username"

class="form-control"

ngModel

name="username">

with this , this control will be registered in js representation of this form.

We also dothis for e mail field and dropdown. We give them ngModel directive and name attribute.

172)TD:Submitting and Using the form

Here we will make submitted so that we can see what user entered. our form has submit button, when we click on it submit event is triggered. Angular provideas us a directive **ngSubmit**  to listen to this event. So we use this directive on <form> leemnt to execute a function. in onSubmit we print something on console. Code-

<form (ngSubmit)="onSubmit()">

onSubmit() {

console.log('angular 2');

}

Now we want to see form object that angular has created for us.so we go to our template, as this is template driven approach so rule of thumb- everything we do, we do it in template. Everything you want to change about this form, you want to add new functionality , you do it in template. Now we want to get access to form created by angular. So we place local refrence on form elment and pass it as a argument to onSubmit function.

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

onSubmit(form: HTMLFormElement) {

console.log(form);

}

As a output we get a HTML elemnt. This is not jvascript object created by angular.

To pass form object created by angular as argument to onSubmit function. we set local refrence equal to something. We can set it equal to something exposed by this <form> here, keep in mind <form> here is kind of selector for directive built into angular which will create this js object automatically, then it will expose some data we can fetch here on this form element. We set it equal to ngForm. What it does is – it tells angular, hey please give me access to form you created automatically. So this is how you get aceess to this js object created by angular automatically.

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

onSubmit(form: NgForm) {

console.log(form);

}

Now we see a object on console. This as many properties. We also have value property. This again is a object, we have name of our controls(we set up names of controls in template) as key and value user filled as value. so this is how we get access to form object and value entered by user.

173)TD: Understanding Form state

In last lecture we got access to form object. There we saw that we have value property on this object by which we can access value entered by user. Our form object has lot of other properties also. This is very cool thing about this object. It allows us to really understand the state of our form. We can see which controls we have registered in our form object by checking controls property-

controls:Object

* 1. email:FormControl {validator: null, asyncValidator: null, pristine: false, touched: true, \_onCollectionChange: *ƒ*, …}
  2. secret:FormControl {validator: null, asyncValidator: null, pristine: false, touched: true, \_onCollectionChange: *ƒ*, …}
  3. username:FormControl {validator: null, asyncValidator: null, pristine: false, touched: true, \_onCollectionChange: *ƒ*, …}
  4. \_\_proto\_\_:Object

Here each control is of type FormControl. Each control has properties, most of these are same that we have on overall on form. Lets go back to overall form-

1. *NgForm {submitted: true, \_directives: Array(3), ngSubmit: EventEmitter, form: FormGroup}*
   1. control:FormGroup
   2. controls:Object
   3. dirty:(...)
   4. disabled:(...)
   5. enabled:(...)
   6. errors:(...)
   7. form:FormGroup {validator: null, asyncValidator: null, pristine: false, touched: true, \_onCollectionChange: *ƒ*, …}
   8. formDirective:(...)
   9. invalid:(...)
   10. ngSubmit:EventEmitter {\_isScalar: false, observers: Array(1), closed: false, isStopped: false, hasError: false, …}
   11. path:(...)
   12. pending:(...)
   13. pristine:(...)
   14. status:(...)
   15. statusChanges:(...)
   16. submitted:true
   17. touched:(...)
   18. untouched:(...)
   19. valid:(...)
   20. value:Object
       1. email:"cec"
       2. secret:"teacher"
       3. username:"csdc"
       4. \_\_proto\_\_:Object
   21. valueChanges:(...)
   22. \_directives:(3) [NgModel, NgModel, NgModel]
   23. \_\_proto\_\_:ControlContainer

Most of these are self explanatory. For ex

1)Dirty property is true because we changed something about this form. If we reload page then dirty is false, because we have’nt types into any input. if just click on fileds and dnt type on it then this property wnt be changed.

2)Invalid is false because we have’nt added any validators. So our form is valid . later we will add validations to our form.

3)We also have touched property to see if we clicked on any fields.

Later we will see how these properties can be useful in changing user experience. For ex- disabling the submit button if form is not valid.

174)TD: getting access to form with @ViewChild

Right now we are getting access to form by passing it method which is called when submit event happens. It is fine approach for most useCases but there is other approach that you can use.

In component section we saw @ViewChild , which allowed us to access local refrence in our typescript code. In case of forms we are also passing local refrence to our function. but here it does bot point to ElementRef but it points to NgForm object, but still it is local refrence in our template. So we can also use @ViewChild to get access to Form object.

So we comment out our old onSubmit method, it will be there for your refrence. We use alternative approach here-

Code-

mport { Component,ViewChild } from '@angular/core';

import { NgForm } from '@angular/forms';

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

@ViewChild('f') signupForm: NgForm;

suggestUserName() {

const suggestedName = 'Superuser';

}

onSubmit() {

console.log(this.signupForm);

}

}

here we get access to FormObject without passing it to onSubmit method. This is special useCase if you need to access the form not just at the point of time when you submitted the form but also earlier. We will see useCase for this in later lecture. So this in another way of getting access to form object in typescript code.

175)Td:Adding input validations to check user input

In template driven approach we can only add validations in template.

1)required - it is built in html attribute, but here angular will detect it so it acts as selector for built in angular directive. now our form will be invalid(valid property of form will be false) if this validation is not fulfilled.

2)email -this is built in angular directive which makes sure that we only enter email adresses.

<div class="form-group">

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

<input type="email"

id="email"

class="form-control"

ngModel

name="email"

required

email>

So if we don’t pass these validations, valid property on form object will be set to false.

now angular tracks the validation on form level and also on individual control level.in form object we have controls property which is another object. here we have all our form controls. All these controls have valid property. Infact these controls have almost all properties that our form object has. Now if we dnt fill email in email field, valid property of email will be false, but that of username will be true. However valid property of form will be true if all controls have have true valid property.so we can track validation on form as well as control level.

there is one another place where we can track validations-

if you will inspect your form elements, you can see it has couple of classes added to it. like ng-pristine,ng-invalid,ng-dirty,ng-touched etc. these are attached or detached dynamically by angular based on form state. we use them to style elemnts differently.

For example if we track username text box we will see this-

**<input \_ngcontent-c0 class="form-control ng-untouched ng-pristine ng-invalid" id="username" name="username" ngmodel="" required="" type="text" ng-reflect-required="" ng-reflect-name="username" ng-reflect-model="">**

So all classes beginning with ng are added by angular. It gives us information that whther we touched a text filed, whther we filled any value in text field etc. with this information we can style these input conditionally. In next lectures we will take advantage of fact that angular tracks the form state.

176)Built-in Validators & Using HTML5 Validation

Which Validators do ship with Angular?

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).

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.

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.

177)Using the form state

1)here we disbaled the button using valid property of form. -

**<button**

**class="btn btn-primary"**

**type="submit"**

**[disabled]="!f.valid"**

**>**

**Submit**

**</button>**

We are able to use f.valid because we have access to local refrence that we placed on form in html code.

2)Now we will take advantage of css classes that are added by angular dynamically to our form controls. these classes are ng-pristine,ng-valid,ng-touched,ng-dirty

we use this-

input.ng-invalid.ng-touched{

border: 1px solid red;

}

this will make border red of input fields which have classes ng-invalid and ng touched. now these classes are dynamically added to them by angular.

178)TD Outputting Validation Error Messages

now lets say i want to display messgae below email input filed only if that validations for email field not true. how do we get access to email field? In other words we want to get access to control added by angular.we use ngModel for it. code-

<input

type="email"

id="email"

class="form-control"

ngModel

name="email"

required

email

#email="ngModel">

<span class="help-block" \*ngIf="!email.valid && email.touched">Please enter a valid email!</span>

179)TD\_Setting Default values with ngModel

Now we want to define some default values which should be displayed for example our drop down dnt have any value selected, it would be nice if it has some value selected buy default.

To do this we have to change way we register our inputs as control, right now we register them by just adding **ngModel** (with out property and event binding) as attribute.

To achieve this we have to add ngModel with property binding.we bind it with some property in our ts class or with hardcoded value. ex-

[ngmodel]=” ’sumit ’”

Or [ngModel] =”name”

So we bind our dropdown with property in class to give it some default value-

<select id="secret"

class="form-control"

[ngModel] = "defaultQuestion"

name="secret">

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

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

</select>

Here we are using ngModel with one way binding. So here we do 2 things with ngModel, - register input as form control and bind it with some property to show some default value.

180)TD Using ngModel with Two-Way-Binding

<textarea

name="questionanswer"

rows="3"

[(ngModel)]= "answer"

>

</textarea>

Here we did 2 things with ngModel- register our textarea as formcontrol and bind it with a variable in class . but here binding is 2 way.

181)TD\_Grouping Form Controls

Now we want to group some things in value object we get in form object. This is what we are getting now-

value:Object

* email:"skjasjkl@gmail.com"
* questionAnswer:" acxwacw"
* secret:"pet"
* username:"sumeet"

we want to group email with password and questionAnswer with secret.we are doing this just to have some structure.this is helpful in very large forms. By this we can also validate the status of induivals form groups. We can do this by using directive ngModelGroup.this will group them into group of inputs.we need to provide it a value(string) also, this will be the key name for this group.

Now our first group i.e username and email has a wrapping div around them. So we will use this directive on wrapping div.

Now if we output the form object then we get-

value:Object

* questionAnswer:" Gewrman"
* secret:"pet"
* userData:Object
* email:"sumitsood3127@gmail.com"
* username:"sumeet"

we have different setup in controls also. Now controls property also have a userData control object. This object has all the properties that a normal form control will have. Like valid etc. So we created new control. also in html code we can see that classes like ng-dirty etc attached to div element on which we used ngModelGroup directive.these clasees are also attched on induival email and username controls.so now we can check validity of overall form group.

We can get acess to js object created for this control group by adding local refrence to element that holds the ngmodelGroup directive.

<div id="user-data"

ngModelGroup="userData"

#userData="ngModelGroup">

<div class="form-group">

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

<input type="text"

id="username"

class="form-control"

ngModel

name="username"

required>

</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"

required

email

#email="ngModel">

<span class="help-block" \*ngIf="!email.valid && email.touched">Please Enter a Valid Email</span>

</div>

</div>

<span class="help-block" \*ngIf="!userData.valid && userData.touched">UserData is Invalid</span>

Now using validity of this form group we can display a message if this form group(e mail and username) is not valid.

<p \*ngIf="userData.invalid && userData.touched">UserData is not Valid </p>

Note:- by using ngModelGroup directive , we are basically combining 2 or more form controls into one control.

182)TD\_Handling RadioButtons

Here we set up radio button in our form and then we use it as control. We also added required validation to it.

<div class="radio" \*ngFor="let gender of genders">

<label>

<input type="radio"

name= "gender"

ngModel

[value]= "gender"

required>

{{gender}}

</label>

</div>

Name of all radio is same because we want one of them to be checked. This is a HTML thing.

We have created a array named gender in our class.

183)Setting and Patching Form Values

Here we want to do set value in username text field when we click on suggetedUsername button. We can do this by 2 way binding but we can do this by 2 other approaches also. we got access to form by ViewChild. so we will use that. we got 2 different methods that we can use.

* Approach 1- setValue. allows us to set the value of entire form. As a argument we need to pass a javascript object exatly reperesenting our form.
* @ViewChild('f') signupForm: NgForm;
* suggestUserName() {
* const suggestedName = 'Superuser';
* this.signupForm.setValue({
* userData: {
* username: suggestedName,
* email: 'sumitsood3127@gmail.com'
* },
* secret: 'pet',
* questionAnswer: 'rocky',
* gender: 'Male'
* });
* }

this approach has one down side. even if we have some value prefilled lets say we have filled e mail, then they all will be reset.(reset to value we provide here). here advantage is we can set value of controls by one method. here we also override the value of each control.

**Approach 2-** here we can pass specific controls that we can over ride.it means we can override only username without chnaging other values.

this.signupForm.form.patchValue({

userData: {

username: suggestedName

}

});

note- patchValue is only avalible on form property.also setValue will be also avalible there.

184)Using Form Data

here we simply show our data of form in html page when we hit submit button. on submit we execute a function which give form value to our class variables. then we show these variable depending in form. we use ngIf to make sure that these variable are shown only if form is submitted.

185)Td\_Resetting the form

to reset the form use function reset on our form object. it will not only clear all values. but it will also reset various form properties like pristine, touched i.e all classes are set back to what they were initially. It’s like we just opened the page.

we can also pass same object to this function that we passed to setvalue function , which will then reset the form to specific values.