186)Introduction to Reactive Approach

In this approach form is created programmatically in typescript and synchronized with DOM. This approach allows us to configure our form in greater detail.

187)Setting up reactive Forms

In reactive approach form is created in code, programmatically. So in ts file I create a variable that will hold my form, type of this variable is **FormGroup**. We create sigupForm variable which is type FormGroup. In template driven approach we stored our form in variable of type NgForm. NgForm is automatically created wrapper in the end. But it was wrapping FormGroup in the end, because in angular, form in the end is the group of control. So overall form is of type FormGroup.

signupForm: FormGroup;

Now this is important in order to reactive approach to work , we need to import a module in app.module. for teamplate driven approach we need to import **FormsModule.** But for reative approach we need import **ReactiveFormsModule.** We can remove the import of FormsModule.

188)Creating Form in code

Now we need to initialize our form. We can do it when we declare our signupForm variable. But we will do it in ngOnInit hook. Make sure that you initialize your form before rendring the template.ngOnInit is called before template is rendered so we use it.

Code-

ngOnInit() {

this.signupForm = new FormGroup({

'username' : new FormControl(null),

'email' : new FormControl(null),

'gender' : new FormControl('male')

});

}

We initialize our form using FormGroup constructor, this constructor takes a js object as argument. If we pass empty object then our form will not have any controls. Controls are key value pair in object that is passes to this constructor. We use **‘ ‘** for proeprties because we want name of properties to be kept during minification when code is mangled, because I will refrence it in HTML form.

Then we create controls using FormControl constructor.to this FormControl constructor. we can pass couple of arguments. First argument is initial state or initial value of formControl.second argument will be single validator or array of validators. Third argument is potential asynchronous validators. We will comeback to validators later.

We want textfield to have some value when it is loaded for first time, then we will do this-

'username' : new FormControl(‘Default name’)

189)Reactive: Syncing HTML and Form

In last lecture we created our form in typescript class. But angular dnt know that this form will be attached to form that we had in html. So we need to synchronize them. By default angular will see <form> directive and create a js form object for us. We need to tell angular dnt create this object we have already created this object, we just need to sync them.

So we have to add some directives to override this behaviours, to give angular different instructions. Now for these directives to work, make sure that you have imported ReactiveForms in our app.module, otherwise you will get errors.

First directive that we add is **formGroup**, this tell angular take my formGroup(FormObject). don’t infer one. Don’t create a form for me. Use my FormGroup. as a value we provide the form that we have created in typescript file. Code-

<form [formGroup]="signupForm">

Another code

</form>

Now both forms are in sync. Now we need to tell which controls in ts form object needs to be linked with inputs in html. For this we use **formControlName** directive on inputs in html.

<input

type="text"

id="username"

[formControlName] ="'username'"

class="form-control">

</div>

Form object in ts-

this.signupForm = new FormGroup({

'username' : new FormControl(null),

'email' : new FormControl(null),

'gender' : new FormControl('male')

});

With this if I run my app and inspect the elements then I can see that classes like ng-valid,ng-pristine are attached with my inputs. So our forms are connected now.

190)Reactive\_Submitting the Form

Here we use (ngSubmit)="name of function()". but here we dnt need to pass our form object as we used to do in template-driven approach. this is because we have created our own form object in ts code.

code-

In Html-

<form [formGroup]="signupForm" (ngSubmit)="onSubmit()">

</form>

In TS-

onSubmit(){

console.log(this.signupForm);

}

Here we get object with same properties as we got in template driven approach.

191)Reactive\_ Adding Validation

In data driven approach we cnt configure our form in HTML code like we do in template driven approach. we can configure it in ts code only. that is why FormControl constructor take more than one arguments.in second argument we pass validaorts. we have some built in validators which we can specify using Validators object.(like Validators.required). note dnt specify parenthesis here, bcoz we dnt want to executre this method.it is static method made avaliable by validators. we only pass refrence to this method.angular will call this method whenever it detects that input of form control has changed. we can also pass mutiple validators using array.

code-

ngOnInit() {

this.signupForm = new FormGroup({

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

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

'gender' : new FormControl('male')

});

}

192)Reactive getting Access to Controls

now we have added validations to our controls now we want to check if a control is valid or not.and depending on this we will show some message to user(enter valid email id) for this we need access to form control. we can access form control in reactive approach using get method on form object(which is created by us). get method takes a argument. we can either specify name of control or path to control. we will see second apprach later.now name is path as we have only one level of nesting in our form.

<form [formGroup]="signupForm" (ngSubmit)="onSubmit()">

<div class="form-group">

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

<input

type="text"

id="username"

[formControlName] ="'username'"

class="form-control">

<span

\*ngIf="signupForm.get('username').valid && signupForm.get('username').touched"

class="help-block">Please EnterValid UserName</span>

</div>

Similarly we can access our overall form in html like this-

<span

\*ngIf="signupForm.get('email').valid && signupForm.get('email').touched"

class="help-block">Please EnterValid Email</span>

note- if we print our form object on console, then we have a property called controls. This object has name of all controls as its properties. Each proeprty again is a object with many property. Now we can access these proeprties on our controls like this-

signupForm.get('username').valid

193)Reactive: Grouping Controls

in last lecture we said that get can take either name of control or path of control. what is path of control?you can specify path there because you may have nested form. We want username and email into a form group. This is how we do it-

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')

});

}

FormGroup basically holds the group of Form Controls.

Now we have nested form , we ned to reflect same in html because we dnt have controls with name username and email. So our code will break. So how do we do this?

We create a new div element and place username and email in it, we do this to replicate the structure that we have in ts object.on this div we place **formGroupName.** So you could see the the schema there **formControlName** to tell angular which property in our ts object representing the form relates to which input and **formGroupName**  to tell it the same for a form group. As a value we pass it name of our formGroup that we set up in ts code.

<div formGroupName="userData">

<div class="form-group">

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

<input

type="text"

id="username"

[formControlName] ="'username'"

class="form-control">

<span

\*ngIf="!signupForm.get('username').valid && signupForm.get('username').touched"

class="help-block">Please EnterValid UserName</span>

</div>

<div class="form-group">

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

<input

type="text"

id="email"

[formControlName] ="'email'"

class="form-control">

<span

\*ngIf="!signupForm.get('email').valid && signupForm.get('email').touched"

class="help-block">Please EnterValid Email</span>

</div>

</div>

Now everything is fine. One problemis that now we cnt acess controls like this, now-

signupForm.get(‘username’)

it is bcoz now username control dnt ecixt on overall form.

Instead we need to use userData.username now.like this-

signupForm.get(‘userData.username’). overall code-

<form [formGroup]="signupForm" (ngSubmit)="onSubmit()">

<div formGroupName="userData">

<div class="form-group">

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

<input

type="text"

id="username"

[formControlName] ="'username'"

class="form-control">

<span

\*ngIf="!signupForm.get('userData.username').valid && signupForm.get('userData.username').touched"

class="help-block">Please EnterValid UserName</span>

</div>

<div class="form-group">

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

<input

type="text"

id="email"

[formControlName] ="'email'"

class="form-control">

<span

\*ngIf="!signupForm.get('userData.email').valid && signupForm.get('userData.email').touched"

class="help-block">Please EnterValid Email</span>

</div>

</div>

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

<label>

<input

type="radio"

[formControlName] ="'gender'"

[value]="gender">{{ gender }}

</label>

</div>

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

</form>

194)Reactive: Arrays of form control(Form Array)

I want to show you another feature of recative approach. let’s add new area to our form, may be below radio buttons. We add <div>, now in this div we want user to dynamically add form controls. Lets say hobbies. We add a button and when user clicks on button I dynamically adds a control to my form here and specifically I want to add this control to an array of controls because I might have multiple hobbies.

Now to add this new control , I go to form object in my class and add new property hobbies, type is **FormArray.** Now form array basically holds an array of controls. So you pass an array to initialize it , in this array you initiliaze some FormControls, or we can leave it empty, to not have any hobbies at beginning of code. In onAddHobby function we add hobbies.

So I access my Formarray and then hobbies array(it is control in my form in end so I use get method).now I need to tell typescript that it of type FormArray, do not get an error. We we explicitly cast it. Now we can use push method on this FormArray, if we dnt cast it then we will get an error, when we try to push it. Hobby is something that user should create , we initialize it with null .

Code-

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([])

});

}

onAddHobby() {

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

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

}

Now we have added new control but we wnt be able to see it, we need to sync it with our HTML code. So I add a directive **FormArrayName ,**  to enclosing dive tag. This tells angular that somewehere in this array our FormArray lives. So in this div I create a input which allows user to create a hobby. Now I need to loop through all controls that are in this array, so we add ngFor loop.

Then we get index from loop. We need this to assign this input to one of these dynamic created controls. Then we use fromControlName on input to link this input with dynamically created input. Now this dynamically created input will not have name choosen by us. It will be in array. So we use array index.

App.component.html-

<form [formGroup]="signupForm" (ngSubmit)="onSubmit()">

<div formGroupName="userData">

<div class="form-group">

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

<input

type="text"

id="username"

[formControlName] ="'username'"

class="form-control">

<span

\*ngIf="!signupForm.get('userData.username').valid && signupForm.get('userData.username').touched"

class="help-block">Please EnterValid UserName</span>

</div>

<div class="form-group">

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

<input

type="text"

id="email"

[formControlName] ="'email'"

class="form-control">

<span

\*ngIf="!signupForm.get('userData.email').valid && signupForm.get('userData.email').touched"

class="help-block">Please EnterValid Email</span>

</div>

</div>

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

<label>

<input

type="radio"

[formControlName] ="'gender'"

[value]="gender">{{ gender }}

</label>

</div>

<div formArrayName="hobbies">

<h4>Your hobbies</h4>

<button class ="btn btn-default" type="button" (click)="onAddHobby()">Add Hobby</button>

<div class="form-group"

\*ngFor ="let hobbyControl of signupForm.get('hobbies').controls; let i = index">

<input type="text" class="form-control" [formControlName]="i">

</div>

</div>

<span

\*ngIf="!signupForm.valid && signupForm.touched"

class="help-block">Form is not valid</span>

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

</form>>

Now value property of my form object is like this-

1. value:
   1. gender:"male"
   2. hobbies:Array(2)
      1. 0:"Coding"
      2. 1:"biking"
      3. length:2
      4. \_\_proto\_\_:Array(0)
   3. userData:
      1. email:"sumitsood3127@gmail.com"
      2. username:"sumeet sood"

now you may ask that what is role of formArrayName directive? Here is it’s use-

when we add input filed, we bind it with control, like this-

<div class="form-group"

\*ngFor ="let hobbyControl of signupForm.get('hobbies').controls; let i = index">

<input type="text" class="form-control" [formControlName]="i">

</div>

Now we don’t have any controls in form which have name -0,1,2,3,4. This is where **formArrayName** comes into play, it tells angular to bind these input fields with aray elemnts of our formArrayDirective.

For more example, see recipe-edit.component of next control.

195)Reactive: Creating Custom Validators

Till now we used built in validators. In most of cases built in validators are enough. But lets say we dnt want to allow some usernames. We can built custom validators for it.

Validator is a function which gets executed by angular automatically when it checks the validity of formControl and it checks the validity whenever you change the control.

So we define a function-**forbiddenNames**. In order to work correctly it should receive a argument which is the control it should check. a validator also needs to return something for angular to be able to handle the return value correctly . this something is javscript object. Following notation might look strange, it should have any key which can be interpreted as string and (this is just a typescript syntax of saying that ,we want to have key value pair where key again can be interpreted as string, which is true for key in object as general ). More importantly value should be a Boolean.

So this function should return something like this-

**{nameisForbidden: true}**

keyName is upto us. Then we add logic that value enterd by us in control is in array of string.

This is important- if validation is successful then we have to return nothing or null. So you omit the return statement or you return null.

So this is our validator-

forbiddenNames(control: FormControl):{[s: string] : boolean} {

if(this.forbiddenUsernames.indexOf(control.value)) {

return {'nameIsForbidden': true};

}

return null;

}

Now we want to add to username control, that is how we do it-

this.signupForm = new FormGroup({

'userData' : new FormGroup({

'username' : new FormControl(null, [Validators.required,this.forbiddenNames]),

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

}),

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

'hobbies': new FormArray([])

});

}

We add this function to validators array , that we a pass as second argument to FormControl constructor.

If we run this code we will get an error.

**Cannot read property 'forbiddenUserNames' of undefined**

This error has something to do with the way javascript handles **this**.

In forbiddenNames function, I am accessing this-

forbiddenNames(control: FormControl): {[s: string]: boolean} {

if (this.forbiddenUserNames.indexOf(control.value) !== -1) {

return {'nameIsForbidden': true};

}

return null;

}

This might look fine because we are in class and I access **this.forbiddeUserNames** .

But think about who is calling function, this.forbiddenNames?we are not calling it from class, angular will call it when it checks the validity, at that point of time **this** will not refer to our class here.

So to fix this we use bind function-

'username' : new FormControl(null, [Validators.required, this.forbiddenNames.bind(this)])

We use this java script trick to make sure that this refers to what we want it to refer to.

Now in next lecture we will see how we can use this strange error code(object that we pass).

Question asked to Max-

Here I asked a question- how to add validation for password and confirm password, see response for that-

<https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/questions/3486390>

196)Reactive: using error codes

In last lecture we added our own validator. Now we are also passing a object in case our input is not valid.what is use of that?

If we see our formObject it has property called error, it is null.

But if we go to controls property,

If we see our form object that w eprint on console, we have a error property, but it is null. However if we go to controls property -> userData ->controls -> username ->errors. Here we see our object. So angular adds the object emitted by us to error property of induvial control.

Now how can we use it. Lets say we want to display message that –“username is not valid”. This is how we use it-

<span

\*ngIf="!signupForm.get('userData.username').valid && signupForm.get('userData.username').touched"

class="help-block">

<span

\*ngIf="signupForm.get('userData.username').errors['nameIsForbidden']">This name is

invalid</span>

<span

\*ngIf="signupForm.get('userData.username').errors['required']">This Field is

Required</span>

</span>

In case we leave this field empty, value of error object is –

errors:{required: true}

So we display error message depending upon error. So this is how we use error messages.e can also use ngswitch set up here. Here key thing to understand is that how we can use these errors codes to show right error messages. And ofcourse you can come up with even more complex setups where you have typescript objects in your typescript code , where you map error codes to specific messages and dynamically output them here.

You can see the class of this input changing from valid to invalid and vice versa in chrome dev tools.

197)Reactive: Creating a Custom Async Validator

We might need to reach out to web server for some validations. This as asynchronous operations. For these cases we have asynchronous validators.

Here we have asynchronous validator- forbiddenEmails. Arguments are same as that of synchronous validator but return type is different. Here in the end we return promise.Code-

forbiddenEmails(control: FormControl): Promise<any> | Observable<any>{

const promise = new Promise<any>((resolve, reject) => {

setTimeout(() => {

if (control.value === 'test@test.com') {

resolve({'emailIsForbidden': true});

}

// tslint:disable-next-line:one-line

else {

resolve(null);

}

}, 2000);

});

return promise;

}

Now we need to add this validator to our code. We add them as third argument to FormControl constructor.

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

Here we dnt use bind to stabilize this value because we dnt use this keyword in our function.

Now if we open chrome dev tools, initially email has class ng-invalid. Now if I type something, then first class changes to ng-pending, then it changes to ng-valid or ng-invalid.

198)Reactive: Reacting to status and value change

In last lecture we added a async validator, there saw that status of input switched from invalid to pending to valid for example. There actually is state that you can track in general. I will do it in ngOnit life cycle hook,

On signupForm or on each control of this form(this.signupForm.get(‘email’)), we have 2 observables we can listen to- **statusChanges** and **valueChanges** . first lets listen to value changes.

ngOnInit() {

this.signupForm.valueChanges.subscribe((value) => {

console.log(value);

});

Now when we type in any text box , value object of form is emitted and we have subscribed to it and we print it out on console. instead of form we can also do it on induival control.

Lets see statusChange object-

this.signupForm.statusChanges.subscribe((value) => {

console.log(value);

});

Here again status property of form or induvial control is emitted, whenever we type anything in textbox. So we see INVALID,VALID or PENDING.

199)Reactive: Setting and Patching Values

Not only we can listen to update son our form we can also update our form . Just like Template driven approach we have setValue and patchValue for you.

To setValue we pass a java script object that resembles our form object that we have created. Code-

this.signupForm.statusChanges.subscribe((value) => {

console.log(value);

});

this.signupForm.setValue({

userData: {

username: 'sumeet',

email: 'sumitsood3127@gmail.com'

},

gender: 'male',

hobbies: []

});

We also have patchValue if we want to partially update the form.

this.signupForm.patchValue({

userData: {

username : 'Nihilent Technologies',

email: 'sumeetsood3127@gmail.com'

}

});

We also have reset fumction if we want to reset the form. Aslo kep in mind that you can pass object to reset function to reset to specific values.