**Introduction about form**

Form have value and state

Template-Driven and Reactive Approach: Before diving in the actual code it’s super important to understand that angular actually offers two approaches when it comes to handling forms:

1. Template-Driven Approach: You set up your form in the template 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 quickly
2. Reactive Approach: Complex Approach, you actually defined the structure of the form in typescript code, You also set up the HTML code and then you manually connect that, therefore it gives you greater control over it

**TD: Creating The Form and Registering the Control**

Let understand how angular create such a javascript object representing our form in the template driven approach. The great thing is almost you don’t do anything. Make sure in the app module you actually import the form module

Of course you can’t see that form representation as of now and it would be very empty because one thing doesn’t happen automatically, angular will not automatically detect your imputs in this form And simple reason for this is that which you can argue(bien ho) that it should be able to scan your html code and detect that you have input here and here and that you have a select dropdown list. You still might not want to add all these elements as controls to your form , and again not every input in your html code might be control you want to have in your javascript form. May be you have dropdown which value you only change something you view on the UI and the input should not actually be a part of what gets submitted

**So you still need to register controls manually, You need to tell to angular: Hey within that form element what should be an actual control of my form And this is what we’re going to do. Now tell angular How does our form look like**

<input

type="text"

id="username"

class="form-control"

ngModel>

You might know ngModel have two way bindind, but this one is enough to tell angular this input is actually a control of my form

<input

type="text"

id="username"

class="form-control"

ngModel

name="username">

**Must also have name attribute as a key**

**Submit and Using the form**

In the component.ts you create onSubmit() method and after that come back to html file, you might think onSubmit method put at button when we click, but it is not good place. If you have a button in a form element this button will submit the form will send the request normally But besides that i will also trigger a javascript event

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

We want to get access to the form created by angular. Now you’ll learn about local reference that you can place on HTML to get access them. So we could place hash tag on the form element and now we could access this form on the f reference in our template and we could pass f as an argument to the onSubmit() method and print that

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

onSubmit(form: ElementRef) {

console.log(form);

}

**Log form here, it’s not a javascript object created by angular, angular created javascript object automatically, but now we can access it**

To Do That come back html file: You set this local reference equal to something. Now we haven’t done that yet. Keep in mind the form element is kind of a selector for a directive built into angular which will create a javascript object automatically and then it will expose some data we can fetch here on this form element. **We can get acces to it by writing ngForm assign to local reference**

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

onSubmit(form: NgForm) {

console.log(form);

}

**It tell angular hey, please give me access to this form you created automatically. This is how you get access to the form to this javascript object created by angular automatically**

****Key-value: email, secret, username( key is name attribute in inputs of form

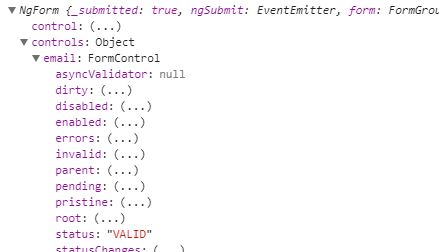
**Summary:**

Something we need to access javascript object in the form:

1. Add name attribute to inputs you want to control
2. Add ngModel to inputs you want to control
3. Add (ngSubmit)=”anyMethod” in beginning of form tag
4. Add localreference to the form
5. Put local reference as a parameter to anyMethod
6. Assign ngForm to local reference
7. Create anyMethod at abc.component.ts

**TD: Understanding Form State**

We also know javascript object from form have a lot of properties, one of them, it allows us to really understand the state of our form, you can view it at “controls” properties of Javascript object



**TD: Accessing the Form with @ViewChild**

Remember the components section where we learned about @ViewChild which allow us to access a local reference element controller or which hold a local reference in our typescript code

@ViewChild("f") signUpForm: NgForm;

onSubmit() {

console.log(this.signUpForm);

}

**Adding Validation To Check User Input**

To play around with form, we must use validation to validate valid or invalid value inputs

<input

type="email"

id="email"

class="form-control"

ngModel

name="email"

required

email>

If valid in HTML file angular automatically give us some css class:

**form-control ng-dirty ng-touched ng-valid**

if invalid in HTML file angular automatically give us some css class:

**form-control ng-dirty ng-touched ng-invalid**

to show state of the individual control here wherever it is dirty or not. So wherever we did changed the initial value wherever it touched or not. Now with that information we can style these inputs conditionally

**Built-in Validator and Using HTML5 Validation**

<https://angular.io/docs/ts/latest/api/forms/index/Validators-class.html>

Built-in Validators & Using HTML5 Validation

Section 15, Lecture 176

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.

**Using the form state**

Angular can track the state of each control of the form

Try with submit button:

Let disable the submit button if the form is not valid

<button

class="btn btn-primary"

type="submit"

[disabled]="!f.valid"

>Submit</button>

We use property binding for disabled and we use local reference f with property valid of f to check. As you know local reference can use at every where in the html => If form is not valid, button submit would be disabled

Try with css ng-invalid:

As you might know, if the form is invalid and controls is invalid, angular will automatically give css class as ng-invalid: so we custom this css class in component.css file

.ng-invalid {

border: 1px solid red;

}

The result is border red in invalid control and also in invalid form, but it’s not nice, because we only want to apply this css border red to only inputs So we continue to custom

input.ng-invalid {

border: 1px solid red;

}

**Perfect! If we do above code, it is only border to inputs but We remain have a problem, when form start it validate inputs and show error, it is also not great because it’s not friendly with users, only when they click submit if form is invalid then error show.**

input.ng-invalid.ng-touched {

border: 1px solid red;

}

Default when form load, we don’t see any errors even though the form isn’t valid, because you haven’t touched it so this state is untouched

**TD: Outputting Validation error messages**

<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>

To show “Please enter a valid email” when user input a invalid email with not be a email format, We create a span tag to show, make sure this span tag must have **ngIf** because It only showed when invalid, and **How we can access ngModel?** , We do same the way we access ngForm by Using **local reference,** you can see it in above code **#email=”ngModel”** and we use this local reference to check it valid or invalid in ngIf of error message and **don’t forget check it touched or untouched to give user a chance to input**

**TD: Set Default Values ngModel PropertyBinding**

Use one way binding ngModel to set default value

Select input such as question input or username input, we can set default value by one way binding as following code:

[ngModel]="defaultQuestion"

defaultQuestion = "teacher";

why we only one way binding here because only use to set default value we don’t need that values changed when we select any thing else

**TD: Using ngModel with Two-way Binding**

<div class="form-group">

<textarea

name="questionAnswer"

rows="3"

class="form-control"

[(ngModel)]="answer"

></textarea>

</div>

<p>Your reply: {{ answer }}</p>

export class AppComponent {

@ViewChild("f") signUpForm: NgForm;

defaultQuestion = "teacher";

answer: string;

Of course if we submit we will get a snapshot of the value at the point of time we hit submit, two way binding still possible, You can still use NgModel with two way binding

**Summary : Three ways of binding in form**

1. No binding to just tell angular data input is a control
2. One way binding to give that control a default value
3. Two way binding to instantly outputted or do whatever you want to do with that value

**TD: Grouping form control**

<div id="user-data" ngModelGroup="userData">

<div class="form-group">

**Handling Radio Button**

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

<label for="">

<input type="radio" name="gender"

ngModel

[value]="gender"

required

>

{{gender}}

</label>

</div>

**TD: Setting and Patching form value**

1. Setting
2. suggestUserName() {
3. const suggestedName = 'Superuser';
4. this.signUpForm.setValue(
5. {
6. userData : {
7. username: suggestedName,
8. email: ''
9. },
10. secret: 'pet',
11. questionAnswer: '',
12. gender: 'male'
13. }
14. );
15. }

**It wouble be override other values => not friendly, SetValue to set whole form**

2. Patching

this.signUpForm.form.patchValue({

userData: {

username: suggestedName

}

})

**Important patch value only available on the form wrapped by ngForm itself, patch value to override parts of the form**

**TD: Using form Data**

onSubmit() {

this.submitted = true;

this.user.username = this.signUpForm.value.userData.username;

this.user.email = this.signUpForm.value.userData.email;

this.user.secretQuestion = this.signUpForm.value.secret;

this.user.answer = this.signUpForm.value.questionAnswer;

this.user.gender = this.signUpForm.value.gender;

}

**TD: Resetting Form**

this.signUpForm.reset(); // reset to empty all fields

We can call reset like this and this wil reset the form and what this means it is will only empty. If you want, you can pass the same object as in setValue() to reset() which will then reset the form to specific value:

this.signUpForm.reset(

{

userData : {

username: 'abd',

email: ''

},

secret: 'pet',

questionAnswer: '',

gender: 'male'

}

);

**FORM: Reactive Setup**

Form will created automatically form typescript code,

Don’t need to FormModule in app module because it is necessary to Template approach, you need

ReactiveFormsModule

To app module, this is the module containing all the tools we need now to build our form on our own and then connected to our HTML code

**Create Form in Code**

You will initialized form in ngOnInit code, you should initialized before rendering a template

ngOnInit() {

this.signupForm = new FormGroup({

});

}

And create javascript object in the FormGroup

ngOnInit() {

this.signupForm = new FormGroup({

'username' : new FormControl(null),

'email' : new FormControl(null),

'gender' : new FormControl('male')

});

}

This is basic set up form with 3 form control

**Reactive: Syncing HTML code and Form in controller**

<form [formGroup]="signupForm">

Hey please take my form group, use my form group and need to set up property binding here because we need to pass our form as an argument to directive. So Now the form in HTML code are really synchronized with the form be created in typescript. But we still need to tell angular which control to be connected to which inputs and template code

We add formControlName directive to tell angular:

Hey what’s the name of this input in my form, in my typescript form

<input

type="text"

id="username"

formControlName="username"

class="form-control">

</div>

If you are wondering Why I’m not using property binding here: I’m passing a string here. If you want to use property binding you can do this

**Reactive: Submitting the Form**

We still do same the way with template Form, but difference are you don’t need to put local reference because hey we created the form on our own

onSubmit() {

console.log(this.signupForm);

}

Just like this, we have accessed to the form.

**Reactive: Adding Validation**

In template driven approach we simple add required or email to validate form-control and validate whole form. How we can validate with Reactive way?

But this way isn’t effective in Reactive approach because keep in mind and the key is you don’t create the form by template, you only synchronize it with the directives form control name and form group so not configuring here, let configure in typescript code

'username' : new FormControl(null, .. …),

Beside create default value, second argument allow you define some validation

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

Required is method but we don’t need call Validators.required() because we don’t execute this method we only make a reference to this method.

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

It also have array or validator

**Reactive: Getting access to control**

Use this form and get method of form to get names of form-control

<input

type="text"

id="username"

formControlName="username"

class="form-control">

<span

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

class="help-block">Please enter a valid username!</span>

**Reactive: Grouping Control**

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')

});

}

Create new form group userData to group username and email and you also must create a div with directive formGroupName and put form-controls: username and email inside it

<div formGroupName="userData">

<span

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

&& signupForm.get('userData.username').touched"

class="help-block">Please enter a valid username!</span>

So now you can get(‘userData.username’), before it can’t understand userData

**Reactive: Arrays of Forms Control (FormArray)**

Form Array hold array of control, and have one button to add form control to form Array

onAddHobby() {

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

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

}

And put hobbies to signUpForm

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([

])

});

Now we must synch above code to HTML code

Because Form group include hobbies, we will create a div tag with formArrayName

<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>

And a signupForm.get(‘hobbies’) will return an array of control, Beginning this array is empty, and when you click button OnAddHobby() it will create new A form-control and add to array and then show it to us. We use index of array to naming to form-controls by using property binding formControlName because index isn’t string value

**Reactive: Creating Custom Validators**

You want to custom or add new validator to serve your specific case. A Validator just a function which gets execute by angular automatically when it check the valid of the form control and it check validity when you change its control

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

// forbiddenUsernames

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

return {'nameIsForbidden': true};

}

return null;

}

Validator must return something which should be javascript object. It should have any key which can be interpreted as a string and is just type of syntax for saying hey: we want to have a key value pair where key again can be interpreted as a string which is true for key and object in general.

**More important the value of key value pair. There should be a Boolean. We check the condition in this method, check the value of control. If validation successful you have to pass nothing or NULL. You should not pass this object: return {'nameIsForbidden': false};**

'userData': new FormGroup({

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

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

}),

If we do as above code it will show error, because angular don’t know **forbiddenUsernames.**But think about who is calling these forbidden names. We are not calling it from inside this class, angular will call it when it check validity. At this point of time this will not refer to our class here. So to fix this I actually need to bind this, a good old javascript trick to make sure it refer to what you wanted refer to

'userData': new FormGroup({

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

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

}),

Let mind in head: **return {'nameIsForbidden': true}; is invalid**

**Reactive: Using Error Codes**

When a control invalid then inside of control have a field are errors: inside of errors are error code with pair key and value: nameIsForbidden: true, or require: true. So we can access these error codes in HTML file to check state to show error or warning messages

<span

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

&& signupForm.get('userData.username').touched"

class="help-block">

<span \*ngIf="signupForm.get('userData.username').errors['nameIsForbidden']"

class="help-message">This is name is invalid</span>

<span \*ngIf="signupForm.get('userData.username').errors['required']"

class="help-message">

This field required!

</span>

</span>

Above code they use errors with error codes are: required and nameIsForbidden

**Reactive: Creating a custom Async Validator**

Last lecture, we learn how to use validator code or custom validator to show error message. **But we might need to reach out web server to check this(Server Side)**

That however is the asynchronous operation because the response is not coming back instantly. Instead it just takes a couple of seconds. So we also kind of need asynchronous validators which are able to wait for a response before well returning true or false

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

return

}

Not returning a object with key and value of error code and value, it return Promise and Observable

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

const promise = new Promise(

(resolve, reject) => {

setTimeout(() => {

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

resolve({'emailForbidden': true});

} else {

resolve(null);

}

},1500);

}

);

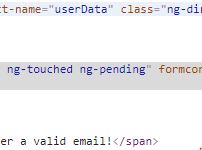
return promise;

}

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

this.forbiddenEmails),

asynchronous validator is third parameter of FormControl so don’t need to put it to array of second parameter.

 When we input new value to form-control have asynchronous validator, before html conclude it is valid or invalid field, it through ng-pending step to check, and then send the last conclusion

**Reactive: Reacting to status or value changes**

You can see last lecture from asynchronous validator, the status of input switch from invalid to pending to valid, or whenever you input value to username or email, it changes. Do you think we can subscribe it by implement observable?. Yes, because whenever have changes it will execute that code

this.signupForm.valueChanges.subscribe(

(value) => {

console.log(value);

}

);

Above code you are subscribing the values in the form, whenever values change, above code would be executed

this.signupForm.statusChanges.subscribe(

(status) => {

console.log(status);

}

);

Or can track to subscribe the status of inputs, watch switch between status, invalid => pending => valid

**Reactive: Setting and Patching values**

1. Setting values

this.signupForm.setValue({

'userData' : {

'username' : 'Max',

'email' : 'max@test.com'

},

'gender' : 'male',

'hobbies' : []

});

1. Patching values
2. this.signupForm.patchValue({
3. 'userData' : {
4. 'username' : 'Anna',
5. }
6. });

**Appy FORM to Recipe project**

pattern="^[1-9]+[0-9]\*$"

This is pattern for numbers with min = 1

Validators.pattern(/^[1-9]+[0-9]\*$/)])

When validate by reactive approach

<div class="row" formArrayName="ingredients">

<div class="col-xs-12">

<div

class="row form-group"

\*ngFor="let ingredient of recipeForm.get('ingredients').controls; let i = index"

[formGroupName]="i">

<div class="col-xs-8">

<input type="text" class="form-control" formControlName="name">

</div>

<div class="col-xs-2">

<input type="number" class="form-control" formControlName="amount">

</div>

<div class="col-xs-2">

<button class="btn btn-danger">X</button>

</div>

</div>

const newRecipe = new Recipe(

this.recipeForm.value['name'],

this.recipeForm.value['description'],

this.recipeForm.value['imagePath'],

this.recipeForm.value['ingredients']);

Don’t need to use :

const ingredients = [];

for(let ingredient of this.recipeForm.value['ingredients']) {

ingredients.push(ingredient)

}

Advantage of Reactive Approach, We also don’t need create

const newRecipe = new Recipe(

this.recipeForm.value['name'],

this.recipeForm.value['description'],

this.recipeForm.value['imagePath'],

this.recipeForm.value['ingredients']);

because the value of this form is map exactly to recipe:

if (!this.editMode) {

this.recipeService.addRecipe(this.recipeForm.value);

} else {

this.recipeService.updateRecipe(this.id, this.recipeForm.value);

}

**When we added new spaghetti and switch to shopping list and then comeback again recipes, the data lost. Where is our spaghetti? Let fix it**

The issue we encounter(cham tran) here is that we provide our recipes service in recipes component. So all component in this area share same instance(recipe service). But if we navigate away to the shopping list area the recipes component is destroy so the instance of this service. So what we need to do to ensure that our service added to app module