Greetings everyone!!!

In this video we will look at building forms in Angular 6.

In the end you will be familiar with different kinds of input fields such as;

Text boxes, check boxes, radio buttons and drop down lists.

You will be able to display validation errors next to our input fields and disable the submit button, if the form is not valid.

I thought it is best to have to have bootstrap, so that we will have a nicer looking UI for our template forms in our application.

To make certain parts of our form to work, we need install bootstrap for angular that is ng-bootstrap and bootstrap for styling.

In the terminal:

|  |
| --- |
| npm i --save @ng-bootstrap/ng-bootstrap  npm i --save bootstrap |

And add bootstrap style to our application CSS that is style.scss

@import "~bootstrap/dist/css/bootstrap.css";

Now let’s generate a component

|  |
| --- |
| ng g c job-application-form |

**Open job-application-form.html**

Now let’s go to job-application.html and get rid of all the mark up here.

Inside this form, let’s add few input fields, in fact I already got the code for building this entire job application form.

You can find the source code in the video description below.

<div class="container">

  <h3 id="branding" class="text-center">Be a part of our amazing team</h3>

  <div class="row">

    <div class="col-md-12 order-md-1">

      <form class="needs-validation">

        <div class="row">

<div class="col-md-6 mb-2">

            <label for="firstName">First name</label>

<input type="text" class="form-control" id="firstName" placeholder="" value="" >

</div>

<div class="col-md-6 mb-2">

<label for="lastName">Last name</label>

<input type="text" class="form-control" id="lastName" placeholder="" value="" >

</div>

</div>

<div class="row">

<div class="col-md-12 mb-1">

<label for="email">EMail<span>\*</span></label>

<input type="text" class="form-control" id="email" placeholder="" value="" >

</div>

</div>

<div class="row">

<div class="col-md-12 mb-1">

<label for="jobPosition">What position are you applying for<span>\*</span></label>

<select class="form-control" id="jobPosition">

<option></option>

<option>Awesome Position 1</option>

<option>Awesome Position 2</option>

<option>Awesome Position 3</option>

<option>Awesome Position 4</option>

<option>Awesome Position 5</option>

</select>

</div>

</div>

<div class="row">

<div class="col-md-12 mb-1">

<label for="availableStartDate">Available Start Date</label>

<div class="input-group mb-2">

<input type="text"

class="form-control"

id="inlineFormInputGroup"

placeholder=""

ngbDatepicker

#d="ngbDatepicker"

(dateSelect)="onDateSelect($event)"

(click)="d.toggle()">

<div class="input-group-append">

<button

class="btn btn-outline-secondary calendar"

(click)="d.toggle()"

type="button"></button>

</div>

</div>

</div>

</div>

<div class="row">

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

<label>What is your current employement status</label>

</div>

<div class="col-md-2 mb-1">

<div class="form-check">

<input class="form-check-input" type="checkbox" value="" id="invalidCheck2" required>

<label class="form-check-label" for="invalidCheck2">

Employed

</label>

</div>

</div>

<div class="col-md-2 mb-1">

<div class="form-check">

<input class="form-check-input" type="checkbox" value="" id="invalidCheck2" required>

<label class="form-check-label" for="invalidCheck2">

Un Employed

</label>

</div>

</div>

<div class="col-md-2 mb-1">

<div class="form-check">

<input class="form-check-input" type="checkbox" value="" id="invalidCheck2" required>

<label class="form-check-label" for="invalidCheck2">

Self employed

</label>

</div>

</div>

<div class="col-md-2 mb-1">

<div class="form-check">

<input class="form-check-input" type="checkbox" value="" id="invalidCheck2" required>

<label class="form-check-label" for="invalidCheck2">

Student

</label>

</div>

</div>

</div>

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

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

</form>

</div>

</div>

</div>

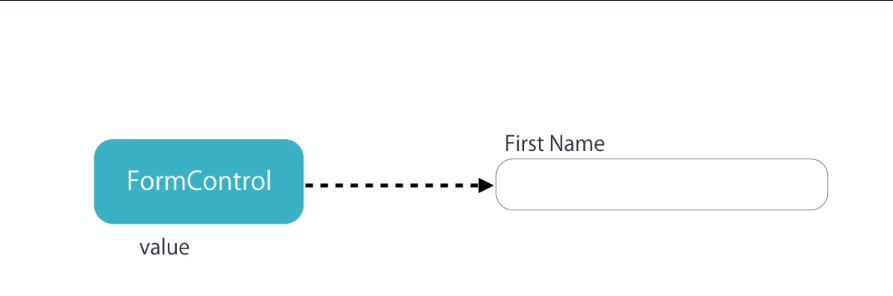
Now if you notice, we didn’t include “action” attribute and we used responsive layout for our form.

So when the screen size or viewport shrinks down, our form elements will stack accordingly, where as in a normal desktop, the form elements will be inline, at least the ones that we wanted to be.

Next is validating our form.

**Types of forms or Validation**

As you can see, we have built this simple job application form, now we need to add a validation to our form.



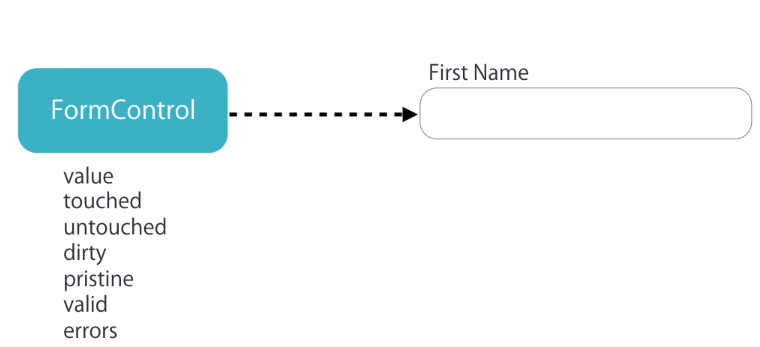
In Angular 6, we have a class called form control. For each form control, we need to create an instance of that control class.

Using that control class, we can check the current value stored in an input field.

Apart from checking the current value, we can see if our input field is touched, untouched, dirty (which means the value is changed), pristine (whether it’s value is not changed), is it valid? and any errors?

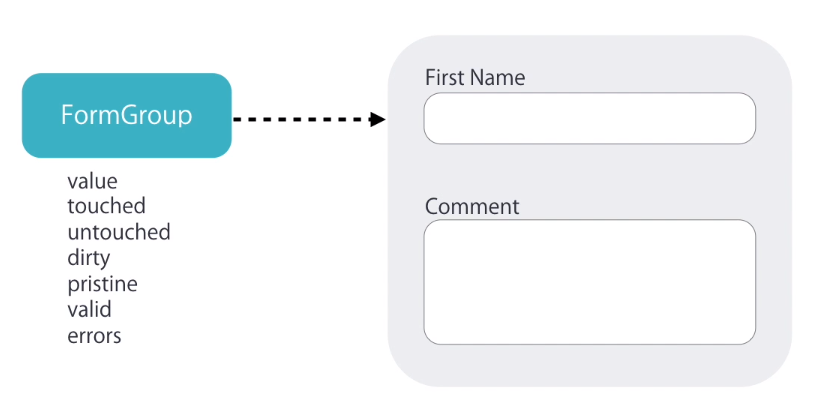
In order to add validation to our form, we need to create a FormGroup object for that form and FormControl object for that form.

So that we can keep track of each input field.



There is another class similar to FormControl class it is called FormGroup.

This FormGroup represents a group of controls. Good thing is, all the properties of FormControl class is also available for FormGroup class.



Now there are two ways to create control objects to our form. One way is by applying a directive to our form and Angular will take care of everything under the hood.

This type of form is known as Template driven forms.

Another method is by explicitly creating control object. In other words, we will be defining new instances of the control group and objects. We call these type of forms as “Reactive Forms”

“Reactive Forms” have more control over our validation logic and it is obviously has more advantageous when working with more complex forms.

In this lecture we are looking only at “Template driven Forms”, so let’s just stick with that.

ngModel

So, to begin with the validation, we going to apply a directive, which is most commonly used one, that is ngModel.

Our input field will look like this.

|  |
| --- |
| <input  ngModel  type="text"  class="form-control"  id="firstName"  placeholder=""  value=""> |

Now open the console in your chrome browser, instantly you will notice an error.

Error: If ngModel is used within a form tag, either the name attribute must be set or the form

control must be defined as 'standalone' in ngModelOptions.

The error basically tells us that ngModel requires a name attribute here and it is a requirement.

So adding a name attribute to our input.

|  |
| --- |
| <input  ngModel  name="firstName"  type="text"  class="form-control"  id="firstName"  placeholder=""  value=""> |

For the name attribute, I am gonna use the same value as it’s id and let us check for any errors in our browser.

It’s gone.

For a template driven approach, this is what all you need.

The next part is, referencing our input field to make use of it.

Let’s say I want to listen to a change event, so whenever someone types in, I should listen to that changes.

Now we need a reference to our ngModel, for that we need to create a template variable, so hashtag and the name of our input field, I will set it as “firstName” and as a value I am gonna use ngModel.

|  |
| --- |
| <input  ngModel  name="firstName"  #firstName="ngModel"  (change)="onChange(firstName)"  type="text"  class="form-control"  id="firstName"  placeholder=""  value=""> |

And then you can finally pass this template variable to our onChange method.

Next part is implementing this onChange method, so in our component class file.

|  |
| --- |
| onChange(x) {  console.log(x);  } |

Let’s quickly create a method with an argument called “x” and console this to our browser.

*NgModel {\_parent: {…}, name: "firstName", valueAccessor: {…}, \_rawValidators: Array[0]…}*

See, this is our ngModel and it is an object.

Now look at the control properties,

control: FormControl

It’s an instance of the FormControl class.

Things to note here is, we can use FormControl class to track the state changes and validity of our input fields when we apply the ngModel directive along with the name attribute of a form field.

So, back to validating our input field. In HTML5, we can use required tag to implement.

Now next to this input field I want to add a div that shows error.

|  |
| --- |
| <input  ngModel  name="firstName"  #firstName="ngModel"  (change)="onChange(firstName)"  type="text"  class="form-control"  id="firstName"  placeholder=""  value=""  required>  <div class="alert alert-danger mt-2">First Name is required</div> |

This is where we will display our validation error, which will show up as “First Name is required”.

And we wanna display this only if the input field is not valid, so we use ngIf.

Now how do we tell if our field is valid or not, simple, back in our console we have seen ngModel and FormControl, in the valid property this will either be true or false.

We can reference our template name directive and check if our input field is valid or not.

So, firstName.valid and add a not operator here to display only if valid is false.

<div class="alert alert-danger mt-2"

\*ngIf="!firstName.valid"

>First Name is required</div>

The functionality works as expected, but we got a problem, the error is shown initially, we don’t want that to happen.

Now let’s modify the code to show error only when user focus on input field and there is no input.

To do that, Angular FormControl has a property called touched which means it receives focus.

Modify the code which receives focus.

<div class="alert alert-danger mt-2"

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

>First Name is required</div>

Now, let’s set validation for this input field to show multiple validation errors. For now, let’s add minLength to our field.

<input

ngModel

name="firstName"

#firstName="ngModel"

(change)="onChange(firstName)"

(focus)="onFocus(firstName)"

type="text"

class="form-control"

id="firstName"

placeholder=""

value=""

**minlength="2"**

**maxlength="10"**

**required**>

<div

class="alert alert-danger mt-2"

**\*ngIf="firstName.invalid && (firstName.dirty || firstName.touched)">**

**<div \*ngIf="firstName.errors.required">First Name is required.</div>**

**<div \*ngIf="firstName.errors.minlength">First Name should be {{firstName.errors.minlength.requiredLength }} characters length.**

**</div>**

</div>

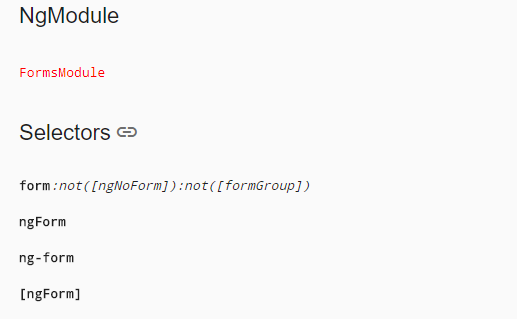


Now we understood about NgModel in our FormControl and how it behaves on working with validation.

We also have another class called FormGroup, which represents group of inputs or FormControls.

Angular by default applies ngForm directive to all the form elements in the HTML.

If you look at Angular.io documentation for ngForm, especially under “Selectors”, there is this selectors, which means by default ngForm is applied to all form elements in template.



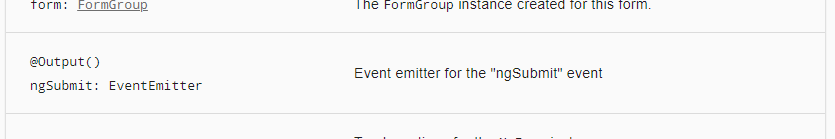
Now, let’s go ahead and declare a template variable, we can call it #jobForm and set it to ngForm.

<form

**#jobForm="ngForm"**

class="needs-validation">

in the documentation, you will notice the output property as ngSubmit. This output property can be used to raise custom events.



To use it, we can expose this ngSubmit in an event binding expression.

So, (ngSubmit)=”jobSubmit(jobForm)” and pass this template variable in to this method.

#jobForm="ngForm"

**(ngSubmit)="jobSubmit(**jobForm**)"**

Now, in the component, add this method submit(jobForm) and simply console.log(jobForm);

In the console, you will notice bunch of properties that are similar to FormGroup, and it also contains properties that we have seen earlier such as dirty, disabled, pristine, value, valid, etc.

These all are computed property key value pair based on the FormControls.

**Working with Radio Buttons**

We already got markup for our radio buttons and it is a static or hard coded.

We need to make our radio buttons dynamic as well as get them under FormControl.

Just like other input fields, we need to apply ngModel to our radio buttons.

For debugging purpose, let’s add

{{ jobForm.value | json }}

What if we want to render the list of radio buttons dynamically, so first thing we need to have a field in our class called employementStatus.

private employementStatus = [

{ id: 'rbEmployed', name: 'Employed' },

{ id: 'rbUnEmployed', name: 'Un Employed' },

{ id: 'rbSelfEmployed', name: 'Self Employed' },

{ id: 'rbStudent', name: 'Student' }

];

Back in the template, I am gonna get rid of all the radio buttons. Let’s start with \*ngFor

\*ngFor="let status of employementStatus"

And set the property binding for the id and value.

**Radio button checked by default**

To set the first radio button as default, we can add two way data binding otherwise known as banana in a box.

<div class="form-check"

\*ngFor="let status of employementStatus">

<input

ngModel

#empStatus="ngModel"

class="form-check-input"

type="radio"

name="employementStatus"

[(ngModel)]="employeeStatus"

[id]="status.id"

[value]="status.name"

(click)="onChange(empStatus)"

retuired

>

**Working with Dropdown Select Button**

Just like other input fields we need to add ngModel directive for our dropdown selectbox and also set the name attribute. So name will be “jobPosition”.

As with any dropdowns, the values will be fetched from an api, here we will use a field in our class called jobPositionList

private jobPositionList = [

{ id: 1, name: 'Awesome position 1' },

{ id: 2, name: 'Awesome position 2' },

{ id: 3, name: 'Awesome position 3' },

{ id: 4, name: 'Awesome position 4' },

{ id: 5, name: 'Awesome position 5' }

];

Now in our template, let’s delete all these options and iterate our jobPosition List

**Disabling the Button Submit**

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

<button

type="submit"

class="btn btn-primary"

**[disabled]="!jobForm.valid">**Submit</button>

</div>

Using our jobForm template variable we can add a property binding to our submit button called as [disabled], and provide an expression called “not jobForm.valid”.

Now our submit button will be enabled only if the form passes through validation.