# TD 4-part 2 : Angular {{}}, @Input, @Output field bindings, [(ngModel)] and ReactiveForm

## TD Objectives

Exercises corresponding to course CM3, on dynamic features of Angular: bindings

The goal is to realize how simple it is to use 1-way and even 2-ways (bi-directional) data binding, using the “[()]” (banana-box notation): [(ngModel)]=”field”.

Then we will focus on composing multiple levels of parent-child components, and make then communicate values using @Input (from parent to child field), and @Output (from child to parent callback).

Finally, we will see alternative field bindings with ReactiveForm, and syntax formControlName=”field”, which support validation status.

Pre-requisites =

* TD3-part 1 : angular project already setup (IntelliJ, ng serve, Chrome DevTools)
* TD3-part 2 : ng-bootstrap, for using bootstrap CSS library
* TD3-part 2 : menu navbar already configured in your main page
* TD4-part 1 : @Component and @Service (know to use “ng g c” and “ng g s”)
* TD4-part 1 : fortawesome for using fontawesome icons library
* TD4-part 1 : 3 pages components : lesson-edit-form, lesson-list, lesson-detail

## Step 1: reminder on creating component page + exposing Route + nav bar link

Create a component “test-page1”

ng g c test-page1

add (in src/app/app-routing.module.ts) a Route to expose it for path “test-page1”,

import {TestPage1Component} from "./test-page1/test-page1.component";  
  
const routes: Routes = [  
 { path:'test-page1', component: TestPage1Component },

and add (in src/app/app.component.html) a menu item with routerLink=”/test-page1”

<a class="dropdown-item" routerLink="/test-page1">Test Page1</a>

## Step 2: add {{ }} one-way binding to html text

Add in test-page1.component.html:

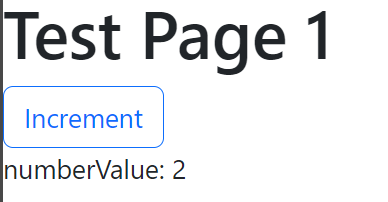
<H1>Test Page 1</H1>  
  
numberValue: {{numberValue}}

And corresponding field declaration in test-page1.component.ts:

numberValue = 1;

## Step 3: add button callback: (click)=”onClickIncrementValue()”

In src/app/test-page1/test-page1.component.html, add a button with a typescript callback, to increment the value when clicking



In test-page1.component.html:

<button type="button" class="btn btn-outline-primary" (click)="onClickIncrementValue()">Increment</button>

In test-page1.component.ts:

onClickIncrementValue() {  
 this.numberValue++;  
}

## Step 4: using one-way binding [value]

Instead of binding “{{numberValue}}” in plain html text, we want to bind to the “value” attribute of an <input> element. As it is a one-way binding yet, put this element in disabled editing mode.

<input disabled="true" [value]="numberValue">

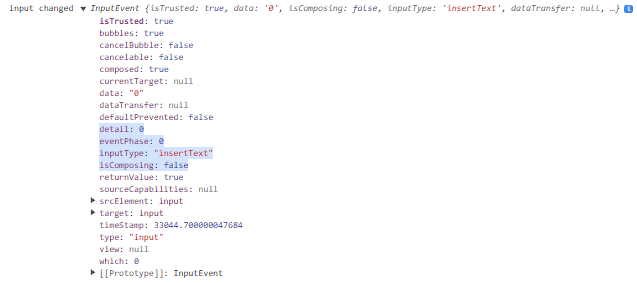
## Step 5: binding output event : (input)=”onInputEvent($event)”

Add another <input>, not disabled, with a handler for the “input” changed event

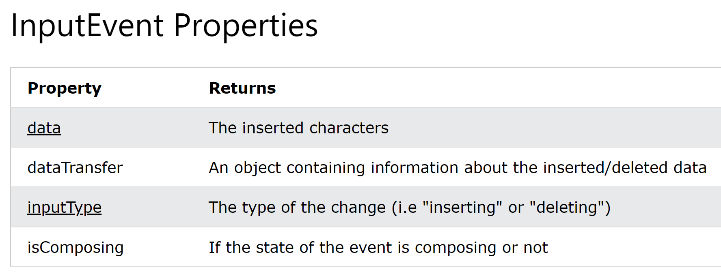
<input [value]="numberValue" (input)="onInputChanged(***$event***)">

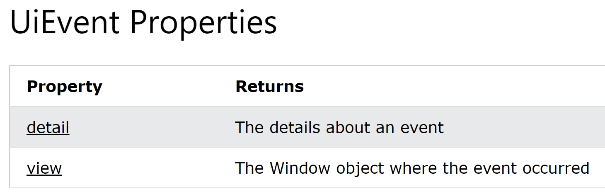
onInputChanged($event: Event) {  
 ***console***.log("input changed", $event)  
}

Check with Chrome debugger that you receive in “$event” parameter, and object with lot of fields, corresponding to the “InputEvent” type:



This InputEvent interface inherits from UIEvent, which inherits from Event. Check docs:

 <https://www.w3schools.com/jsref/obj_inputevent.asp>



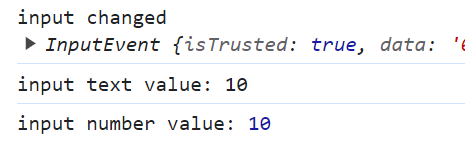
<https://www.w3schools.com/jsref/obj_uievent.asp>

## Step 6 : extract the text value from the input event, convert it to a number

retrieve the edited text value, and convert it to a number

onInputChanged($event: Event) {  
 ***console***.log("input changed", $event);  
 const input = <HTMLInputElement> $event.target;  
 const textValue: string = input.value;  
 ***console***.log('input text value:', textValue);  
 const numberValue: number = +textValue;  
 ***console***.log('input number value:', numberValue);  
}

Check in Chrome debugger the console output. Notice the first line show “10” in black (this is a string), and the second line “10” in blue (this is a number).

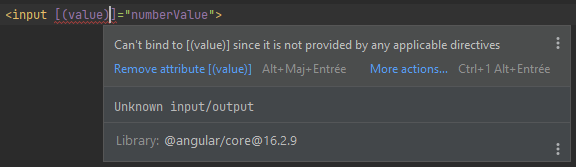


Using such boilerplate code, you could do 2-way data binding, but admittedly, this is not practical

onInputChanged($event: Event) {  
 this.numberValue = +(<HTMLInputElement> $event.target).value;  
}

Check that is works. When editing this input field, value change elsewhere

Also notice that there is a “value” attribute, but unfortunately, there is no “(valueChange)” event emitted. This is why you can’t use the following syntax to do 2-way data binding. Check the compile error code



## Step 7 : add FormModule to the main app.module.ts

We will simplify a lot the output binding, and 2-way data binding, by using [(ngModel)], as supported by angular FormModule.

In app.module.ts, add

import { FormsModule } from "@angular/forms";

@NgModule({  
… lines ommited

imports: [  
 BrowserModule,  
 AppRoutingModule,  
 FormsModule, // <= for [(ngModel)] supports

## Step 8: use [(ngModel)]=field

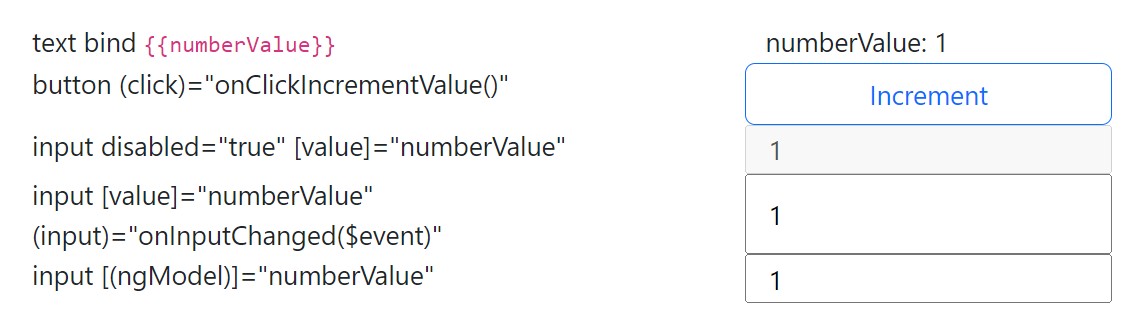
Now, you can add a third <input> field, with 2-way data binding between ngModel and your field “numberValue”:

<input type="number" [(ngModel)]="numberValue">

## summary code for Step 1,2, … 8

<div class="container">  
  
 <div class="row">  
 <label class="col-4">text bind <code ngNonBindable>{{numberValue}}</code></label>  
 <div class="col-2">numberValue: {{numberValue}}</div>  
 </div>  
 <div class="row">  
 <label class="col-4">button (click)="onClickIncrementValue()"</label>  
 <button class="col-2 btn btn-outline-primary" type="button" (click)="onClickIncrementValue()">Increment</button>  
 </div>  
 <div class="row">  
 <label class="col-4">input disabled="true" [value]="numberValue"</label>  
 <input class="col-2" disabled="true" [value]="numberValue">  
 </div>  
 <div class="row">  
 <label class="col-4">input [value]="numberValue" (input)="onInputChanged($event)"</label>  
 <input class="col-2" [value]="numberValue" (input)="onInputChanged(***$event***)">  
 </div>  
 <div class="row">  
 <label class="col-4">input [(ngModel)]="numberValue"</label>  
 <input class="col-2" type="number" [(ngModel)]="numberValue">  
 </div>  
  
</div>

Which renders as:



## Step 9: bind one-way field from parent to child component, using @Input()

Create 2 new components, called “number-display” and “number-edit”

ng g c number-display

ng g c number-steps

ng g c number-edit

We will NOT create Route for these components (they are not standalone pages), but we will use them directly as

<app-number-display></app-number-display>

<app-number-steps></app-number-steps>

<app-number-edit></app-number-edit >

Add a value field, annotated with @Input(), and display it in the corresponding html (notice, as it is a 1 line html, no need to externalize in a separate html file, so use template:’…’ instead of templateUrl: ‘file.html’)

@Component({  
 selector: 'app-number-display',  
 template: '<div> display value component: {{value}} </div>'  
})  
export class NumberDisplayComponent {  
  
 @Input()  
 value: number = 0;  
  
}

Use this component in your test-page1. Check that it works.

<app-number-display [value]="numberValue"></app-number-display>

## Step 10: bind event from child to parent, using @Output()

In component app-number-steps.component.html, declare 4 buttons, respectively -10, -1, +1, +10.

<div>  
 <button type="button" class="btn btn-small btn-outline-primary" (click)="onClickDecr10()">-10</button>  
 <button type="button" class="btn btn-small btn-outline-primary" (click)="onClickDecr1()">-1</button>  
 <button type="button" class="btn btn-small btn-outline-primary" (click)="onClickIncr1()">+1</button>  
 <button type="button" class="btn btn-small btn-outline-primary" (click)="onClickIncr10()">+10</button>  
</div>

Each button has a handler (click)=”onClickXXX()”, that emit an event to a corresponding @Output Subject.

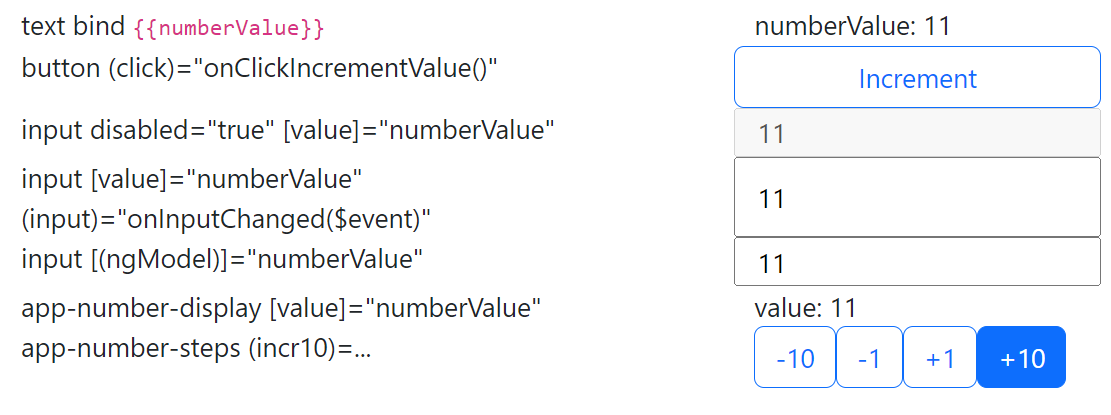
@Output()  
decr10 = new Subject<number>();  
@Output()  
decr1 = new Subject<number>();  
@Output()  
incr1 = new Subject<number>();  
@Output()  
incr10 = new Subject<number>();  
  
onClickDecr10() {  
 this.decr10.next(-10);  
}  
  
onClickDecr1() {  
 this.decr1.next(-1);  
}  
  
onClickIncr1() {  
 this.incr1.next(+1);  
}  
  
onClickIncr10() {  
 this.incr10.next(+10);  
}

In test-page1.html, add app-number-steps component, and bind 4 corresponding @Output to methods

<app-number-steps class="col-3"  
 (decr10)="onDecr10(***$event***)" (decr1)="onDecr1(***$event***)" (incr1)="onIncr1(***$event***)" (incr10)="onIncr10(***$event***)"  
></app-number-steps>

These method expectedly increment by -10,-1,+1,+10 the field numberValue

onDecr10($event: number) {  
 this.numberValue -= 10;  
}  
onDecr1($event: number) {  
 this.numberValue -= 1;  
}  
onIncr1($event: number) {  
 this.numberValue += 1;  
}  
onIncr10($event: number) {  
 this.numberValue += 10;  
}

Check that it works

## Step 11: combining @Input() field and @Output() fieldChanged: 2-way data binding

In number-edit.component.ts, add both @Input and @Output with name “value” (type number) and corresponding name “valueChange” (event type number)

import {Component, Input, Output} from '@angular/core';  
import {Subject} from "rxjs";  
  
@Component({  
 selector: 'app-number-edit',  
 template: '<input [(ngModel)]="value" (ngModelChange)="onInputChange()">'  
})  
export class NumberEditComponent {  
  
 @Input()  
 value: number = 0;  
  
 @Output()  
 valueChange = new Subject<number>();  
  
 onInputChange() {  
 this.valueChange.next(this.value);  
 }  
  
}

Add it in test-page1, check that it works

<app-number-edit [(value)]="numberValue"></app-number-edit>

Summary code in test-page1.html for Steps 9,10,11 :

<div class="row">  
 <label class="col-4">app-number-display [value]="numberValue"</label>  
 <app-number-display class="col-3" [value]="numberValue"></app-number-display>  
</div>  
<div class="row">  
 <label class="col-4">app-number-steps (incr10)=...</label>  
 <app-number-steps class="col-3"  
 (decr10)="onDecr10(***$event***)" (decr1)="onDecr1(***$event***)" (incr1)="onIncr1(***$event***)" (incr10)="onIncr10(***$event***)"  
 ></app-number-steps>  
</div>  
<div class="row">  
 <label class="col-4">app-number-edit [(value)]=numberValue</label>  
 <app-number-edit class="col-3" [(value)]="numberValue"></app-number-edit>  
</div>

## Step 12 : working on edit Form: re-open your lesson-edit-form page from TD3

Open file src/app/ lesson-edit-form/lesson-edit-form.component.html

You had defined it using bootstrap class=”container”, class=”row”, class=”col-md-\*”.

This form page was for editing object with fields “title”, “description”, “category”, “level”, etc.

export interface LessonPackage {

title: string;

description: string;

category: string;

level: number;

prerequisite: string[];

tags: string[];

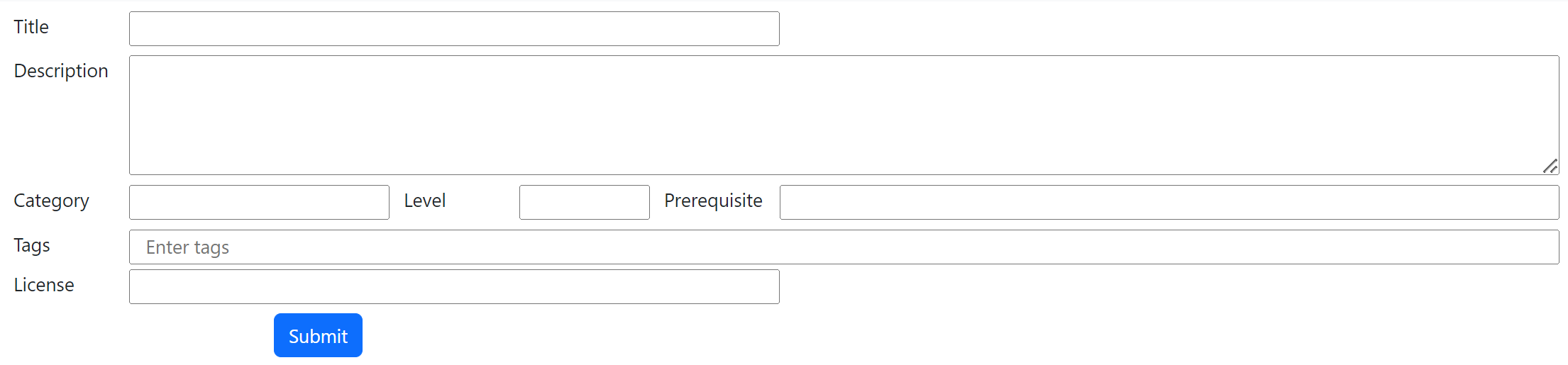
copyright: string;

}

Example html code :

<div class="container">  
  
 <div class="row mt-2">  
 <label class="col-md-1" for="title">Title</label>  
 <input id="title" type="text" class="col-md-5">  
 </div>  
 <div class="row mt-2">  
 <label class="col-md-1" for="description">Description</label>  
 <textarea id="description" class="col-11" rows="4"></textarea>  
 </div>  
 <div class="row mt-2">  
 <label class="col-md-1" for="category">Category</label>  
 <input id="category" type="text" class="col-md-2">  
 <label class="col-md-1" for="level" ngbTooltip="Enter value between 1(basics) to 10(advanced)" placement="top">Level</label>  
 <input id="level" type="number" class="col-md-1" min="1" max="10">  
 <label class="col-md-1" for="prerequisites">Prerequisite</label>  
 <input id="prerequisites" type="text" class="col-md-6">  
 </div>  
 <div class="row mt-2">  
 <label class="col-md-1" for="Tags" ngbTooltip="Enter tags, separated by commas" placement="top">Tags</label>  
 <input id="tags" type="text" class="col-md-11" placeholder="Enter tags">  
 </div>  
 <div class="row mt-1">  
 <label class="col-md-1" for="license">License</label>  
 <input id="license" type="text" class="col-md-5">  
 </div>  
 <div class="row mt-2">  
 <div class="col-md-2"></div>  
 <div class="col-md-2">  
 <button type="button" class="btn btn-primary">Submit</button>  
 </div>  
 </div>  
</div>

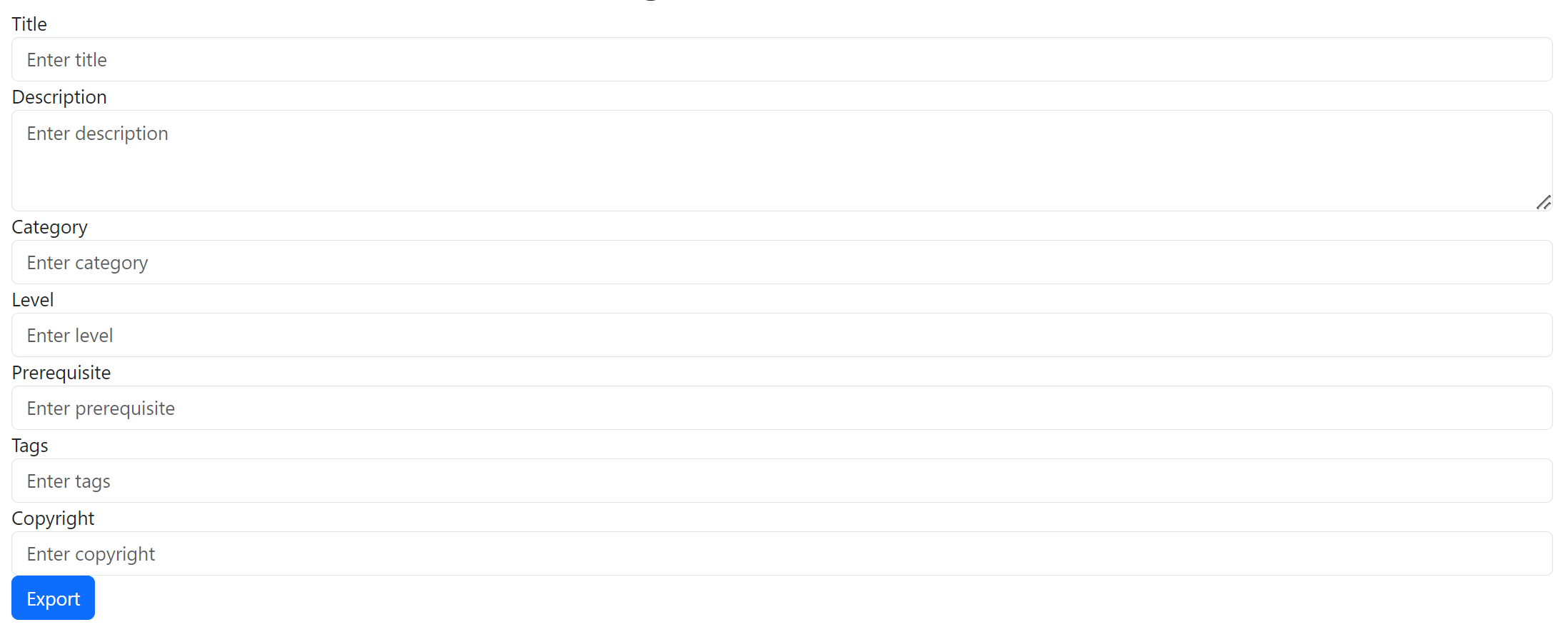
This renders as :



Equivalently you could define it in a flat way using <form> and class=“form-group” and class=”form-control”:

<form>  
 <div class="form-group">  
 <label for="titleField">Title</label>  
 <input type="text" class="form-control" id="titleField" placeholder="Enter title">  
 </div>  
 <div class="form-group">  
 <label for="descriptionField">Description</label>  
 <textarea class="form-control" id="descriptionField" rows="3" placeholder="Enter description"></textarea>  
 </div>  
 <div class="form-group">  
 <label for="categoryField">Category</label>  
 <input type="text" class="form-control" id="categoryField" placeholder="Enter category">  
 </div>  
 <div class="form-group">  
 <label for="levelField" ngbTooltip="Enter value between 1(basics) to 10(advanced)">Level</label>  
 <input type="text" class="form-control" id="levelField" placeholder="Enter level" min="1" max="10">  
 </div>  
 <div class="form-group">  
 <label for="prerequisiteField">Prerequisite</label>  
 <input type="text" class="form-control" id="prerequisiteField" placeholder="Enter prerequisite">  
 </div>  
 <div class="form-group">  
 <label for="tagsField" ngbTooltip="Enter tags, separated by commas">Tags</label>  
 <input type="text" class="form-control" id="tagsField" placeholder="Enter tags">  
 </div>  
 <div class="form-group">  
 <label for="copyrightField">Copyright</label>  
 <input type="text" class="form-control" id="copyrightField" placeholder="Enter copyright">  
 </div>  
 <button type="submit" class="btn btn-primary">Export</button>  
</form>

This renders as :



Notice in both form that each <label> has an attribute

<label for=”someFieldId”>

And each <input> or <textarea> has the corresponding id attribute

<input id=”someFieldId”>

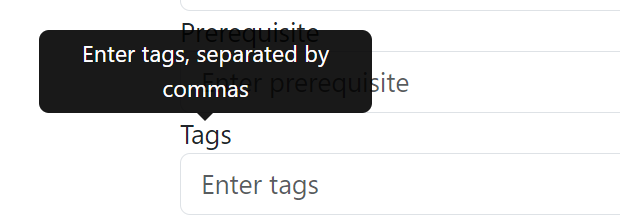
There could also be “aria-\*” attributes (aria means “Accessible Rich Internet Applications”), to improve accessibility particularly for users with disabilities.

For <input> fields, there can also be placeholder attribute, like

placeholder="Enter tags"

Finally, you could add tooltip text that pop-over on the label, for example:

ngbTooltip="Enter tags, separated by commas" placement="top"



## Step 13 : bind simple <input> values to a corresponding object field

Bind field “title: string”, using [(ngModel)]

<input id="title" type="text" class="col-md-5"   
 [(ngModel)]="title">

Field must be declared in corresponding lesson-edit-form.component.ts (otherwise Angular fails to compile):

title: string = '';

This would force you to declare all fields, then to re-assemble these fields in a json object for saving later (call the server)

title: string = '';  
description: string = '';  
category: string = '';  
level: number = 1;  
prerequisite: string[] = [];  
tags: string[] = [];  
copyright: string = '';  
  
onClickSubmit() {  
 const formValues: LessonPackage = {  
 title: this.title,  
 description: this.description,  
 category: this.category,  
 level: this.level,  
 prerequisite: this.prerequisite,  
 tags: this.tags,  
 copyright: this.copyright  
 };  
 ***console***.log('form values to save to server', formValues);  
}

Alternatively, you could directly declare your target object to be filled,

model: LessonPackage = { title: '', description: '', category: '', level: 1, prerequisite: [], tags: [], copyright: ''};

and bind directly sub-fields of this instance to corresponding <input>s.

<input id="title" type="text" class="col-md-5"  
 [(ngModel)]="model.title">

The object is ready to be used :

onClickSubmit() {  
 ***console***.log('form values to save to server', this.model);  
}

## Step 14: Better way of defining angular form fields, ReactiveForms with validation + dirty/pristine classes

Unfortunately, in previous step, our class did not handle any validation checks. For example, if a field is mandatory, it must be filled, otherwise the label should be displayed in “red”, the submit button greyed, and an error explanation message displayed. If there is a format to respect (a regular expression, a min-max constraint, etc), it must be validated, etc.

For every value that we have, we should add several corresponding boolean values (valid,dirty,pristine,etc…) , and bind them with a lot of boilerplate code.

Change your form to use angular ReactiveForm

In app.module.ts:

import {FormsModule, ReactiveFormsModule} from "@angular/forms";

imports: [  
 BrowserModule,  
 AppRoutingModule,  
 FormsModule,  
 ReactiveFormsModule, // <= for supports FormGroup/FormBuilder

in lesson-edit-form.ts

import {FormBuilder, FormGroup, Validators} from "@angular/forms";

lessonForm: FormGroup;  
  
constructor(formBuilder: FormBuilder) {  
 this.lessonForm = formBuilder.group({  
 title: ['', Validators.*required*],  
 description: ['', Validators.*required*],  
 category: [''],  
 level: [''],  
 prerequisite: [''],  
 tags: [''],  
 copyright: ['']  
 });  
}  
  
onSubmit() {  
 if (this.lessonForm.valid) {  
 const formData = this.lessonForm.value;  
 ***console***.log('Form data submitted:', formData);  
 } else {  
 ***console***.log('Form is invalid. Please check the required fields.');  
 }  
}

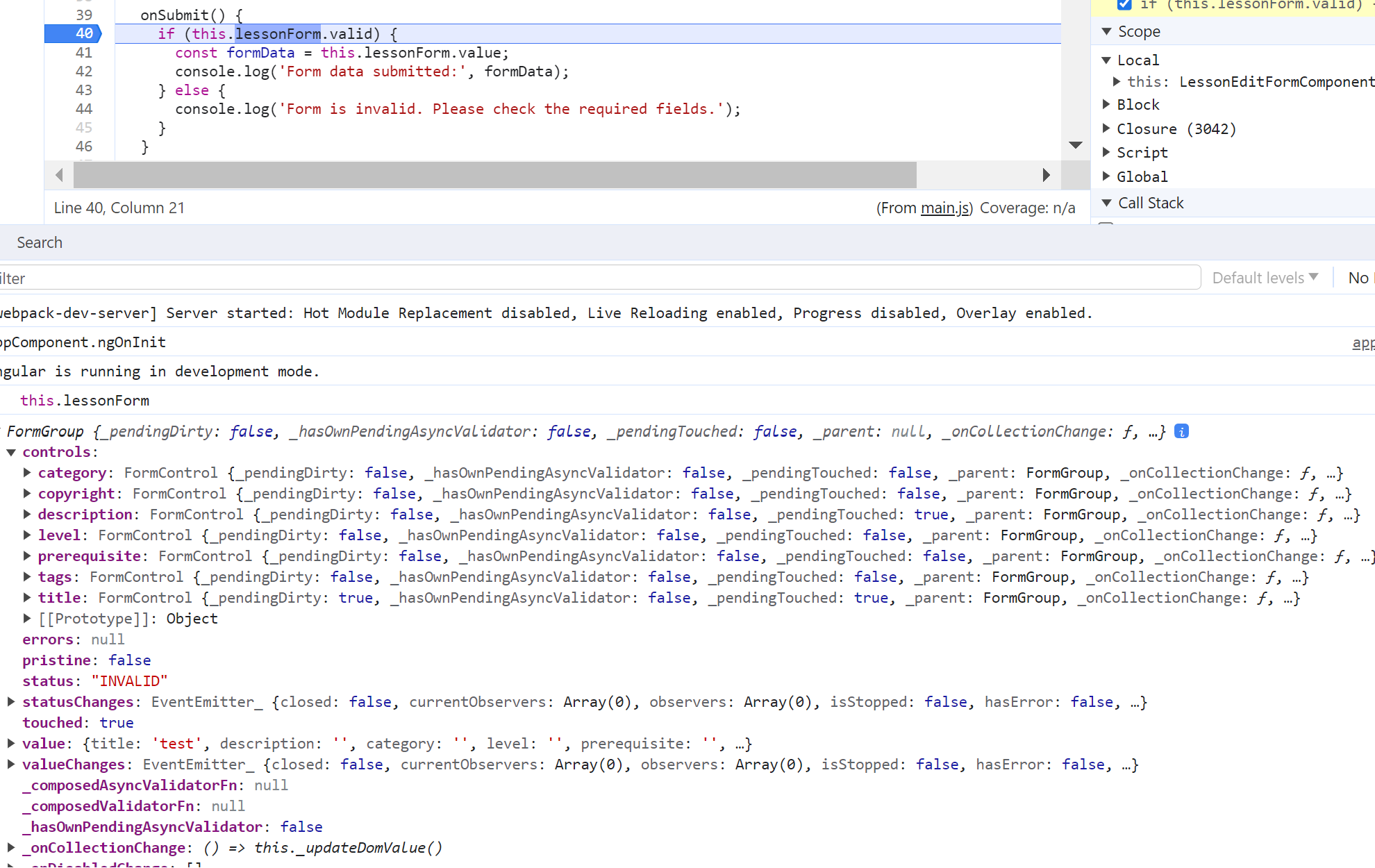
Then change your lesson-edit-form.component.html, to declare formControlName=”..” attributes, instead of binding [(ngModel)]=”..”

<input id="title" type="text" class="col-md-5"  
 formControlName="title">

<textarea id="description" class="col-11" rows="4"  
 formControlName="description"></textarea>

## Step 15: Debug for VALID / INVALID form

See in Chrome DevTool the content of object “this.lessonForm” when clicking on submit button

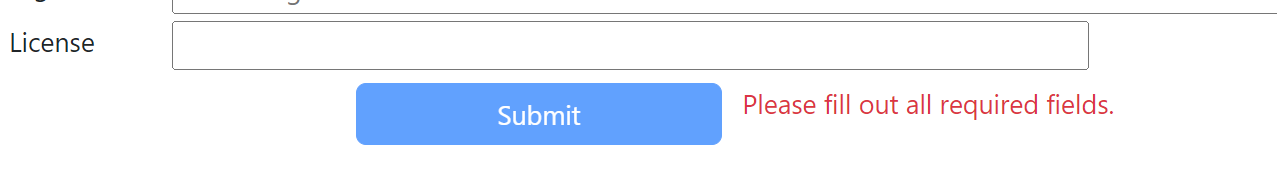


## Step 16: Disable the submit button when form is invalid

<button type="button" class="btn btn-primary" (click)="onSubmit()"  
 [disabled]="!lessonForm.valid"  
 >Submit</button>

## Step 17: add warning message when the form is invalid

<div class="row mt-2">  
 <label class="col-md-2"></label>  
 <button type="button" class="btn btn-primary col-md-2" (click)="onSubmit()"  
 [disabled]="!lessonForm.valid"  
 >Submit</button>  
 <span *\*ngIf*="lessonForm.invalid" class="col-md-8 text-danger">  
 Please fill out all required fields.  
 </span>  
</div>



Check that when the form is valid, the message disappears, and the submit button becomes enabled.

## Step 18: add classes to each label when corresponding field is invalid, to change it to red

<label class="col-md-1" for="title"  
 [class]="lessonForm.get('title')?.invalid ? 'text-danger' : ''"  
 >Title</label>

<label class="col-md-1" for="description"  
 [class]="lessonForm.get('description')?.invalid ? 'text-danger' : ''"  
 >Description</label>

