# TD 4-part 2 : Angular \*ngIf, \*ngFor, @Input, @Output field bindings, [(ngModel)] and ReactiveForm

## TD Objectives

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

This hands-on session 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). The goal is also to realize how simple it is to use 2-way (bi-directional) data binding, using the “[()]” (banana-box notation). In particular, we will use “[(ngModel)]” for <input> field. Then, we will see more in-depth binding with Form fields validation.

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: check \*ngIf feature as in course

## Step 2: check \*ngFor feature as in course

## Step 3: check using @Input() as in course

## Step 4: check using (click)=”onClick()” for built-in @Output on button

## Step 5: check using @Output() as in course

## Step 6: check using @Output() with “fieldChanged” naming convention, 2-way-binding

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

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

@NgModule({  
… lines ommited

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

## Step 8: add [(ngModel)] field binding

TODO

TODO

## Step 12 : re-open and finalize 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[];

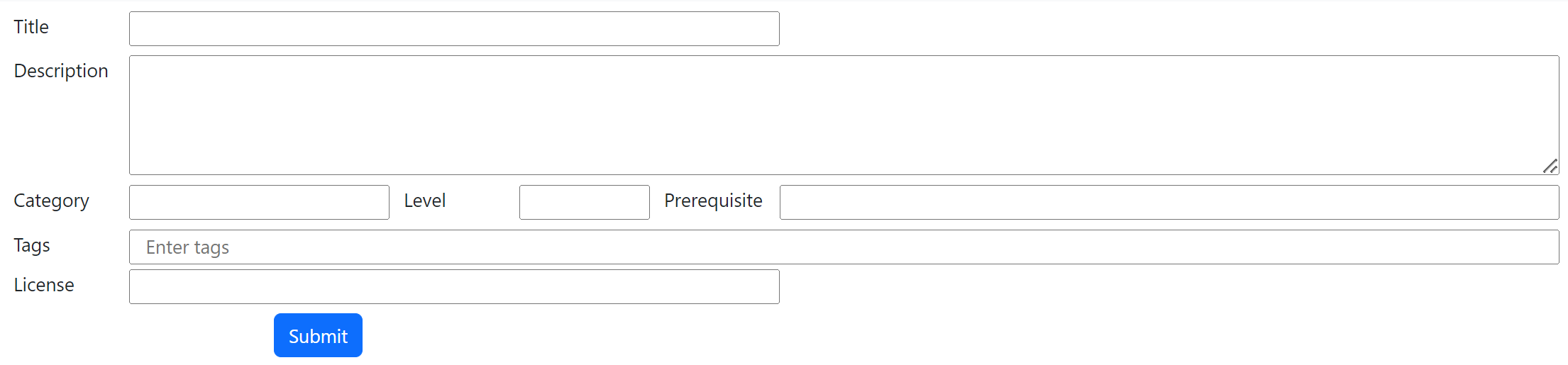
copyright: string;

}

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>

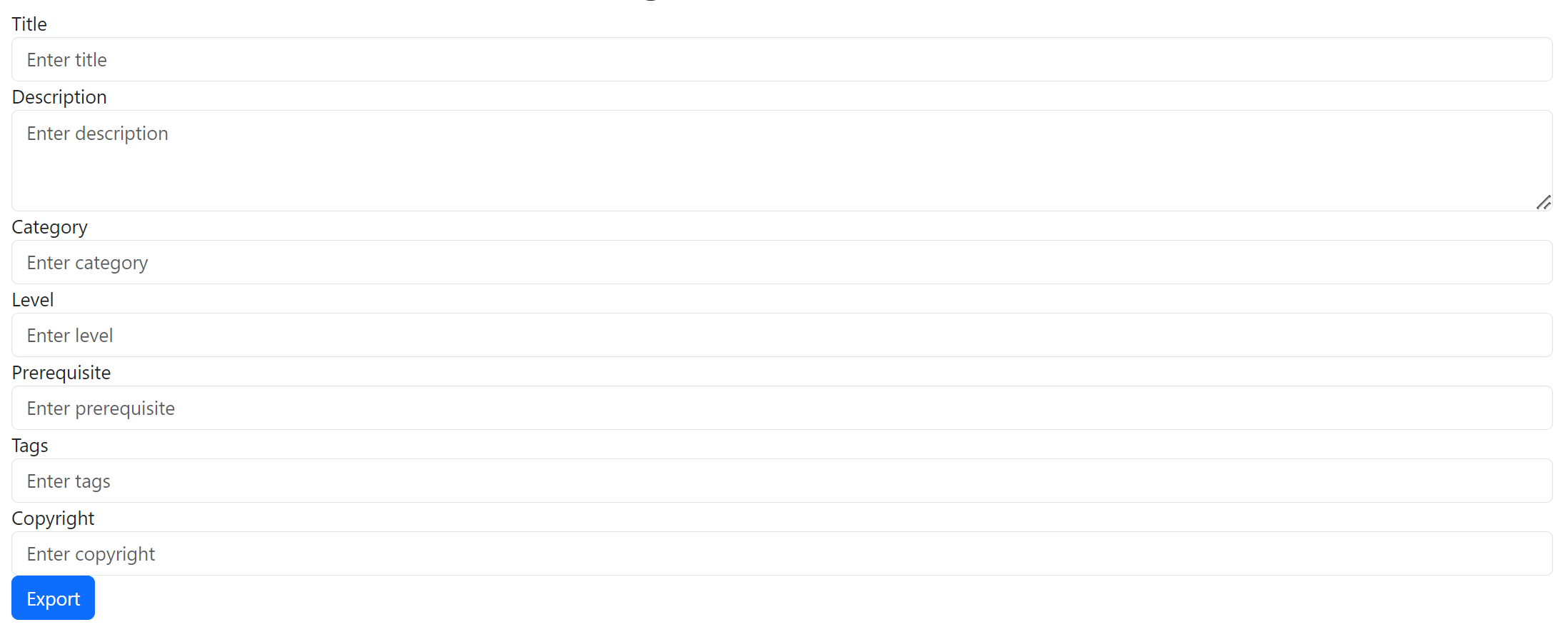
This renders as :



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> as the corresponding id attribute

<input id=”someFieldId”>

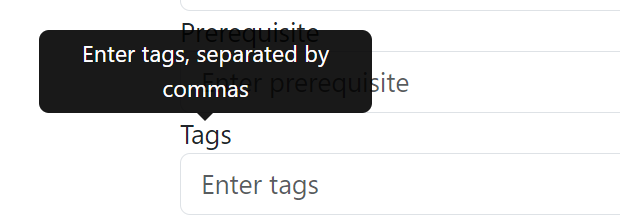
There could also be “aria-\*” attributes (aria means “Accessible Rich Internet Applications”), to improve accessibility particulary for users with disabilities.

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

Example for field “title: string”

<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 use :

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.');  
 }  
}

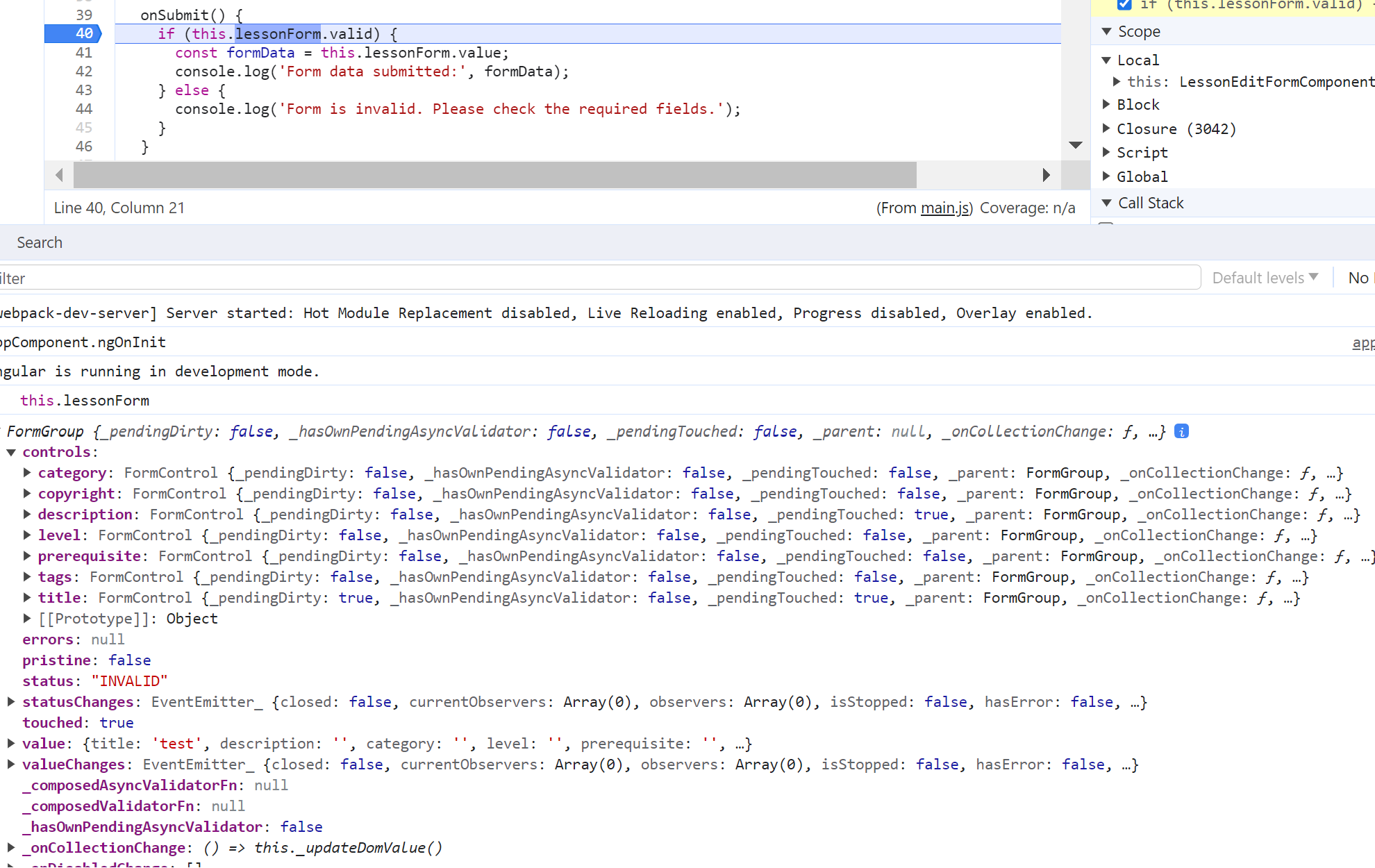
Then change your lesson-edit-form.component.html, to declare formControlName=”..” attributes, instead of binding [(ngModel)]=”..”

<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

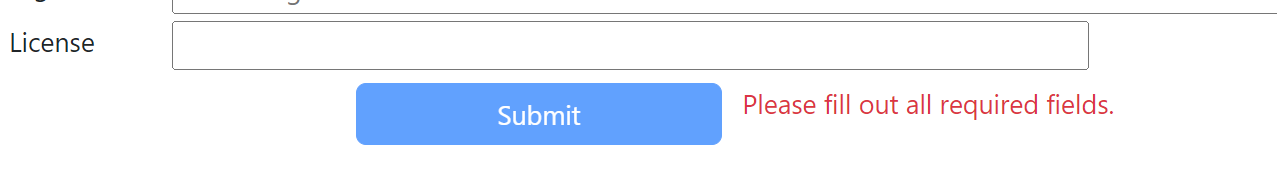


## Step 16: Disable the submit button when form is invalid

<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>

