Angular | Lecture 5

Marina Magdy

Agenda

- Recap last lecture points
- Forms
- Template driven forms
- Reactive forms
- Validation

Forms

Forms

Angular provides two different approaches to handling user input through forms: reactive and template-driven. Both capture user input events from the view, validate the user input, create a form model and data model to update, and provide a way to track changes.

Docs: https://angular.dev/guide/forms

Forms (Reactive vs Template Driven)

- **Reactive forms** provide direct, explicit access to the underlying forms object model. Compared to template-driven forms, they're more scalable, reusable, and testable. If forms are a key part of your application use reactive forms.
- **Template-driven forms** rely on directives in the template to create and manipulate the underlying object model. They are useful for adding a simple form to an app, such as an email list signup form. They're easy to add to an app, but they don't scale as well as reactive forms. If you have very basic form requirements and logic that can be managed solely in the template, template-driven forms could be a good fit.

Template-driven Forms

Forms : Template driven forms

- Template-driven forms use two-way data binding to update the data model in the component as changes are made in the template and vice versa.
- Template-driven forms rely on directives defined in the FormsModule.
- The NgModel directive reconciles value changes in the attached form element with changes in the data model, allowing you to respond to user input with input validation and error handling.
- The NgForm directive creates a top-level FormGroup instance and binds it to a <form> element to track aggregated form value and validation status. As soon as you import FormsModule, this directive becomes active by default on all <form> tags. You don't need to add a special selector.

Forms : Template driven forms

To start using Form in Angular we need to explicitly import them in you component imports array.

```
import { FormsModule } from '@angular/forms';
@Component({
  selector: 'app-add-form',
  standalone: true,
  imports: [FormsModule],
   …
})
```

Forms : Template driven forms

The ngForm does the following

- The form is set up using ngForm directive
- controls are set up using the ngModel directive
- Use ngModel to create two-way data bindings for reading and writing input-control values.
- The Validations are configured in the template via directives

Forms : Template driven forms

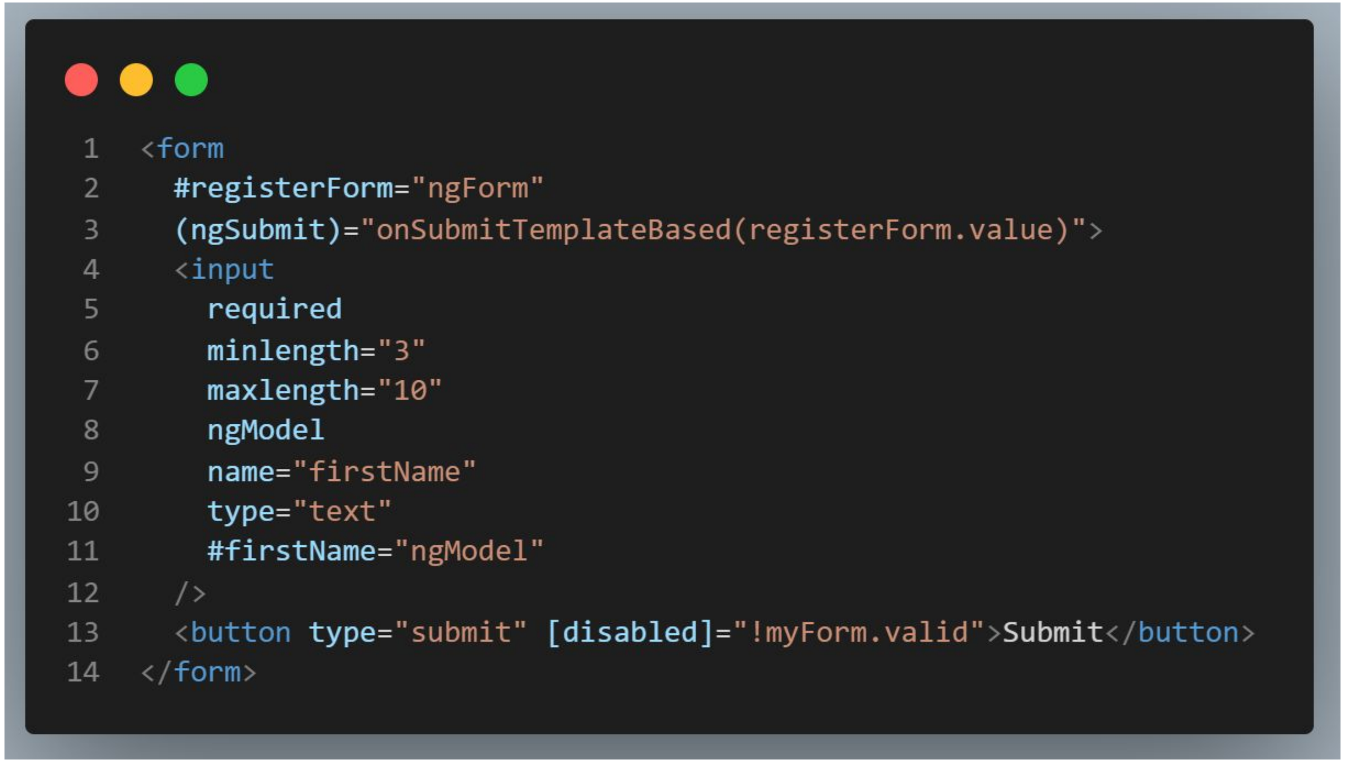
- **<form #registerForm="ngForm">**

  The ***registerForm*** template variable is now a reference to the NgForm directive instance that governs the form as a whole.

- **<input #username="ngModel" ngModel name="username" />**

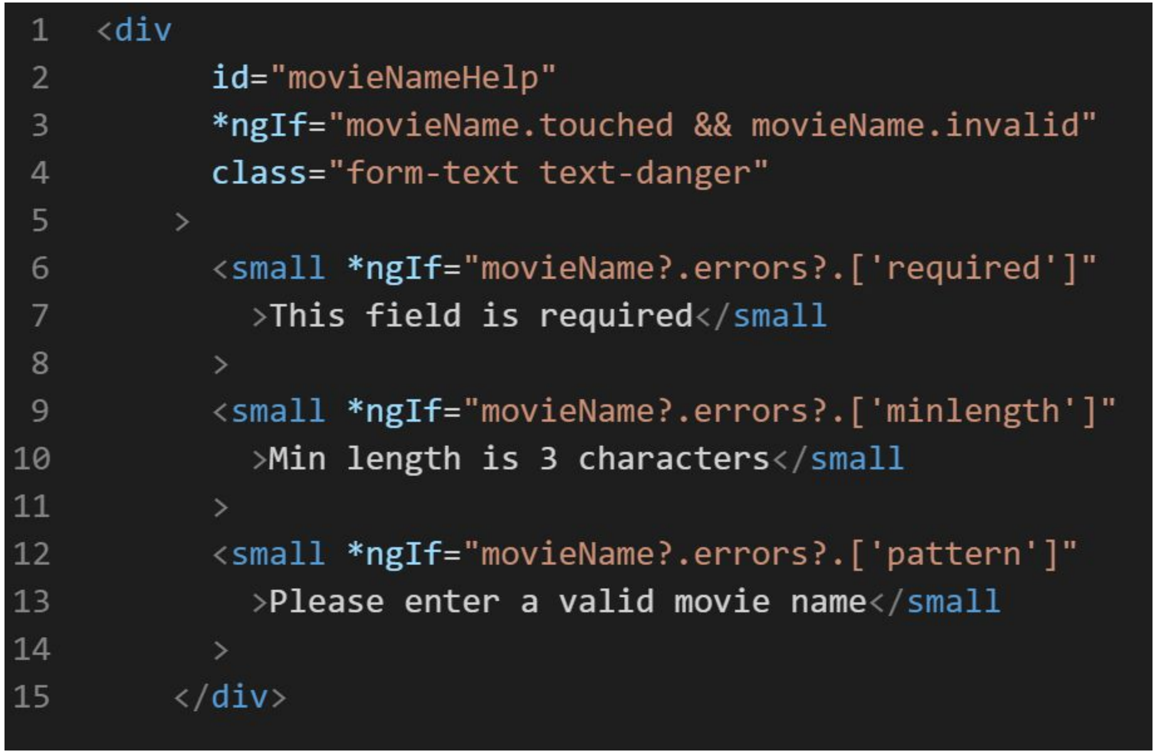
  When you use ngModel on an element, you must define a name attribute for that element. Angular uses the assigned name to register the element with the NgForm directive attached to the parent <form> element.

Forms : Template driven forms



```html
<form
  #registerForm="ngForm"
  (ngSubmit)="onSubmitTemplateBased(registerForm.value)">
  <input
    required
    minlength="3"
    maxlength="10"
    ngModel
    name="firstName"
    type="text"
    #firstName="ngModel"
  />
  <button type="submit" [disabled]="!myForm.valid">Submit</button>
</form>
```

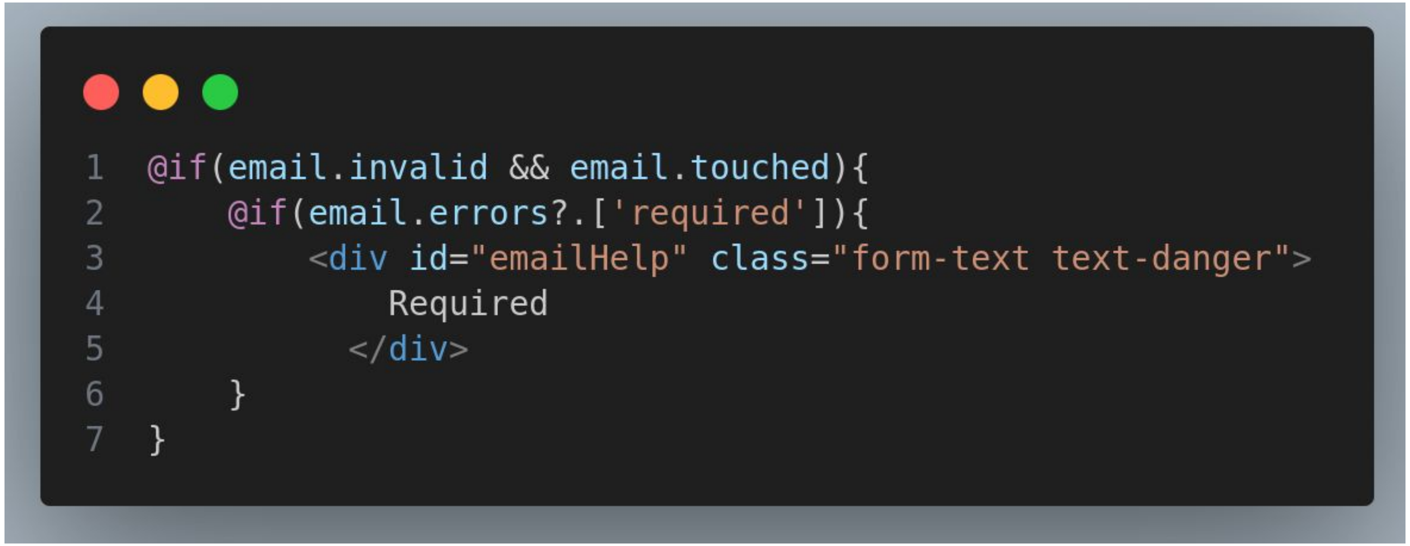
Forms : Template driven forms | Handling Errors



```
<div
      id="movieNameHelp"
      *ngIf="movieName.touched && movieName.invalid"
      class="form-text text-danger"
    >
      <small *ngIf="movieName?.errors?.['required']"
        >This field is required</small
      >
      <small *ngIf="movieName?.errors?.['minlength']"
        >Min length is 3 characters</small
      >
      <small *ngIf="movieName?.errors?.['pattern']"
        >Please enter a valid movie name</small
      >
    </div>
```

Forms : Template driven forms | Handling Errors

Or if using angular v17 use directly the built in control flow



```
@if(email.invalid && email.touched){
    @if(email.errors?.['required']){
        <div id="emailHelp" class="form-text text-danger">
            Required
          </div>
    }
}
```

Reactive Forms

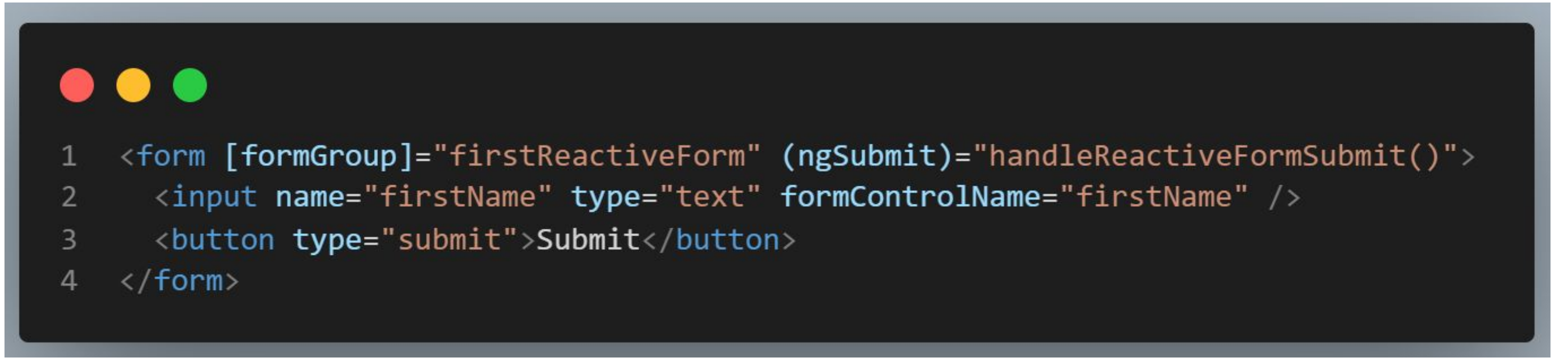
Forms : Reactive forms

To start using Form in Angular we need to explicitly import them in you component imports array.

```
import { ReactiveFormsModule } from '@angular/forms';
@Component({
  selector: 'app-add-form',
  standalone: true,
  imports: [ReactiveFormsModule],
   …
})
```

Forms : Reactive forms

Handle form with input and submit :



```
<form [formGroup]="firstReactiveForm" (ngSubmit)="handleReactiveFormSubmit()">
  <input name="firstName" type="text" formControlName="firstName" />
  <button type="submit">Submit</button>
</form>
```

Forms : Reactive forms

Let’s break it down:

- **formGroup:** The form will be treated as a FormGroup in the component class, so the formGroup directive allows to give a name to the form group.
- **ngSubmit:** This is the event that will be triggered upon submission of the form.
- **formControlName:** Each form field should have a formControlName directive with a value that will be the name used in the component class.

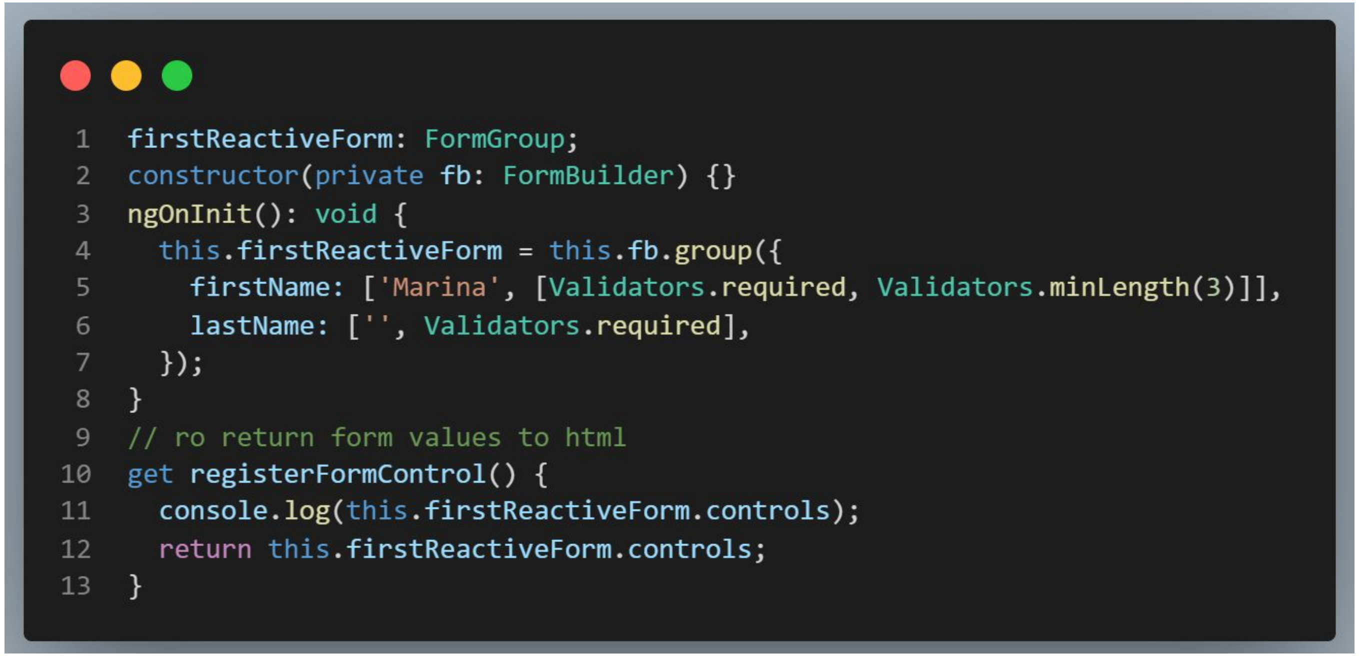
Forms : Reactive forms

Each formGroup contains multiple new FormControl with form fields:

```
this.userForm = new FormGroup({
      user_name: new FormControl(null, [Validators.required]),
});
```

Forms : Reactive forms

In .ts file :

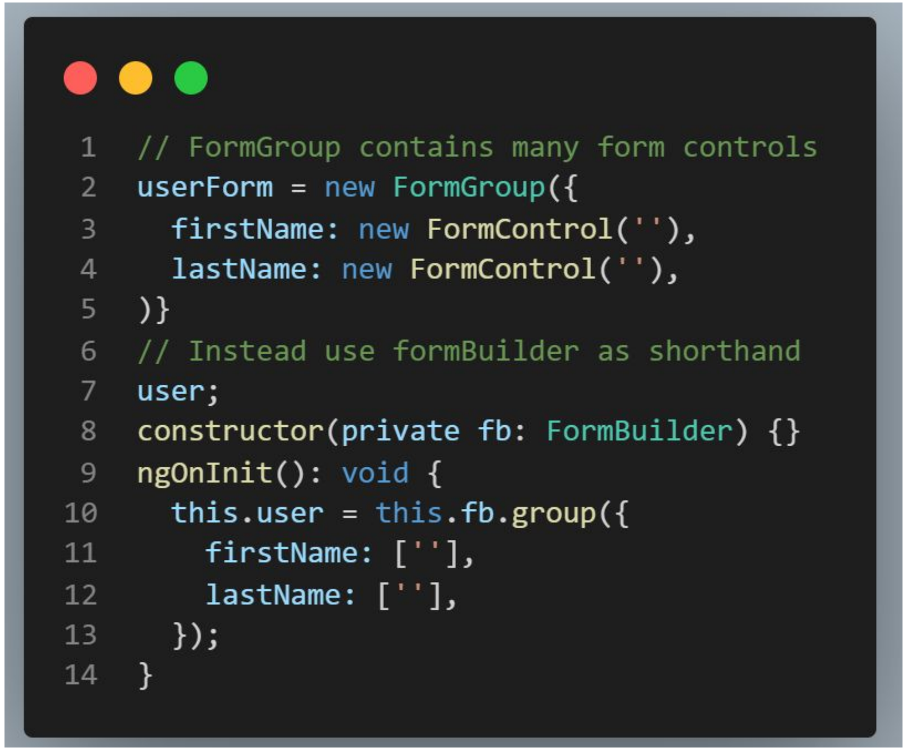


```
firstReactiveForm: FormGroup;
constructor(private fb: FormBuilder) {}
ngOnInit(): void {
  this.firstReactiveForm = this.fb.group({
    firstName: ['Marina', [Validators.required, Validators.minLength(3)]],
    lastName: ['', Validators.required],
  });
}
// ro return form values to html
get registerFormControl() {
  console.log(this.firstReactiveForm.controls);
  return this.firstReactiveForm.controls;
}
```

Forms : Reactive forms

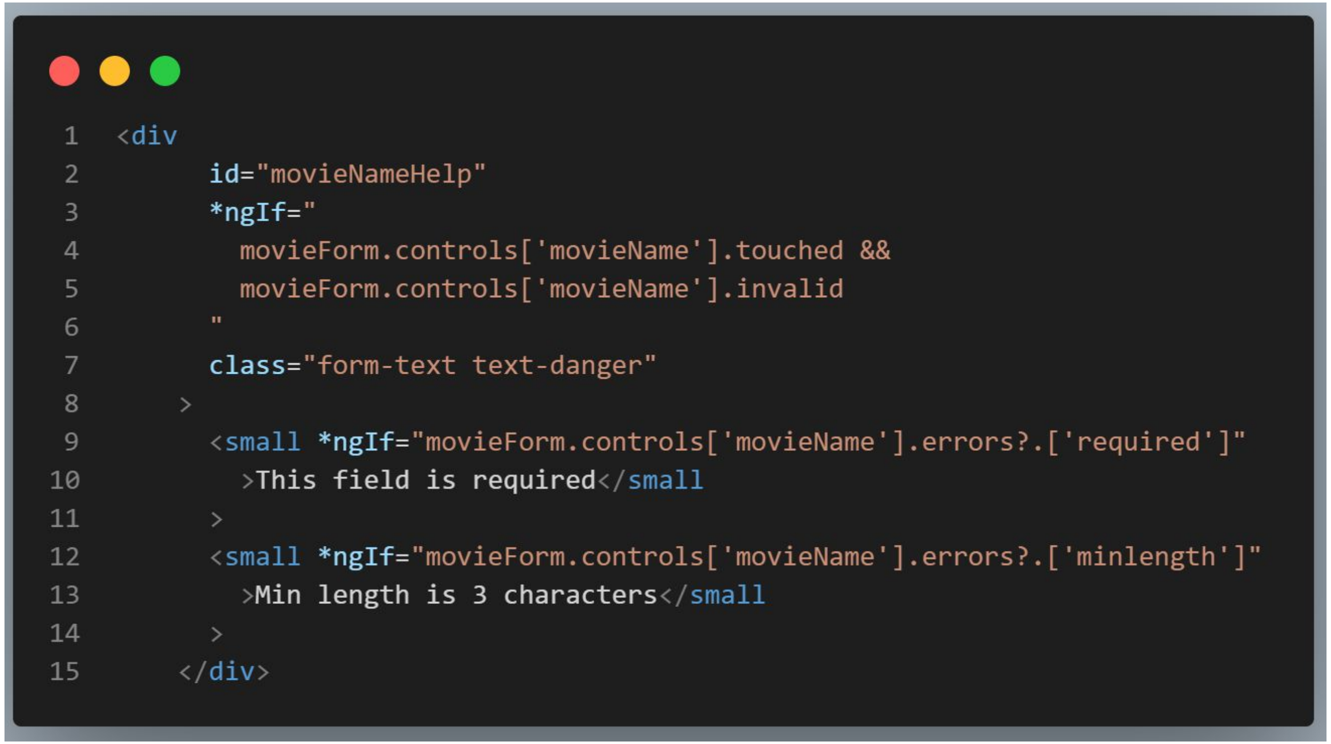
FormBuilder in Angular helps you streamline the process of building complex forms while avoiding repetition.

FormBuilder provides syntactic sugar that eases the burden of creating instances of FormControl, FormGroup, or FormArray and reduces the amount of boilerplate required to build complex forms.



```
// FormGroup contains many form controls
userForm = new FormGroup({
  firstName: new FormControl(''),
  lastName: new FormControl(''),
)}
// Instead use formBuilder as shorthand
user;
constructor(private fb: FormBuilder) {}
ngOnInit(): void {
  this.user = this.fb.group({
    firstName: [''],
    lastName: [''],
  });
}
```

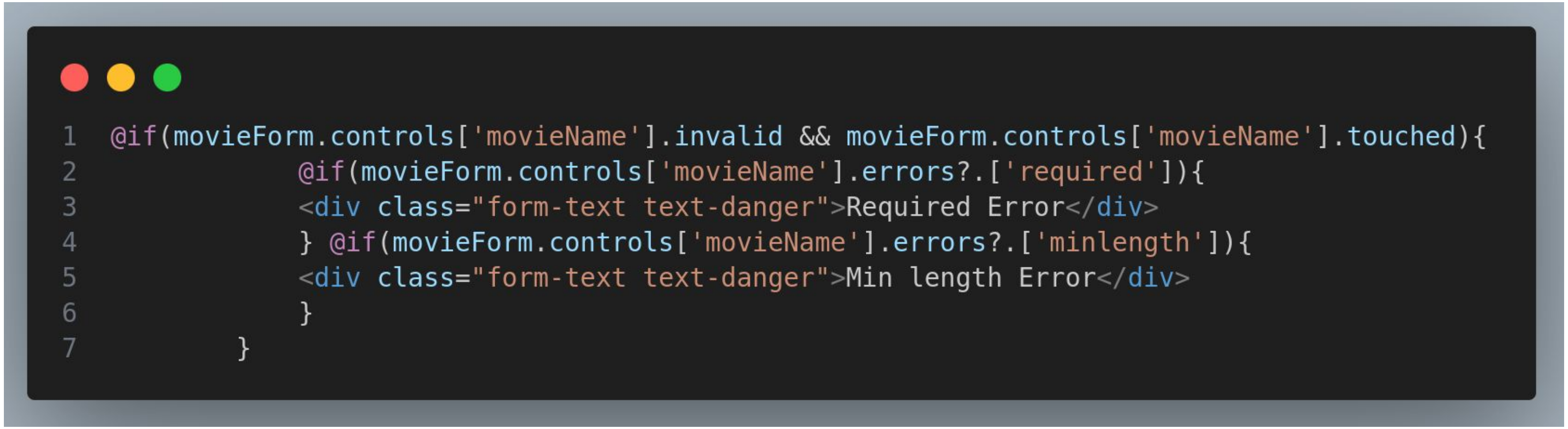
Forms : Reactive forms | Handling Errors



```
<div
      id="movieNameHelp"
      *ngIf="
        movieForm.controls['movieName'].touched &&
        movieForm.controls['movieName'].invalid
      "
      class="form-text text-danger"
    >
      <small *ngIf="movieForm.controls['movieName'].errors?.['required']"
        >This field is required</small
      >
      <small *ngIf="movieForm.controls['movieName'].errors?.['minlength']"
        >Min length is 3 characters</small
      >
    </div>
```

Forms : Reactive forms | Handling Errors

Or if using angular v17 use directly the built in control flow



```
@if(movieForm.controls['movieName'].invalid && movieForm.controls['movieName'].touched){
            @if(movieForm.controls['movieName'].errors?.['required']){
            <div class="form-text text-danger">Required Error</div>
            } @if(movieForm.controls['movieName'].errors?.['minlength']){
            <div class="form-text text-danger">Min length Error</div>
            }
        }
```

Forms : Reactive forms

What is the difference between setValue and patchValue? [SELF STUDY]

https://angular.dev/guide/forms/reactive-forms#updating-parts-of-the-data-model

Template Form

- Import **{FormsModule}**
- Started from HTML
- Define form reference **#form=”ngForm”**
- Define function to submit **(ngSubmit)=”function(form)”**
- Define fields , and connect fields with ngForm how!!!
- Each field => **name** , **ngModel** , define reference to field to be accessible **#field=ngModel**
- Add your html validations on input “Required, minlength…”
- Handle validations from reference **field.errors?.[‘required’] OR field.hasError(‘required’)**

Reactive Forms

- Import **{ReactiveFormsModule}**
- Started from TS file
- Define **FormGroup** , Initialize inside constructor formGroup has group of **FormControls || or use FormBuilder Syntax**
- Each control has Initial Value and Validators [‘’ , Validatiors]
- Connect html with TS, Property Binding **[FormGroup]=”formGroupName”**
- Define submit function **(ngSubmit)=”function()”**
- Each field should have **formControlName=”controlName’**

controlName should be same to the one defined in TS [FormControl]

- Handle validations from formGroup

FormName.controls[‘fieldName’].errors?.[‘required’]

Thank you

Lab

Products App

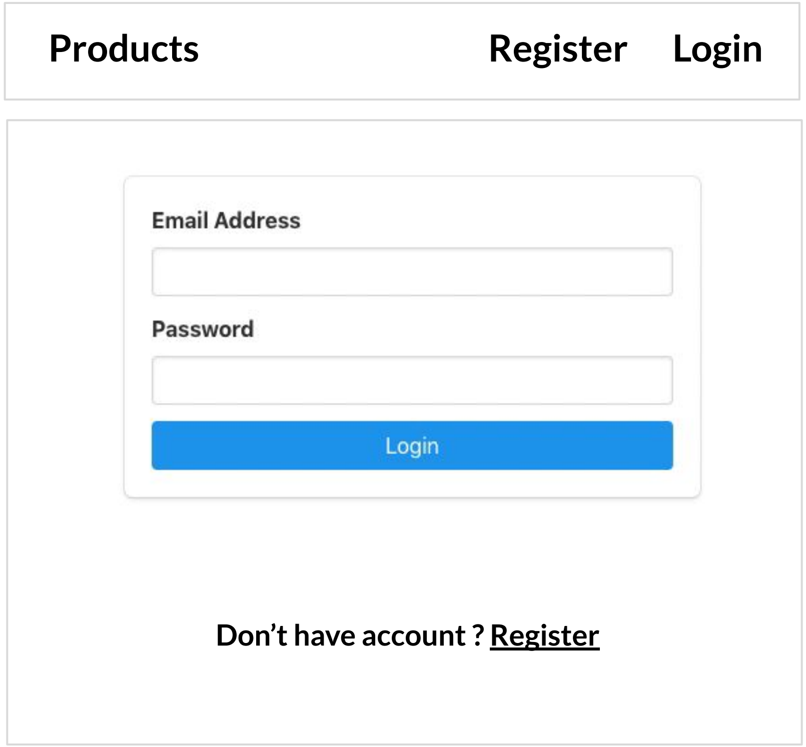
Create an auth app contains of 2 pages :

1. Login page
2. Register page

First step will contain login page with forms has 2 fields using **template driven** form :

- **Email address** [required - email format] [Using Regex]
- **Password** [required]

Bonus: Show validations and errors on Submit.



Products App

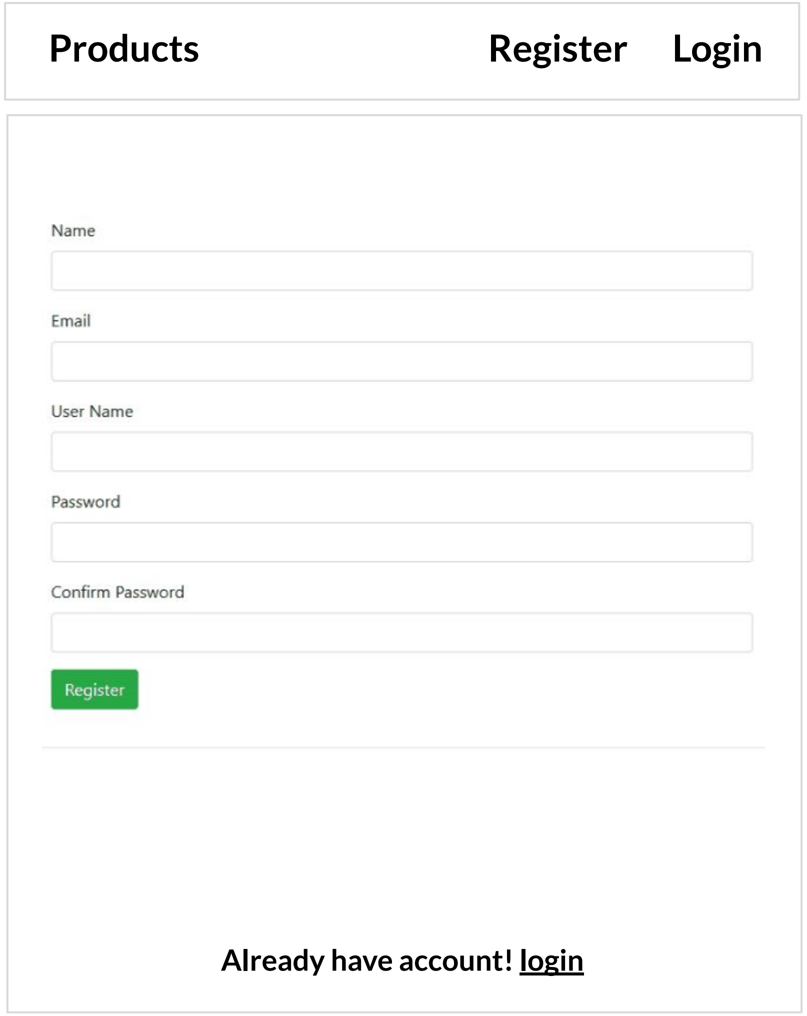
Second page is register page with the following fields using **Reactive form**:

- **Email address** [required - email format]
- Name [required]
- Username [required - contains no spaces]
- **Password** [required - password length not less than 8 characters , contains at least one lowercase , one uppercase , at least one digit and special character [example : *@%$#]]

Example for valid Password : **P@ssword1234**

- **Confirm password :** [required - matches previous password] [Bonus] [Custom Validators]

Bonus: Show validations and errors on Submit.


Products Register Login

Name

Email

User Name

Password

Confirm Password

Register

Already have account! login

Products App [Bonus]

Using REACTIVE FORMS, Build FORM ARRAY [Extra]

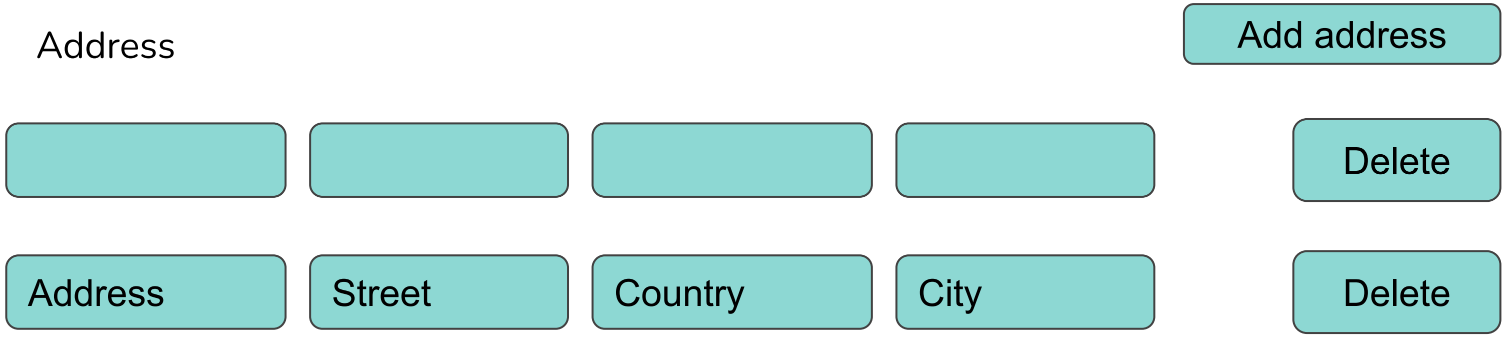
On the registration page use the dynamic forms to added multiple addresses for the user while register

Each address should have

- **Address**
- **Street**
- **Country**
- **City**

All Fields are Required.

Products App [Bonus]



https://blog.angular-university.io/angular-form-array/