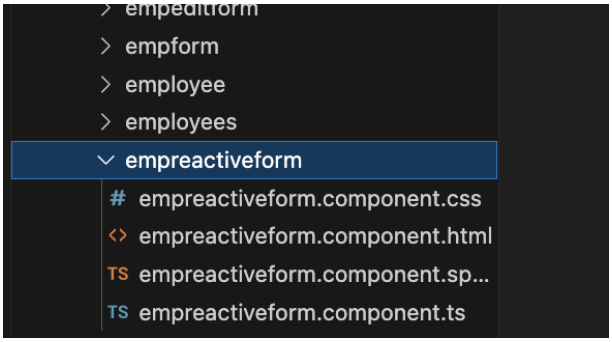
Reactive Driven Forms

Key concepts in Reactive Forms:

a. FormGroup: Represents a group of form controls. It aggregates the values of each control into a single object.
b. FormControl: Represents a single input field — a form control instance manages the value, validation status, and user interactions of an input field.
c. FormBuilder: A service that provides convenient methods for creating instances of FormGroup and FormControl.
d. Validators: Functions used for synchronous and asynchronous validation of form controls.

Steps to implement reactive forms:

1. Make sure to add **ReactiveFormsModule** in app.module.ts file
2. Add empreactiveform component from within the angulardemo folder as follows:
   ng g c empreactiveform



3. Update the ts component as follows :
   import { FormControl } from '@angular/forms';

   FormControl is a directive that allows you to create and manage a FormControl instance directly.

4. Add below property in the ts component
   name = new FormControl();

   Here you are creating a FormControl called name. It will be bound in the template to an HTML input box for the emp name.

   A FormControl constructor accepts three, optional arguments:
   a. the initial data value,
   b. an array of validators,
   c. and an array of async validators.

5. Add below in html component
```
<h2>Emp Detail</h2>
<h3><i>Just a FormControl</i></h3>
{{ name.value}}
 <label class="center-block">
      Name: <input class="form-control" [formControl]="name">
</label>
```

   To let Angular know that this is the input that you want to associate to the name FormControl in the class, you need [formControl]="name" in the template on the <input>.

6. Add <app-empreactiveform></app-empreactiveform> in app component html

7. The above is useful when there is onle 1 or 2 form controls. To. Manage a group of form elements we need FormGroup

8. Import following in the component :
   import { FormControl, FormGroup } from '@angular/forms';

9. In the class, wrap the FormControl in a FormGroup
   empForm = new FormGroup ({ name: new FormControl() });

10. Update the template to reflect FormGroup.

```
<form [formGroup]=”empForm" novalidate>
    <div class="form-group">
    <label class="center-block">Name: <input class="form-control" formControlName="name"> </label>
    </div>
</form>
```

11. To see the form model, add the following line after the closing form tag.

```
    <p>Form value: {{ empForm.value | json }}</p>
     <p>Form status: {{ empForm.status | json }}</p>
```

12. For using Validators:
    For using Validators.required in reactive forms import the Validators symbol.
    import { FormBuilder, FormGroup, Validators } from '@angular/forms';

13. To make the name FormControl required, replace the name property in the FormGroup with an array.
    The first item is the initial value for name; the second is the required validator, Validators.required.
    name: ['', Validators.required],

14. Update the empreactive form and html as follows for custom validation messages

.ts file :

```
 name:FormControl = new FormControl('Shalini', Validators.required);

 form:FormGroup ;

 constructor(){
  console.log(this.name)
  this.form = new FormGroup({
   name:this.name,
   phone:new FormControl('', [Validators.required, Validators.pattern(/^\d{10}$/)])
  })
 }
```

.html file:

```
<div class="mb-4">
   <h3>Reactive Driven Form Demo</h3>
   <p>{{form.value | json}}</p>
   <p>{{form.status | json}}</p>
   <form [formGroup]="form">
   <!-- <input type="text" placeholder="Enter"
   name="name" class="form-control" [formControl]="name" /> -->
   <input type="text" placeholder="Enter"
   name="name" class="form-control" formControlName="name" />

   <div *ngIf="name.invalid && name.errors?.['required']">
      <span class="text-danger">Name is required</span>
    </div>

   <input type="text" placeholder="Enter"
   name="phone" class="form-control" formControlName="phone" />

   <div *ngIf="form.controls['phone'].invalid">
      <div *ngIf="form.controls['phone'].errors?.['required']">
         <span class="text-danger">Phone is required</span>
      </div>
      <div *ngIf="form.controls['phone'].errors?.['pattern']">
         <span class="text-danger">Phone should be 10 digits only</span>
      </div>
```

```
    </div>

    </form>
</div>
```

15. Can update all the above code to create form with in-built and custom validations

16. Create a folder validators and a file password.ts file within it. Add below code to create custom validation for password

```
import { AbstractControl, FormControl } from "@angular/forms";

export function hasExclamationMark(input: AbstractControl) {
    const hasExclamation = input.value && input.value.indexOf('!') >= 0;
    return hasExclamation ? null : { needsExclamation: true };
  }
```

17. Update the ts component as follows:

```
  subsemail:FormControl;
  empform:FormGroup;
  email:FormControl;
  address:FormGroup
  city:FormControl
  password:FormControl;
  constructor() {
    this.subsemail = new FormControl(",Validators.required);
    this.email=new FormControl(",Validators.required);
    this.city = new FormControl(",Validators.required);
    this.address = new FormGroup({city:this.city})
    this.password=new FormControl(",hasExclamationMark)
    console.log(this.subsemail)
    this.empform = new FormGroup({
      ename:new FormControl('Sample name',[Validators.required, Validators.minLength(5)]),
      email:this.email,
      address:this.address,
      password:this.password
    })
  }
  subscribe()
  {
    console.log(this.subsemail.value)
  }
onSubmit() {
    console.log('Form values:', this.empform.value);
    this.empform.reset();
  }
```

18. Update the HTML as follows:

```
<h1>Reactive Forms</h1>
<h3>{{empform.value | json}}</h3>
<h3>{{empform.status }}</h3>
<form [formGroup]="empform" (ngSubmit)="onSubmit()>
    <div class="mb-3">
        <input type="text" class="form-control"
        [ngClass]="{'is-invalid':empform.controls['ename'].invalid && empform.controls['ename'].touched}"
        formControlName="ename" placeholder="Enter name"/>
        <div *ngIf="empform.controls['ename'].invalid && empform.controls['ename'].touched">
            <span class="text-danger" *ngIf="empform.controls['ename'].hasError('required')">Name is
required</span>
            <span class="text-danger" *ngIf="empform.controls['ename'].hasError('minlength')">length 5</span>
        </div>

    </div>
    <div class="mb-3">
```

```
        <input type="text" class="form-control" [ngClass]="{'is-invalid':email.invalid && email.touched}"
        formControlName="email" placeholder="Enter email"/>
        <div *ngIf="email.invalid && email.hasError('required')">
          <span class="text-danger">Email is required</span>
        </div>
      </div>
 <div class="mb-3">
        <input type="text" class="form-control" [ngClass]="{'is-invalid':password.invalid &&
password.touched}"
        formControlName="password" placeholder="Enter password"/>
        <div *ngIf="password.invalid && password.hasError('needsExclamation')">
          <span class="text-danger">! is required</span>
        </div>
      </div>
        <div [formGroup]="address">
        <div class="mb-3">
          <input type="text" class="form-control"
          formControlName="city" placeholder="Enter city"/>
          <div *ngIf="city.invalid && city.hasError('required')">
            <span class="text-danger">It is required</span>
          </div>
        </div>
      </div>
    <button type='submit'>Submit</button>
</form>
<input type="text" class="form-control"
[formControl]="subsemail"
placeholder="enter email to subscribe">
<button (click)="subscribe()">Subscribe</button>
```

19. Form Control properties:

    In reactive forms, FormControl is used to manage the value and validation state of an individual form control. It provides methods to set and get the value, update the validation status, and listen to value changes.

```
nameControl = new FormControl('');


constructor() {
  this.nameControl.valueChanges.subscribe(value => {
    console.log('Value changed:', value);
  });
}
}
```

20. **Form Array and Form Builder :**

    FormArray is used to handle the controls dynamically and store all the controls in one unit like Array or List. FormArray is a collection of Controls like formControl, formGroup, or another formArray and we can able to add, remove, and manipulate these controls dynamically

    We can group Form Controls in Angular forms in two ways.

    One is using the FormGroup and the other one is FormArray. The difference is how they implement it.
    In FormGroup controls becomes a property of the FormGroup. Each control is represented as key-value pair.
    While in FormArray, the controls become part of an array

DEMO using FormArray

a. Import FormBuilder from the @angular/forms
b. Add below property in ts file:

```
skillsForm: FormGroup;
```

c. Inject FormBuilder in constructor to initiate the skillsForm in the constructor as follows:

```
constructor(private fb:FormBuilder) {

      // previous code as it is
   this.skillsForm = this.fb.group({
    name: '',
    skills: this.fb.array([]) ,
   });

 }
```

d. Next, a getter method skills, which returns the skills FormArray from the skillsForm

```
get skills() : FormArray {
  return this.skillsForm.get("skills") as FormArray
}
```

e. We need to capture two fields under each skill. Name of the skill & years of exp. Hence we create a FormGroup with two fields. The method newSkill creates a new FormGroup and returns it. Note that we won't be able to assign a name to Form Group.

```
newSkill(): FormGroup {
   return this.fb.group({
     skill: '',
     exp: '',
   })
}
```

f. We need to add a new skill to the skills FormArray. Since it is an array we can use the push method to add the new skill using the the newSkill method. Note that newSkill() method returns a FormGroup. The name of the FormGroup is its Index in the FormArray.

```
addSkills() {
   this.skills.push(this.newSkill());
}
```

g. Use the removeAt method to remove the element from the skills FromArray.

```
removeSkill(i:number) {
  this.skills.removeAt(i);
}
```

h.
```
onSubmitSkills() {
   console.log(this.skillsForm.value);
}
```

i. Update template as follows:

```
<h1>Skills Form </h1>

<form [formGroup]="skillsForm" (ngSubmit)="onSubmitSkills()">

 <p>
   <label for="name">Name </label>
   <input type="text" id="name" name="name" formControlName="name">
 </p>

 <p>
   <button type="submit">Submit</button>
```

```
  </p>

</form>
```

j. Use the formArrayName directive to bind the skills form array to the div element. Now the div and anything inside the div element is bound to the skills form array. Add below code before the submit button

```
Skills :
<div formArrayName="skills">

</div>
```

k. Inside the div use ngFor to loop through each element of skills FormArray. let i=index will store the index value of the array in template local variable i. We will make use of it to remove the element from the skills array

```
<div formArrayName="skills">
  <div *ngFor="let skill of skills.controls; let i=index">

  </div>
</div>
```

l. Each element under the skills is a FormGroup. We do not have a name to the FormGroup. The Index of the element is automatically assigned as the name for the element.

Hence we use the [formGroupName]="i" where i is the index of the FormArray to bind the FormGroup to the div element.

```
<div formArrayName="skills">
  <div *ngFor="let skill of skills().controls; let i=index">
     <div [formGroupName]="i">

     </div>
  </div>
</div>
```

m. Finally, we add the controls using the formControlName directive.

```
 Skills:
 <div formArrayName="skills">
   <div *ngFor="let skill of skills().controls; let i=index">
     <div [formGroupName]="i">
       {{i}}
       skill name :
       <input type="text" formControlName="skill">
       exp:
       <input type="text" formControlName="exp">

       <button (click)="removeSkill(i)">Remove</button>
     </div>
   </div>
 </div>
```

n. Also, pass the index i to removeSkill

```
<button (click)="removeSkill(i)">Remove</button>
```

o. Finally, call the addSkills method to add new skills.

```
<p>
  <button type="button" (click)="addSkills()">Add</button>
</p>
```

p. Add below in the last to check the form status:

{{this.skillsForm.value | json}}

https://angular.dev/guide/forms/reactive-forms
https://www.tektutorialshub.com/angular/formgroup-in-angular/ [In detail]