AngularForms

- Form is a container that encapsulates a set of elements and provide an UI from where user can interact with our application.
- Technically form comprises of elements like buttons, textbox, radios, dropdown list etc.
- A form allows a user to query and submit the data to server.
- Angular can give a dynamic behaviour for forms so that they can handle client side interactions and validations.
- Angular forms are classified in to 2 types : -
  1. Template Driven Forms.
  2. Reactive Forms.

Template Driven Forms

- A Template driven form is a dynamic form that can handle client side interactions.
- The configuration for a dynamic form is defined at Template level i.e. in HTML.
- Most of the interactions are dynamically handled at UI-Level.
- These forms are heavy in page and take more time in rendering.
- The forms and their elements are configured by using directives: -
  1. ngForm.
  2. ngModel.
- ngForm is defined in “forms-module” and used to handle <form> element dynamically.
- ngModel is also in “forms-module” and used to handle input element dynamically. [text, password, select…]
- SYNTAX:-
  <form #referenceName=”ngForm” >
  <input type=”text” #txtName=”ngModel” ngModel name=”txtName”>
  </form>
  Example:
  1. Go to ‘app.module.ts ‘ and import

```
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';

imports: [

BrowserModule,
    FormsModule,
  ]
```

2. Add a new component

```
>ng g c templateform –-spec=false.
```

3. templateform.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-templateform',
  templateUrl: './templateform.component.html',
  styleUrls: ['./templateform.component.css']
})
export class TemplateformComponent  {
public SubmitClick(obj) {
alert(obj.txtName + ' is shipped to ' + obj.shippedTo + '.');
}
}
```

4.templateform.component.html

```
<div class="container">
  <h2>Register Products</h2>
  <form #frmRegister="ngForm" (submit)="SubmitClick(frmRegister.value)" class="formBody">
    <div class="form-group">
      <label>Name</label>
      <div>
        <input name="txtName" type="text" #txtName="ngModel" ngModel class="form-control">
      </div>
    </div>
    <div class="form-group">
      <label>Shipped To</label>
      <div>
        <select name="shippedTo" #shippedTo="ngModel" ngModel class="form-control">
          <option>HYD</option>
          <option>Mumbai</option>
        </select>
      </div>
    </div>
    <button class="btn btn-primary">Submit</button>
```

```
        </form>
        <div class="round">
          Name: {{frmRegister.value.txtName | uppercase}}
            <br>
          ShippedTo: {{frmRegister.value.shippedTo}}
        </div>
      </div>
```

5. <u>templateform.component.css</u>

```
.round{
  background-color: lightcyan;
  border: 2px solid black;
  width: 400px;
  height: 150px;
  border-radius: 20px;
  padding: 20px;
  box-shadow: 0 0 15px 15px;
  color: darkcyan;
  margin-top: 20px;
  font-family: 'Times New Roman', Times, serif;
  margin-left: 300px;
}
.formBody{
  border: 2px solid darkcyan;
  padding: 10px;
  border-radius: 20px;
  box-shadow: 0 0 8px 8px;
}
```

Angular Form Validation

- Validation is the process of verifying user input.
- Validation is required to ensure that any contra dictionary and unauthorized data is not get stored in the database.
- Client side validations can be handled by using patterns, regular expressions and functions.
- Angular provides pre-defined services to handle validations.
- The Angular validation services are categorized in to 2 types: -
    1. Form State validation services.
    2. Input State validation services.

Form-state Validation :

The form state validation services can verify all input fields in a form simultaneously at the same time and return a Boolean value.

The various form state services are :-

SERVICE	PROPERTY	DESCRIPTION
ngPristine	Pristine	- It verifies whether any field in the form have modified its value with user input. - It returns true when no field modified.
ngDirty	dirty	-It returns Boolean true when any one field has been modified.
ngInvalid	invalid	-It returns true when at least one field is invalid.
ngValid	valid	-It returns true when all fields are valid.
ngSubmitted	submitted	-It returns true when form is submitted.

SYNTAX:- formName.invalid
formName.pristine

Example:

1. Add a new component
   >ng g c formstatevalidation –-spec=false

2. <u>formstatevalidation.component.html</u>

```
<div class="container">
    <div class="boxStyle">
        <h3>Register</h3>
        <form novalidate name="frmRegister" #frmRegister="ngForm">
            <div class="form-group">
                <label>User Name</label>
                <div>
                    <input type="text" name="txtName" ngModel #txtNa
me="ngModel" class="form-control" required>
                </div>
            </div>
            <div class="form-group">
                <label>Mobile Number</label>
                <div>
```

```
                    <input type="text" placeholder="+91-
xxxxxxxxxx" name="txtMobile" ngModel #txtMobile="ngModel" class="for
m-control" required pattern="\+91[0-9]{10}">
                </div>
            </div>
            <div class="form-group">
                <button [disabled]="frmRegister.invalid" class="btn
btn-primary">submit</button>
            </div>
        </form>
    </div>
    <div class="boxStyle">
        <h3>Form State Services</h3>
        <dl>
            <dt>Pristine- No field Modified</dt>
            <dd>{{frmRegister.pristine}}</dd>
            <dt>Dirty-At least one field Modified </dt>
            <dd>{{frmRegister.dirty}}</dd>
            <dt>Invalid- At least one field invalid</dt>
            <dd>{{frmRegister.invalid}}</dd>
            <dt>Valid-All fields are valid</dt>
            <dd>{{frmRegister.valid}}</dd>
            <dt>Submitted- Form submitted</dt>
            <dd>{{frmRegister.submitted}}</dd>
        </dl>
    </div>
</div>
```

3. **formstatevalidation.component.css**

```
.boxStyle{
    width: 350px;
    height: 350px;
    float: left;
    margin: 20px;
    padding: 20px;
    border: 2px solid darkcyan;
    border-radius: 20px;
    box-shadow: 0 0 10px 10px;
}
```

Input state Validation Services:

- Angular provides predefined services that are used to verify the state of every field individually.
- The input validation services are

SERVICE	PROPERTY	DESCRIPTION
ngTouched	Touched	- It returns true when form

		elements gets focus & blurred.
ngUntouched	Untouched	- It returns true when the input fields never focused.
ngPristine	Pristine	- It verifies whether any field in the form have modified its value with user input. - It returns true when no field modified.
ngDirty	dirty	-It returns Boolean true when any one field has been modified.
ngInvalid	invalid	-It returns true when at least one field is invalid.
ngValid	valid	-It returns true when all fields are valid.
errors	errors	-It is an input field object that contains collection of error properties that are used to verify specific error.

SYNTAX: fieldName.invalid
fieldname.errors.required

NOTE:

Error object can be invoked only when input field is invalid.

Example:

1. Inputvalidation.component.html

```
<div class="container">
        <div class="boxStyle">
            <h3>Register</h3>
            <form novalidate name="frmRegister" #frmRegister="ngForm">
                <div class="form-group">
                    <label>User Name</label>
                    <div>
                        <input [ngClass]="{validStyle:txtName.valid,inv
alidStyle:txtName.invalid}" type="text" name="txtName" ngModel #txtName
="ngModel" class="form-control" required>
                        <span *ngIf="txtName.invalid && frmRegister.sub
mitted" class="text-danger" >Name Required</span>
                    </div>
                </div>
                <div class="form-group">
                    <label>Mobile Number</label>
                    <div>
                        <input [ngClass]="{validStyle:txtMobile.valid,i
nvalidStyle:txtMobile.invalid}" type="text" placeholder="+91-
xxxxxxxxxx" name="txtMobile" ngModel #txtMobile="ngModel" class="form-
control" required pattern="\+91[0-9]{10}">
                    </div>
                    <div *ngIf="txtMobile.invalid && frmRegister.submit
ted">
                        <span *ngIf="txtMobile.errors.required" class="
text-danger">Mobile required</span>
                        <span *ngIf="txtMobile.errors.pattern" class="t
ext-danger">Invalid Mobile</span>
                    </div>
                </div>
                <div class="form-group">
                    <button  class="btn btn-success">submit</button>
                </div>
            </form>
        </div>
    </div>
```

Dynamically CSS effects for validation

- You can define CSS classes for validation errors.
- You can dynamically apply classes by using [ngClass] directive.
- It requires to use an object reference to turn ON/OFF any CSS classes.
- The Boolean value is sent by using validation services.
- SYNATX:

  <input [ngClass]= “{className:txtName.valid}”>

  Example:

2. Go to **Inputvalidation.component.css**

```
.boxStyle{
    width: 600px;
    height: 350px;
    float: left;
    margin: 20px;
    padding: 20px;
    border: 2px solid darkcyan;
    border-radius: 10px;
}
label{
    color: rgb(41, 29, 4);
}
.validStyle{
    border:1px solid green;
}
```

```
.invalidStyle{
    border:1px solid red;
}
```

3. Go to **Inputvalidation.component.html**

```
<input [ngClass]="{validStyle:txtMobile.valid, invalidStyle:txtMobile.invalid}" type="text" placeholder="+91-xxxxxxxxxx" name="txtMobile" ngModel #txtMobile="ngModel" class="form-control" required pattern="\+91[0-9]{10}">
```

Angular Built-in Validation CSS classes

- Angular provides a set of pre-defined css classes for validation.
- The built-in classes can identify the validation states and apply effects automatically to any specified element.
- The built-in classes doesn't have any pre-defined effects , you have to configure the effects manually .
- The Angular css classes are :

CLASS NAME	DESCRIPTION
.ng-valid	- Applies effect when input state is Valid.
.ng-invalid	- Applies effect when input state is invalid.
.ng-pristine	-Applies effect when input field is not modified.
.ng-dirty	- Applies effect when input field Is modified.
.ng-touched	- Effect when input field is touch-ed.

Example:

```
input.ng-invalid{
    background-color: aquamarine;
}
```

NOTE:
You can apply pre-defined css angular validation classes in form state validation.

```
form.ng-invalid{
    background-color: aquamarine;
}
```

CUSTOM VALIDATION IN ANGULAR

- HTML provides a limited set of validation properties like required, email, url, minlength etc..
- The input value can be verified using the properties provided by HTML.

- However various other validations need to be defined by using custom functions and events.
- You can create a custom validation by accessing the input value of any specific event and verify with the required value.
- The error messages can be displayed by using a custom Boolean property that identifies the validation errors.

  Example:

  1. >ng g c customvalidation –-spec = false
  2. Customvalidation.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-customvalidation',
  templateUrl: './customvalidation.component.html',
  styleUrls: ['./customvalidation.component.css']
})
export class CustomvalidationComponent  {
public showCityError = false;
public showEvenError = true;
public emailError=true;

public errorClass= true;
public validClass=false;

public SelectedCityChanged(cityValue) {
 if(cityValue=='nocity'){
   this.showCityError=true;
   this.errorClass=true;
   this.validClass=false;
 }else{
   this.showCityError=false;
   this.errorClass=false;
   this.validClass=true;
 }
}
public VerifyEvenNumber(n){
if(n%2 == 0){
  this.showEvenError=false;
}else{
  this.showEvenError=true;
}
}
public VerifyEmail(email) {
let atPos = email.indexOf('@');
let dotPos = email.lastIndexOf('.');
// let lastPos = email.charAt(dotPos + 2);

// let mailformat = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;


if (atPos < 2 && (dotPos - atPos) < 2 ) {
  this.emailError = true;
} else  {
  this.emailError = false;
}
}

}
```

3. Customvalidation.component.html

```
<div class="container">
    <form #frmRegister="ngForm" name="frmRegister" novalidate>
        <h2>Register</h2>
        <div class="form-group">
            <label>Select Your City</label>
            <select [ngClass]="{errorStyle:errorClass,validStyle:validC
lass}" (change)="SelectedCityChanged(lstCities.value)" class="form-
control" name="lstCities" #lstCities="ngModel" ngModel>
                <option value="nocity">Select a City</option>
                <option value="Delhi">Delhi</option>
                <option value="Hydrabad">Hydrabad</option>
            </select>
            <span *ngIf="showCityError" class="text-
danger">Please select a city</span>
        </div>
        <div class="form-group">
            <label>Enter a even Number</label>
            <div>
                <input type="text" (blur)="VerifyEvenNumber(txtEven.val
ue)" class="form-control" name="txtEven" #txtEven="ngModel" ngModel>
                <span *ngIf="showEvenError" class="text-
danger">Not an even number</span>
            </div>
        </div>
        <div class="form-group">
            <label>Email</label>
            <input (blur)="VerifyEmail(txtEmail.value)" type="email" cl
ass="form-control" name="txtEmail" #txtEmail="ngModel" ngModel>
            <span class="text-
danger" *ngIf="emailError">Invalid Email</span>
        </div>
    </form>
</div>
```

4. Customvalidation.component.css

```
.errorStyle{
    border:1px solid red;
}
.validStyle{
    border:1px solid green;
}
```

Example: Changing validation pattern dynamically for any Element.

>ng g c dynamicvalidation – - spec= false

1. dynamicvalidation.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-dynamicvalidation',
  templateUrl: './dynamicvalidation.component.html',
  styleUrls: ['./dynamicvalidation.component.css']
})
export class DynamicvalidationComponent {
public pic;
public tip;
public regExp;
public mobileError;

public SelectedCountryChanged(countryName){
  switch (countryName) {
    case 'India':
    this.pic = 'assets/india.jpg';
    this.tip = 'India calling code is +91 and followed by 10 digits';
    this.regExp = /\+91[0-9]{10}/;
    break;
    case 'US':
    this.pic = 'assets/us.jpg';
    this.tip = 'US calling code is +001 and followed by 6 digits';
    this.regExp = /\+001[0-9]{6}/;
    break;
    case 'UK':
    this.pic = 'assets/uk.jpg';
    this.tip = 'UK calling code is +44 and followed by 6 digits';
    this.regExp = /\+44[0-9]{8}/;
    break;
  }
}
public VerifyMobile(mobile) {
if (mobile.match(this.regExp)) {
 this.mobileError = 'Mobile Verified Successfully..';
} else {
  this.mobileError = 'Invalid Mobile number Please check calling code!!';
}
}
}
```

2. Dynamicvalidation.component.html

```
<div class="container">
    <fieldset>
        <legend>
            Verify Your Mobile<img [src]="pic" alt="&#164;" width="30" heig
ht="30">
        </legend>
        <div class="form-group">
            <label>Select Your Country</label>
            <div>
                <select (change)="SelectedCountryChanged(lstCountry.value)"
 name="lstCountry" #lstCountry="ngModel" ngModel class="form-control">
                    <option value="India">India</option>
                    <option value="US">US</option>
                    <option value="UK">UK</option>
                </select>
            </div>
        </div>
        <div class="form-group">
            <label>Mobile Number</label>
            <div>
                <input type="text" name="txtMobile" #txtMobile="ngModel" ng
Model placeholder="{{tip}}" class="form-control">
            </div>
        </div>
        <div class="form-group">
            <button (click)="VerifyMobile(txtMobile.value)" class="btn btn-
primary">VerifyMobile</button>
        </div>
    </fieldset>
    <div>
        <h2><marquee scrollamount="20">{{mobileError}}</marquee></h2>
        <p>{{tip}}</p>
    </div>
</div>
```

REACTIVE FORMS / MODEL DRIVEN FORMS

- Angular supports the frameworks MVC & MVVM where the data is represented and handled by a Model.
- The Template Driven forms are not bound to model. Hence, all interactions are handled at view level.
- The model changes are not updated to view.
- Reactive Forms provide a model driven approach to handle the form-input.

- The forms are bound to the model hence the model changes are updated to view dynamically.
- Reactive forms uses an explicit approach where the form and controls can be create dynamically.
- Reactive forms are built around observables streams hence; they can identify and track the model changes.
- Reactive forms also provide a straightforward path to testing because they are continuously connected to data.
- Reactive forms can be asynchronous i.e. support partial updates so that only specific portion can be updated and submitted using ajax calls.
- The library required for handling Reactive forms is '@angular/core' with module 'ReactiveFormsModule'.
- The form elements are created by implementing the base class 'FormControl' – it provides all properties and methods that are used to handle a control dynamically.
- SYNTAX:
  Public name = new FormControl('value');
- The dynamic formControls are banded with HTML form elements by using the property [formControl].

SYNTAX:

< input type="text" [formControl]="name">

- The values for FormControl can be updated dynamically by using the function

  setValue( ) and patchValue( )

  SYNTAX:

  This.name.setValue('someValue');

  Example:

  1. Go To 'app.module.ts'

```
import { FormsModule, ReactiveFormsModule } from '@angular/for
ms';
```

```
imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule
  ]
```

```
>ng g c reactivedemo – spec= false
```

Reactivedemo.component.ts

```
import { Component, OnInit } from '@angular/core';
import {FormControl} from '@angular/forms';

@Component({
  selector: 'app-reactivedemo',
  templateUrl: './reactivedemo.component.html',
  styleUrls: ['./reactivedemo.component.css']
})
export class ReactivedemoComponent {
public txtName = new FormControl('Chiranjib');
public lstCities = new FormControl('Delhi');

public UpdateName() {
  this.txtName.setValue('Chandrakanta');
}

}
```

<u>Reactivedemo.component.html</u>

```
<div class="container">
    <div class="form-group">
        <label>User Name</label>
        <div>
            <input type="text" [formControl]="txtName" class="form-
control" required ><span *ngIf="txtName.invalid" class="text-
danger">Name required</span>
        </div>
    </div>
    <div class="form-group">
        <label>Your City</label>
        <select [formControl]="lstCities" class="form-control">
            <option>Delhi</option>
            <option>Chennai</option>
        </select>
    </div>
    <div class="form-group">
        <button class="btn btn-
primary" (click)="UpdateName()">Update Name</button>
    </div>
    <div class="form-group">
        Name: {{txtName.value}}
        <br>
        City: {{lstCities.value}}
    </div>
</div>
```

<u>NESTED FORMS</u>

- A form element can handle another form within the context of the parent.

- Dynamically a formGroup can be configured with childGroup by using nested hierarchy.
  SYNTAX:
  Public parentForm = new FormGroup({
    Control : new formControl( ),
   childForm: new formGroup({
     control : new formControl( )
  })
});
- The parent form is configured to the <form> element by using [formGroup] and the childform is defined to any container within the form by using the property ‘’formGroupName’’.
- SYNTAX: <form [formGroup] = ‘’parentForm’’>
  <div formGroupName=’’childform’’>
  </div>
  </form>

  >ng g c nestedform –spec=false

<u>Nestedform.component.ts</u>

```
import { Component, OnInit } from '@angular/core';
import {FormControl, FormGroup} from '@angular/forms';

@Component({
  selector: 'app-nestedform',
  templateUrl: './nestedform.component.html',
  styleUrls: ['./nestedform.component.css']
})
export class NestedformComponent {

public frmRegister = new FormGroup({
  Name: new FormControl(''),
  Price: new FormControl(''),
  frmDetails: new FormGroup({
    City: new FormControl(''),
    IsInStock: new FormControl('')
  })
});
public UpdateDetails() {
  this.frmRegister.patchValue({
    Name: 'Samsung TV',
    Price: 45000,
    frmDetails: {
      City: 'Bbsr',
      IsInStock: true,
    }
    });
  }

  }
```

Nestedform.component.html

```
<div class="container">
    <h2>Register Product</h2>
    <form [formGroup]="frmRegister">
        <fieldset>
            <legend>Basic Details</legend>
            <dl>
                <dt>Name</dt>
                <dd>
                    <input type="text" formControlName="Name" class="form-
control">
                </dd>
                <dt>Price</dt>
                <dd>
                    <input type="text" formControlName="Price" class="form-
control">
                </dd>
            </dl>
        </fieldset>
        <div formGroupName="frmDetails">
            <fieldset style="background-color: lightcyan;">
                <legend>Stock Details</legend>
                <dl>
                    <dt>Shipped To</dt>
                    <dd>
                        <select class="form-control" formControlName="City">
                            <option>Delhi</option>
                            <option>Bbsr</option>
                        </select>
                    </dd>
                    <dt>Is In Stock</dt>
                    <dd>
                        <input type="checkbox" class="form-check-
label" formControlName="IsInStock">Yes
                    </dd>
                </dl>
            </fieldset>
            <br>
            <button (click)="UpdateDetails()" class="btn btn-
primary">Update Details</button>
        </div>
    </form>
    <div style="background-color: aquamarine; color:brown; font-
family: bold; font-size: 20px;">
        <br>
    Name: {{frmRegister.value.Name}}
    <br>
    Price: {{frmRegister.value.Price | currency:'INR'}}
    <br>
    City: {{frmRegister.value.frmDetails.City}}
    <br>
    IsInStock: {{(frmRegister.value.frmDetails.IsInStock==true)?"Available":"O
ut Of stock"}}
    </div>
</div>
```

FORM BUILDER

- Form builder is a service provided by angular that allows to configure a form and its elements by using 'single Ton' mechanism.
- The form builder service injects a set of functions that allows to configure the form they are:
    a. group( )
    b. control( )
    c. array( )
- The 'group()' configures a form-group.
- The 'control()' configures a form-control.
- The 'array()' configures set of controls that are dynamically added into form-group.

Examples:

>ng g c formbuilderdemo –spec=false

Formbuilderdemo.component.ts

```
import { Component, OnInit } from '@angular/core';
import {FormBuilder, Validators, FormArray} from '@angular/forms';

@Component({
  selector: 'app-formbuilderdemo',
  templateUrl: './formbuilderdemo.component.html',
  styleUrls: ['./formbuilderdemo.component.css']
})
export class FormbuilderdemoComponent implements OnInit {

  constructor(private fb: FormBuilder) { }
  public frmRegister = this.fb.group({
    Name: [''],
    Price: [''],
    frmDetails: this.fb.group({
      City: [''],
      IsInStock: ['']
    })

  });

  ngOnInit() {
  }
}
```

Formbuilderdemo.component.html

```
<div class="container">
    <h2>Register Product</h2>
    <form [formGroup]="frmRegister">
        <fieldset>
            <legend>Basic Details</legend>
            <dl>
                <dt>Name</dt>
                <dd>
                    <input type="text" formControlName="Name" class="form-
control" >
                </dd>
                <dt>Price</dt>
                <dd>
                    <input type="text" formControlName="Price" class="form-
control">
                </dd>
            </dl>
        </fieldset>
        <div formGroupName="frmDetails">
                <h2>Stock Details</h2>
                <dl>
                    <dt>City</dt>
                    <dd>
                        <select class="form-control" formControlName="City">
                            <option>Delhi</option>
                            <option>Bbsr</option>
                        </select>
                    </dd>
                    <dt>Is In Stock</dt>
                    <dd>
                        <input type="checkbox" class="form-check-
label" formControlName="IsInStock">Yes
                    </dd>
                </dl>
        </div>
```

FORM VALIDATORS

- Angular provides a validator service that handles all validations for controls dynamically.
- It provides a set of validation attributes configured in validators collection of '@angular/forms' library.
- The validators include all html file validations like required, email, url, maxlength, pattern etc..
- The validators are dynamically configured for controls which you can verify by using a status attribute of <form> element.
- SYNTAX:

  Go to .ts file of previous example

```
import {FormBuilder, Validators } from '@angular/forms';

public frmRegister = this.fb.group({
    Name: ['', Validators.required],
    Price: [''],
```

Go to .ts file of previous example

```
         <dd>
            <input type="text" formControlName="Name" class="form-
control" required>
{{frmRegister.status}}
        </dd>
```

NOTE:

You can configure an alias name for any specific property in Typescript so that it can directly accessed by using alias name. It requires the "get()" accessor.

```
SYNTAX:  get refName( ) {
              return this.yourPropertyHeirarchy as Type
                 }
```

```
  newControls: this.fb.array([
       this.fb.control('')
    ])
  });

  ngOnInit() {
  }
get newControls() {
return this.frmRegister.get('newControls') as FormArray;
}
```

FORM ARRAY

- Angular provides a form array service that allows to add controls dynamically.
- It uses a form-builder function Array( ), which creates a dynamic form Array with set of controls.
- You can dynamically add/remove controls by using the array functions push(), splice(), pop()….
    - A form Array allows to add elements dynamically using ''push()'' and remove by using ''removeAt()''
    - SYNTAX:

      this.formArray.push(this.fb.control(' '));
      this.formArray.removeAt(index);

Formbuilderdemo.component.ts

```
import { Component, OnInit } from '@angular/core';
import {FormBuilder, Validators, FormArray} from '@angular/forms';

@Component({
  selector: 'app-formbuilderdemo',
  templateUrl: './formbuilderdemo.component.html',
  styleUrls: ['./formbuilderdemo.component.css']
})
export class FormbuilderdemoComponent implements OnInit {

  constructor(private fb: FormBuilder) { }
  public frmRegister = this.fb.group({
    Name: ['', Validators.required],
    Price: [''],
    frmDetails: this.fb.group({
      City: [''],
      IsInStock: ['']
    }),
    newControls: this.fb.array([
      this.fb.control('')
    ])
  });

  ngOnInit() {
  }
get newControls() {
return this.frmRegister.get('newControls') as FormArray;
}
public AddPhoto() {
  this.newControls.push(this.fb.control(''));
}
public RemovePhoto(i) {
this.newControls.removeAt(i);
}
}
```

Formbuilderdemo.component.html

```
<div class="container">
    <h2>Register Product</h2>
    <form [formGroup]="frmRegister">
        <fieldset>
            <legend>Basic Details</legend>
            <dl>
                <dt>Name</dt>
                <dd>
                    <input type="text" formControlName="Name" class="form-
control" required>{{frmRegister.status}}
                </dd>
                <dt>Price</dt>
                <dd>
                    <input type="text" formControlName="Price" class="form-
control">
                </dd>
            </dl>
        </fieldset>
        <div formGroupName="frmDetails">
                <h2>Stock Details</h2>
                <div class="form-group">
                    <button (click)="AddPhoto()">Upload More Photos</button>
                </div>
                <div *ngFor="let c of newControls.controls; let i=index">
                    <label>photo</label>
                <div>
                    <input type="file" formControlName="i">
                    <button (click)="RemovePhoto(i)" class="btn btn-
link">Remove</button>
                </div>
                </div>
                <dl>
                    <dt>City</dt>
                    <dd>
                        <select class="form-control" formControlName="City">
                            <option>Delhi</option>
                            <option>Bbsr</option>
                        </select>
                    </dd>
                    <dt>Is In Stock</dt>
                    <dd>
                        <input type="checkbox" class="form-check-
label" formControlName="IsInStock">Yes
                    </dd>
                </dl>
        </div>
```

Example.2

HTML File

```
<div class="container">
    <h2>Register Product</h2>
    <form [formGroup]="frmRegister">
        <label>Product Name</label>
        <input type="text" class="form-
control" formControlName="Name" >{{frmRegister.status}}<br>
        <label>Product Price</label>
        <input type="number" class="form-control" formControlName="Price">
        <h3>Stock Details</h3>
        <div formGroupName="frmDetails">
            <label>Shipped To</label>
            <select class="form-control" formControlName="City">
                <option>Bbsr</option>
                <option>Hyd</option>
                <option>Banglore</option>
            </select>
            <label>IsInStock</label>
            <input type="checkbox"  formControlName="IsInStock">
        </div>
        <div class="form-group">
            <button class="btn btn-
success" (click)="AddPhotos()">Upload More photos</button>
        </div>
        <div *ngFor="let c of newControls.controls; let i = index">
            <label>Photos[{{i+1}}]</label>
            <div>
                <input type="file" class="btn btn-dark" formControlName="i">
                <button class="btn btn-
danger" (click)="RemovePhoto(i)">Remove</button>
            </div>
        </div>
```

```html
    </form>
    <div class="colorbg">
        <dl>
            <dt>Name</dt>
            <dd>{{frmRegister.value.Name}}</dd>
            <dt>Price</dt>
            <dd>{{frmRegister.value.Price | currency:'&#8377;'}}</dd>
            <dt>Shipped To</dt>
            <dd>{{frmRegister.value.frmDetails.City}}</dd>
            <dt>IsInStock</dt>
            <dd>{{(frmRegister.value.frmDetails.IsInStock === true)?'Available
':'Out Of Stock'}}</dd>
        </dl>

    </div>
</div>
```

TypeScript File

```typescript
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormBuilder, Validators, FormArray } from '@angular/forms'
;

@Component({
  selector: 'app-formbuilder',
  templateUrl: './formbuilder.component.html',
  styleUrls: ['./formbuilder.component.css']
})
export class FormbuilderComponent implements OnInit {

  constructor(private fb: FormBuilder) { }
  public frmRegister = this.fb.group({
    Name: ['', Validators.required],
    Price: [''],
    frmDetails: this.fb.group({
      City: [''],
      IsInStock: ['']
    }),
    newControls: this.fb.array([
      this.fb.control('')
    ])
  });

  ngOnInit() {
  }
   get newControls() {
     return this.frmRegister.get('newControls') as FormArray;
   }
   public AddPhotos() {
    this.newControls.push(this.fb.control(''));
   }
   public RemovePhoto(index) {
     this.newControls.removeAt(index);
   }
}
```

Thank You

Swainchiranjib6@gmail.com

Whatsapp No- 9853004369