

## 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. templateform.component.css

```

.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 contradictory 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	<ul style="list-style-type: none"> <li>- It verifies whether any field in the form have modified its value with user input.</li> <li>- It returns true when no field modified.</li> </ul>
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. formstatevalidation.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 type="text" name="txtName" ngModel #txtName="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	<ul style="list-style-type: none"> <li>- It returns true when the input fields never focused.</li> </ul>
ngPristine	Pristine	<ul style="list-style-type: none"> <li>- It verifies whether any field in the form have modified its value with user input.</li> <li>- It returns true when no field modified.</li> </ul>
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,invalidStyle:txtName.invalid}" type="text" name="txtName" ngModel #txtName="ngModel" class="form-control" required>
          <span *ngIf="txtName.invalid && frmRegister.submitted" class="text-danger">Name Required</span>
        </div>
      </div>
    </form>
  </div>
</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.
- SYNTAX:

```
<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 :

<u>CLASS NAME</u>	<u>DESCRIPTION</u>
.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(1stCities.value)" class="form-
control" name="1stCities" #1stCities="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" height="30">  
    </legend>  
    <div class="form-group">  
      <label>Select Your Country</label>  
      <div>  
        <select (change)="SelectedCountryChanged(1stCountry.value)"  
          name="1stCountry" #1stCountry="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" ngModel 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>
```