Angular

Introduction:

Angular is a platform and framework for building single-page client applications using HTML and Typescript. Angular is written in Typescript. It implements core and optional functionality as a set of Typescript libraries that you import into your applications.

Commands: - all input given in Terminal

1. Installing Angular CLI
   npm install -g @angular/cli
2. To create, build, and serve a new, basic Angular project on a development server

   ng new my-first-project – create new project

   Name of project

   cd my-first-project - get into project directory

   ng serve - Execute project

3. Create new component
   ng g c new-component

   Name of component

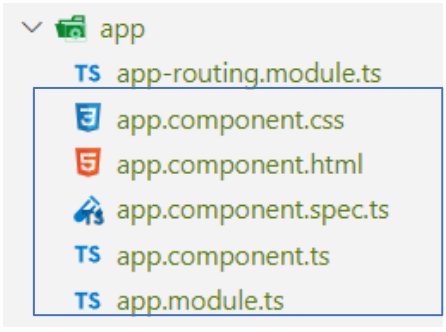
Component properties

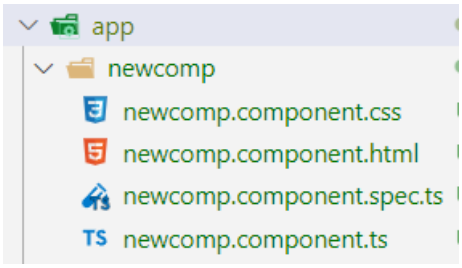
Default component for each project

(app-component)

User define Component

(new component)





Angular

1. Data Binding

What is Angular data binding?

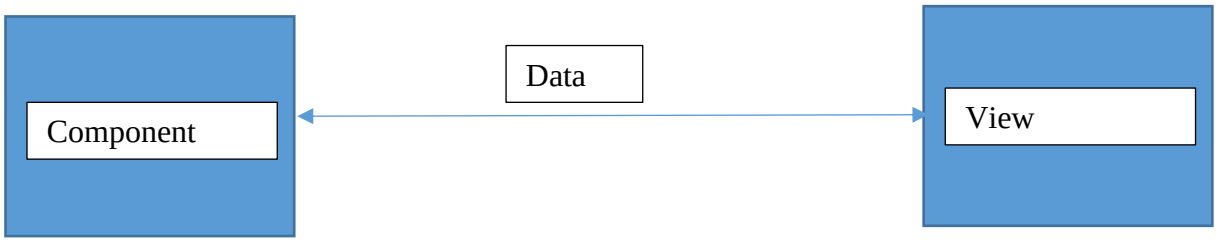
Angular data binding is a two-way process: it can both send and receive data. This means that when you change something in your view, the model updates automatically, and vice versa. The ngModel directive makes this two-way data binding possible.

Which are 2 types of data binding?

1) **One-way binding** is a relatively simple type of data binding. ...
2) **Two-way binding** is where changes to either the data provider or the data consumer automatically updates the other.

One – Way Data binding

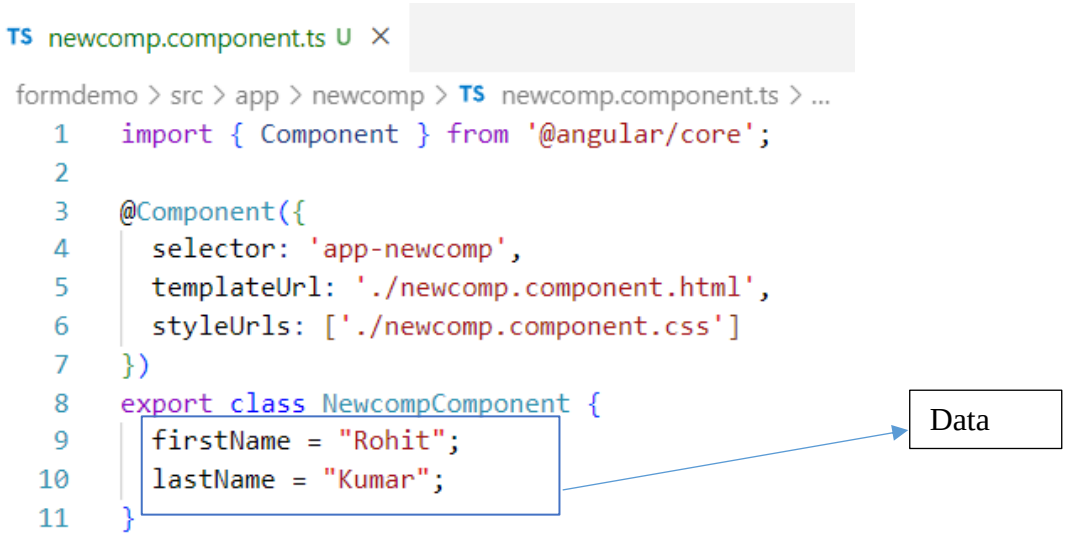
One-way data binding will bind the data from the component to the view (DOM) or from view to the component. One-way data binding is unidirectional. You can only bind the data from component to the view or from view to the component.



Component to View

Step: 1

In user defined component **newcomp.component.ts** creating data in class of NewcompComponent



Step: 2

Creating html element to display the data in **newcomp.component.html**

{{}} – string Interpolation

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > ...

Go to component

```
1  <h1>{{firstName}} {{lastName}}</h1>
2
```

Step: 3

Copy the selector of newcomp.component.ts and paste in app.component.html

newcomp.component.ts U

formdemo > src > app > newcomp > newcomp.component.ts > ...

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-newcomp',
5    templateUrl: './newcomp.component.html',
6    styleUrls: ['./newcomp.component.css']
7  })
8  export class NewcompComponent {
9    firstName = "Rohit";
10   lastName = "Kumar";
11 }
```

app.component.html M

formdemo > src > app > app.component.html > app-newcomp

```
1  <app-newcomp></app-newcomp>
```

Step: 4

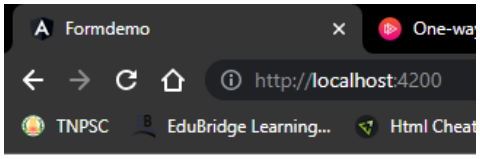
Execute the code by using terminal comment **ng server**

Make sure you're in the same project folder

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\Edubridge\Anguler\formdemo> ng serve
✓ Browser application bundle generation complete.
```

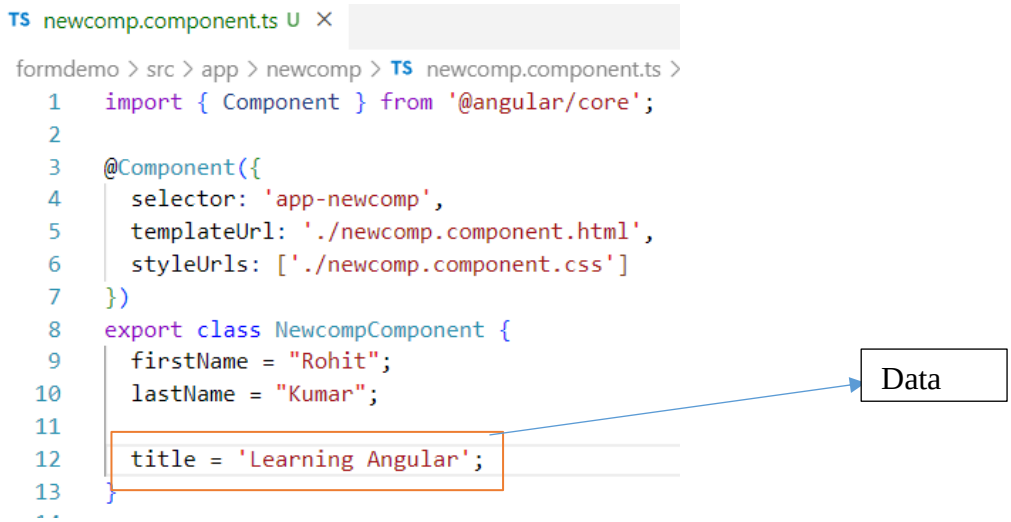
Output



Rohit Kumar

Property Binding

Another way of binding the data



Note:

In both the case of string interpolation and property binding you cannot reassign the value in html

View to Component

It is done with the help of using **Event Binding.**

2. Two-Way Binding

It can be done with the help of **ngModel** directive

Step: 1

Open **app.module.ts**

Import FormsModule.

Make available FormsModule in imports section

Now you can use ng module directive

app.module.ts M

formdemo > src > app > app.module.ts > ...

```
4   import { AppRoutingModule } from './app-routing.module';
5   import { AppComponent } from './app.component';
6   import { NewcompComponent } from './newcomp/newcomp.component';
7   import { FormsModule } from '@angular/forms';
8   @NgModule({
9     declarations: [
10      AppComponent,
11      NewcompComponent
12    ],
13    imports: [
14      BrowserModule,
15      AppRoutingModule,
16      FormsModule
17    ],
18    providers: [],
19    bootstrap: [AppComponent]
20  })
21  export class AppModule { }
```

Step: 2

Create data in newcomp.component.ts in class of NewcompComponent

newcomp.component.ts U

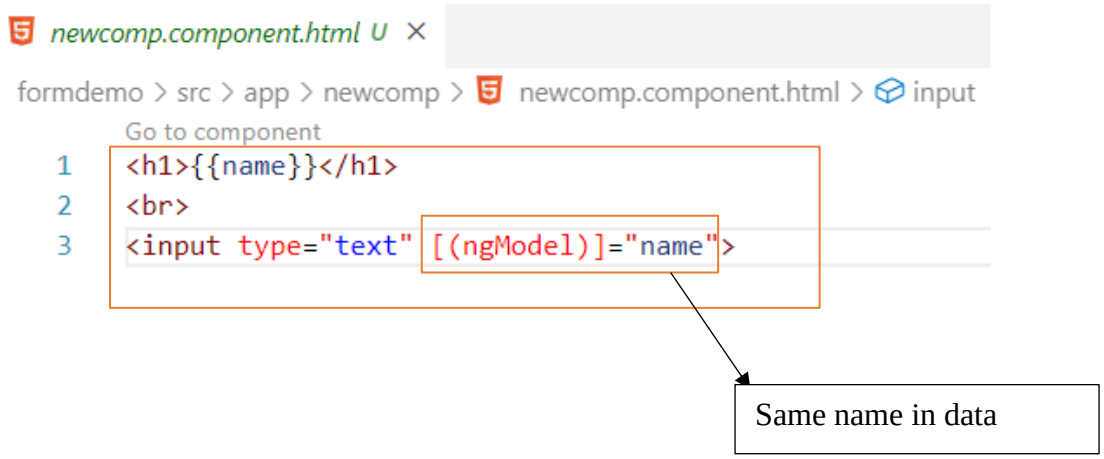
formdemo > src > app > newcomp > newcomp.component.ts > ...

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-newcomp',
5     templateUrl: './newcomp.component.html',
6     styleUrls: ['./newcomp.component.css']
7   })
8   export class NewcompComponent {
9     name ="Praveen";
10  }
```

Step: 3

Create html element in newcomp.component.html

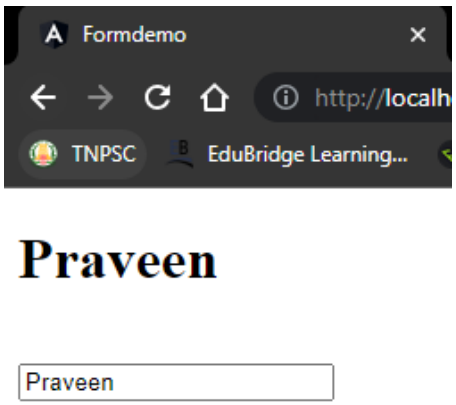
In addition with ngModule directive in input tag.

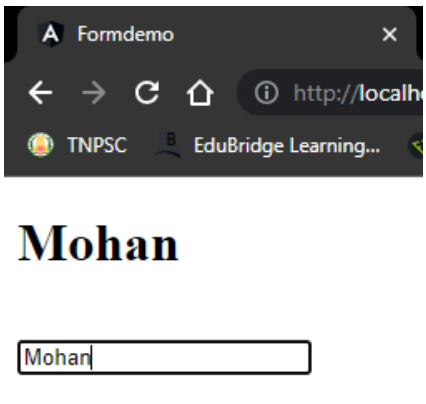


Output

The value entered in the text box rewrites its original value

Data view → component and component → view





3. Interpolation

Interpolation refers to embedding **expressions** into marked up text. By default, interpolation uses the double curly braces {{ and }} as delimiters.

Angular

1. String expression

TS newcomp.component.ts U

formdemo > src > app > newcomp > TS newcomp.component.ts >

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

@Component({
  selector: 'app-newcomp',
  templateUrl: './newcomp.component.html',
  styleUrls: ['./newcomp.component.css']
})
export class NewcompComponent {
  title ="Angular";
}
```

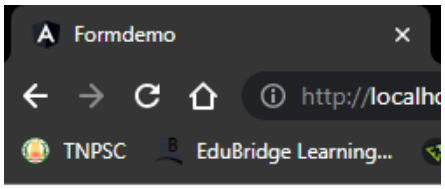
newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > ...

Go to component

```html
<h1>{{title}}</h1>

<h2>{{'Hello '+'World '+'to '+'Angular'}}</h2>

```

Output



Angular

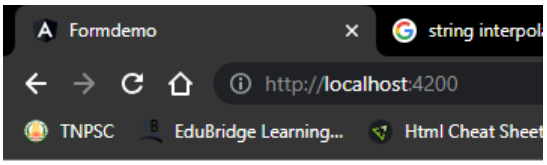
Hello World to Angular

2. Function expression

```
TS newcomp.component.ts U ×
formdemo > src > app > newcomp > TS newcomp.component.ts > ...
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-newcomp',
5     templateUrl: './newcomp.component.html',
6     styleUrls: ['./newcomp.component.css']
7   })
8   export class NewcompComponent {
9     title ="Angular title by function";
10
11    getTitle(){
12      return this.title
13    }
14  }
```

```
newcomp.component.html U ×
formdemo > src > app > newcomp > newcc
Go to component
1   <h1>{{getTitle()}}</h1>
2
```

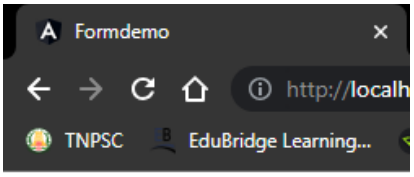
Output



Angular title by function

3. Arithmetic Expression

```
newcomp.component.html U ×
formdemo > src > app > newcomp > newcomp
Go to component
1   <p>100 x 80 = {{100*80}}</p>
2
```



100 x 80 = 8000

Through Functions and arguments

newcomp.component.ts U

formdemo > src > app > newcomp > TS newcomp.component.ts > .

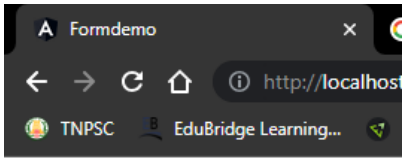
```ts
import { Component } from '@angular/core';

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

  getMax(a:number,b:number){
    return Math.max(a,b);
  }
}
```

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html

Go to component

```html
<h2>Largest Number : {{getMax(78,98)}}</h2>

```



Largest Number : 98

Defining data type to number

4. Input to Property of element through Interpolation

newcomp.component.ts U

formdemo > src > app > newcomp > TS newcomp.component.ts

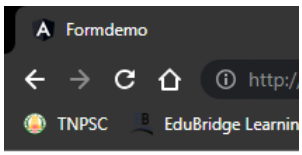
```ts
import { Component } from '@angular/core';

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

  color = 'red';
}
```

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > .

Go to component

```html
<h2 style.color="{{color}}">This is Red.</h2>
```



This is Red.

4. Property Binding

Value assigning html element properties is called property binding
Eg:
<h1 [innerText]=”title”></h1>

newcomp.component.ts U

formdemo > src > app > newcomp > newcomp.component.ts

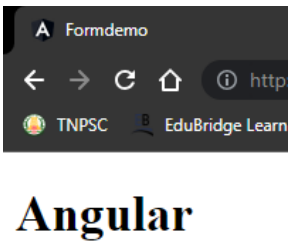
```ts
import { Component } from '@angular/core';

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

  title="Angular";
}
```

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.

Go to component

```html
<h1 [innerText]="title"></h1>
```



Button disabled- Attribute

newcomp.component.ts U

formdemo > src > app > newcomp > newcomp.component.ts

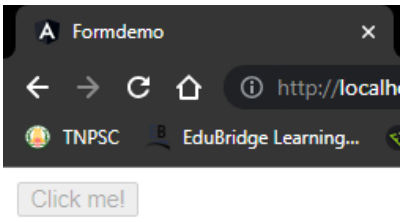
```ts
import { Component } from '@angular/core';

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

  isDisabled=true;
}
```

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > ...

Go to component

```html
<button [disabled]="isDisabled" >Click me!</button>
```



Or

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > ...

Go to component

```html
<button [attr.disabled]="isDisabled" >Click me!</button>

```

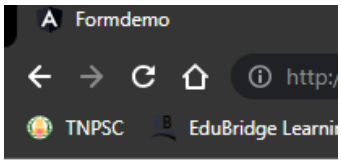
String in Property binding

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > ...

Go to component

```
1   <p [innerText]="'Welcome '+'to '+'Angular'"></p>
```



Welcome to Angular

5. Event Binding

Event binding lets you listen for and respond to user actions such as keystrokes, mouse movements, clicks, and touches.

// Full list of Angular Events

1. (click)="myFunction()"
2. (dblclick)="myFunction()"
3. (submit)="myFunction()"
4. (blur)="myFunction()"
5. (focus)="myFunction()"
6. (scroll)="myFunction()"
7. (cut)="myFunction()"
8. (copy)="myFunction()"
9. (paste)="myFunction()"
10. (keyup)="myFunction()"
11. (keypress)="myFunction()"
12. (keydown)="myFunction()"
13. (mouseup)="myFunction()"
14. (mousedown)="myFunction()"
15. (mouseenter)="myFunction()"
16. (drag)="myFunction()"
17. (drop)="myFunction()"
18. (dragover)="myFunction()"

TS newcomp.component.ts U

formdemo > src > app > newcomp > TS newcomp.component.ts

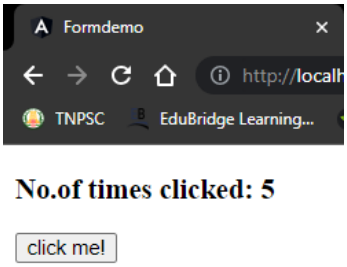
```ts
import { Component } from '@angular/core';

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

  clickCount = 0;

  clickMe(){
    return this.clickCount++;
  }
}
```

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html

Go to component

```html
<h3>No.of times clicked: {{clickCount}}</h3>

<button (click)="clickMe()">click me!</button>
```



Or - give attribute **on-click**

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > ...

Go to component

```html
<h3>No.of times clicked: {{clickCount}}</h3>

<button on-click="clickMe()">click me!</button>
```

Click also used to call variable

TS newcomp.component.ts U

formdemo > src > app > newcomp > TS newcomp.component.ts

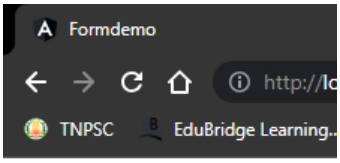
```ts
import { Component } from '@angular/core';

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

  clickCount = 0;

  clickCount1 =0;

  clickMe(){
    return this.clickCount++;
  }
}
```

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > ...

Go to component

```
1  <h3>No.of times clicked: {{clickCount}}</h3>
2  <h3>copy of Click count :{{clickCount1}} </h3>
3
4  <button (click)="clickMe();clickCount1=clickCount">click me!</button>
```



No.of times clicked: 6

copy of Click count :6

click me!

Carrying input data through event

By using $event property – carries the event payload.

newcomp.component.ts U

formdemo > src > app > newcomp > newcomp.component.ts > ...

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-newcomp',
5    templateUrl: './newcomp.component.html',
6    styleUrls: ['./newcomp.component.css']
7  })
8  export class NewcompComponent {
9
10   value='';
11
12   handleInput(event:any){
13     this.value = (event.target as HTMLInputElement).value;
14   }
15 }
```

Type as assertion

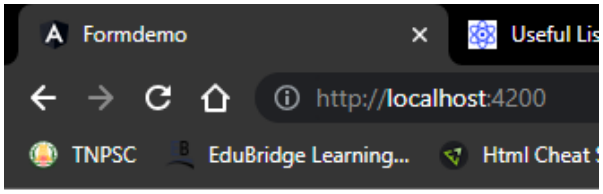
newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > ...

Go to component

```
1  Type Something :<input type="text" (input)="handleInput($event)">
2
3  <p>{{value}}</p>
4
```

Event



Same output without handleInput Function

newcomp.component.ts U

formdemo > src > app > newcomp > newcomp.component.ts > ...

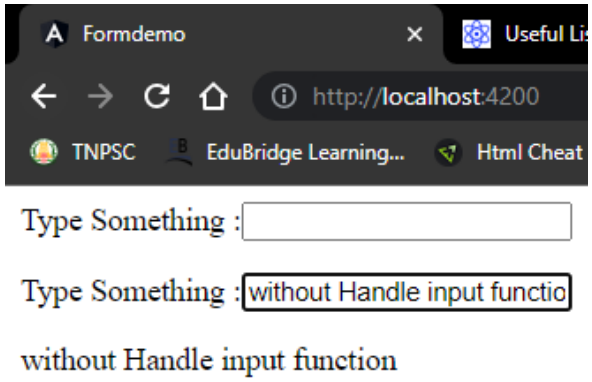
```ts
import { Component } from '@angular/core';

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

  value='';
  value1 = '';

  handleInput(event:any){
    this.value = (event.target as HTMLInputElement).value;
  }
}
```

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > ...

Go to component

```html
Type Something :<input type="text" (input)="handleInput($event)">

<p>{{value}}</p>

Type Something :<input type="text" (input)="value1= $any($event.target).value">
<p>{{value1}}</p>
```

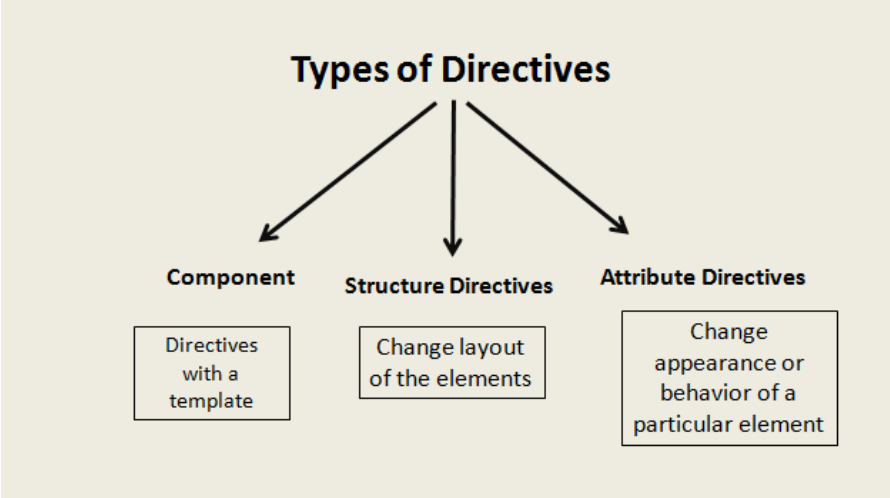


Various types of Events

https://developer.mozilla.org/en-US/docs/Web/Events

6. Directives

Directives are defined as classes that can **add new behavior to the elements** in the template or **modify existing behavior.** The purpose of Directives in Angular is to maneuver the DOM, be it by adding new elements to DOM or removing elements and even **changing the appearance** of the DOM elements.



Component Directives:

Special directives in Angular are called Components since this type of directive has a template or template URLs. In effect, it is a component directive that shows something in DOM.

Structural Directive

This type of directive is used to make changes in the layout of the DOM. Elements can be added or removed, hence changing the structure of the DOM. An example would be *ngIf(adding or removing element from DOM) or ***ngFor**(lists elements of every iteration).

1. ngFor directives

The ngFor is an Angular structural directive, which repeats a portion of the HTML template once per each item from an iterable list (Collection). The ngFor is similar to ngRepeat in AngularJS

TS newcomp.component.ts U

formdemo > src > app > newcomp > TS newcomp.component.ts > NewcompComponent

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

@Component({
  selector: 'app-newcomp',
  templateUrl: './newcomp.component.html',
  styleUrls: ['./newcomp.component.css']
})
export class NewcompComponent {
  title="List of Movies";
  title2 = "Fruits Details";

  movies = ['Batman','Avengers','Avatar','Spiderman','Kushi'];

  fruits = [
    {fname:"Apple",vit:"vitamin-A",price:120},
    {fname:"Orange",vit:"vitamin-C",price:90},
    {fname:"Banana",vit:"vitamin-B",price:70}
  ];

}
```

Array

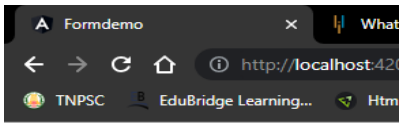
Object

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > h2

Go to component

```html
<h2>{{title}}</h2>

<ul>
    <li *ngFor="let mov of movies">{{mov}}</li>
</ul>

<h2>{{title2}}</h2>

<ol>
    <li *ngFor="let fr of fruits">{{fr.fname}} - {{fr.vit}} - {{fr.price}}</li>
</ol>
```



List of Movies

- Batman
- Avengers
- Avatar
- Spiderman
- Kushi

Fruits Details

1. Apple - vitamin-A - 120
2. Orange - vitamin-C - 90
3. Banana - vitamin-B - 70

Object output in table

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > div.panel.panel-primary

Go to component

```
<div class="panel panel-primary">
    <div class="panel-heading">
       <h2>{{title2}}</h2>
    </div>
    <div class="panel-body">
    <div class="tabel-responsive">
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>S.No</th>
                    <th>Fruits name</th>
                    <th>Vitamin</th>
                    <th>Price</th>
                </tr>
            </thead>
            <tbody>
                <tr *ngFor="let fru of fruits; let i=index, let o=odd, let e=even">
                    <td>{{i +1}}</td>
                    <td>{{fru.fname}}</td>
                    <td>{{fru.vit}}</td>
                    <td>{{fru.price}}</td>
                </tr>

            </tbody>
        </table>
    </div>
    </div>
</div>
```

Fruits Details

S.No	Fruits name	Vitamin	Price
1	Apple	vitamin-A	120
2	Orange	vitamin-C	90
3	Banana	vitamin-B	70

2. ngSwitch

The ng-switch directive lets you hide/show HTML elements depending on an expression.

Child elements with the ng-switch-when directive will be displayed if it gets a match, otherwise the element, and its children will be removed.

You can also define a default section, by using the ng-switch-default directive, to show a section if none of the other sections get a match.

TS newcomp.component.ts U ×

formdemo > src > app > newcomp > TS newcomp.component.ts > ...

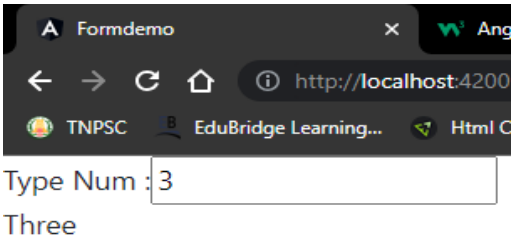
```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-newcomp',
  templateUrl: './newcomp.component.html',
  styleUrls: ['./newcomp.component.css']
})
export class NewcompComponent {
  num=0;

}
```

newcomp.component.html U ×

formdemo > src > app > newcomp > newcomp.component.htm

Go to component

```html
<label for="">Type Num : </label>
<input type="text" [(ngModel)]="num">
<div [ngSwitch]="num">
    <div *ngSwitchCase="1">One</div>
    <div *ngSwitchCase="2">Two</div>
    <div *ngSwitchCase="3">Three</div>
    <div *ngSwitchCase="4">Four</div>
    <div *ngSwitchCase="5">Five</div>
    <div *ngSwitchCase="6">Six</div>
</div>
```



3. ngIf

The ng-if directive removes the HTML element if the expression evaluates to false.

If the if statement evaluates to true, a copy of the Element is added in the DOM.

The ng-if directive is different from the ng-hide, which hides the display of the element, where the ng-if directive completely removes the element from the DOM.

TS newcomp.component.ts U

formdemo > src > app > newcomp > TS newcomp.component.ts > ...

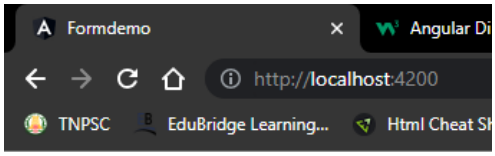
```ts
import { Component } from '@angular/core';

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

  showMe=true;

}
```

newcomp.component.html U

formdemo > src > app > newcomp > newcomp.component.html > h2

Go to component

```html
<h1>ngIf Directives</h1>
<h2>Content Show and hide</h2>

<h3>Example:1 true or false</h3>
<p *ngIf="true">Can you see me? </p>
<p *ngIf="false">Can you see me? </p>

<h3>Example:2  </h3>
<div class="row" >
    <label for="checkbox" >Checkbox <input type="checkbox" [(ngModel)]="showMe"></label>
    {{showMe}}
</div>
<p *ngIf="showMe">Showme is Checked </p>
<p *ngIf="!showMe">Showme is Unchecked </p>
```



ngIf Directives

Content Show and hide

Example:1 true or false

Can you see me?

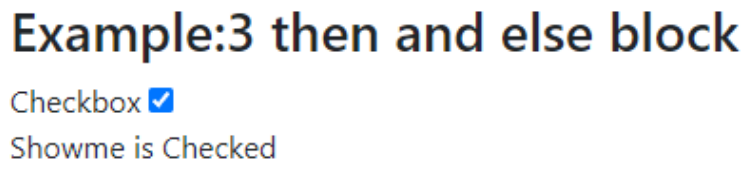
Example:2

Checkbox ☑

true

Showme is Checked

```
<h3>Example:3 then and else block  </h3>
<div class="row" >
    <label for="checkbox" >Checkbox <input type="checkbox" [(ngModel)]="showMe"></label>
</div>
<div *ngIf="showMe; then thenBlock; else elseBlock"></div>
<ng-template #thenBlock>
    <p>Showme is Checked</p>
</ng-template>

<ng-template #elseBlock>
    <p>Showme is Unchecked</p>
</ng-template>
```



3. Attribute Directives

The attribute directive changes the appearance or behavior of a DOM element.

1. ngClass

The ngmodel directive binds the value of HTML controls (input, select, textarea) to application data. With the ng-model directive you can bind the value of an input field to a variable created in Angular. The binding goes both ways. If the user changes the value inside the input field, the Angular property will also change its value.

The ngmodel directive is not part of the Angular Core library. It is part of the **FormsModule** library. You need to import the FormsModule package into your Angular module.

attri-drict.component.css U

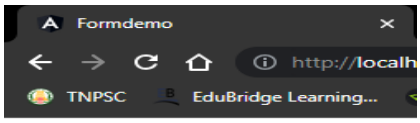
formdemo > src > app > attri-drict > attri-drict.con

```
1   .primary{
2       color: red;;
3   }
4   .secondary{
5       color: green;
6   }
7   .big{
8       font-size:20px;
9   }
10  .small{
11      font-size: 12px;
12  }
13  .italic{
14      font-style: italic;
15  }
16  .bold{
17      font-weight: 900px;
18  }
```

attri-drict.component.html U

formdemo > src > app > attri-drict > attri-drict.component.html > div

```
Go to component
1   <div [ngClass]="['primary','big','bold']"> <!--Array type-->
2       sample text
3   </div>
```



sample text

TS attri-drict.component.ts U

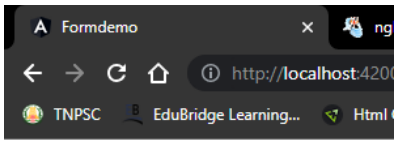
formdemo > src > app > attri-drict > TS attri-drict.component.ts > ...

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-attri-drict',
5     templateUrl: './attri-drict.component.html',
6     styleUrls: ['./attri-drict.component.css']
7   })
8   export class AttriDrictComponent {
9     enableRed=false;
10    enableBig=false;
11    enableItalic= false;
12
13  }
```

attri-drict.component.html U

formdemo > src > app > attri-drict > attri-drict.component.html > div

```
Go to component
1   Red<input type="checkbox" [(ngModel)]="enableRed">
2   Big<input type="checkbox" [(ngModel)]="enableBig">
3   Italic<input type="checkbox" [(ngModel)]="enableItalic">
4
5   <div [ngClass]="{'primary':enableRed,'big':enableBig,'italic':enableItalic}"> <!--Object Type-->
6       Sample Text 2
7   </div>
```



Red☑ Big☐ Italic☑

Sample Text 2

2. ngStyle

Adds and removes a set of HTML styles.

Example:1

ng-style.component.ts U

formdemo > src > app > ng-style > ng-style.component.ts > NgStyleComponent > status

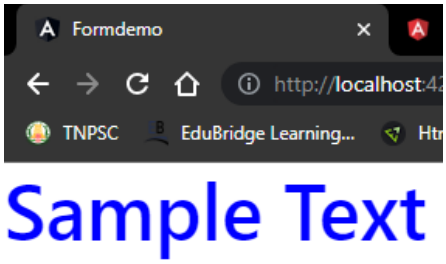
```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-ng-style',
  templateUrl: './ng-style.component.html',
  styleUrls: ['./ng-style.component.css']
})
export class NgStyleComponent {
  status = 'success';
}
```

ng-style.component.html U

formdemo > src > app > ng-style > ng-style.component.html > ...

Go to component

```html
<div [ngStyle]="{'color':status=='error'?'red':'blue'}"><!--Ternary operator-->
 <h3>Sample Text</h3>
</div>
```



Example: 2

ng-style.component.ts U

formdemo > src > app > ng-style > ng-style.component.ts > ...

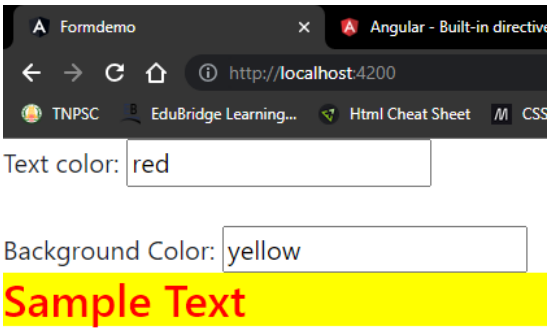
```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-ng-style',
  templateUrl: './ng-style.component.html',
  styleUrls: ['./ng-style.component.css']
})
export class NgStyleComponent {
  color='';
  backgroundColor='';
}
```

ng-style.component.html U

formdemo > src > app > ng-style > ng-style.component.html > ...

Go to component

```
1  Text color: <input type="text" [(ngModel)]="color"> <br> <br>
2  Background Color: <input type="text" [(ngModel)]="backgroundColor">
3  <div [ngStyle]="{'color':color, 'backgroundColor':backgroundColor}">
4    <h3>Sample Text</h3>
5  </div>
```



7. Pipes

Use pipes to transform strings, currency amounts, dates, and other data for display. Pipes are simple functions to use in template expressions to accept an input value and return a transformed value.

https://angular.io/guide/pipes

1) Date Pipes

Pre-defined format options

Option	Equivalent to	Examples (given in en-US locale)
'short'	'M/d/yy, h:mm a'	6/15/15, 9:03 AM
'medium'	'MMM d, y, h:mm:ss a'	Jun 15, 2015, 9:03:01 AM
'long'	'MMMM d, y, h:mm:ss a z'	June 15, 2015 at 9:03:01 AM GMT+1
'full'	'EEEE, MMMM d, y, h:mm:ss a zzzz'	Monday, June 15, 2015 at 9:03:01 AM GMT+01:00
'shortDate'	'M/d/yy'	6/15/15
'mediumDate'	'MMM d, y'	Jun 15, 2015
'longDate'	'MMMM d, y'	June 15, 2015
'fullDate'	'EEEE, MMMM d, y'	Monday, June 15, 2015
'shortTime'	'h:mm a'	9:03 AM
'mediumTime'	'h:mm:ss a'	9:03:01 AM
'longTime'	'h:mm:ss a z'	9:03:01 AM GMT+1
'fullTime'	'h:mm:ss a zzzz'	9:03:01 AM GMT+01:00

pipes.component.ts

pipe > src > app > pipes > pipes.component.ts > PipesComponent

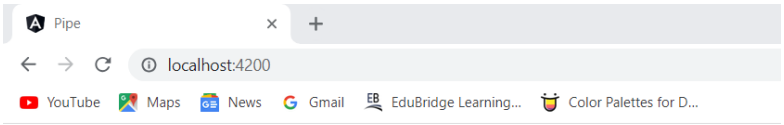
```ts
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-pipes',
  templateUrl: './pipes.component.html',
  styleUrls: ['./pipes.component.css']
})
export class PipesComponent implements OnInit {
  toDate=new Date()


  constructor() {}
  ngOnInit(): void {

  }

}
```

pipes.component.html

pipe > src > app > pipes > pipes.component.html > ...

Go to component

```html
<p>Unformated Date : {{toDate}}</p>

<p>Formated Date : {{toDate|date}}</p>

<p>Medium Argument : {{toDate|date:'medium'}}</p>

<p>Short Argument : {{toDate|date:'short'}}</p>

<p>fulldate Argument : {{toDate|date:'fullDate'}}</p>

<p>longdate Argument : {{toDate|date:'longDate'}}</p>
```



Unformated Date : Mon Jun 12 2023 06:57:54 GMT+0530 (India Standard Time)

Formated Date : Jun 12, 2023

Medium Argument : Jun 12, 2023, 6:57:54 AM

Short Argument : 6/12/23, 6:57 AM

fulldate Argument : Monday, June 12, 2023

longdate Argument : June 12, 2023

2) Uppercase, lowercase and Slice

pipes.component.ts

pipe > src > app > pipes > pipes.component.ts > PipesComponent

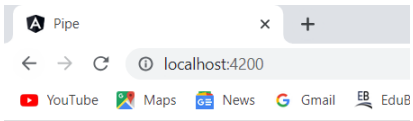
```ts
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-pipes',
  templateUrl: './pipes.component.html',
  styleUrls: ['./pipes.component.css']
})
export class PipesComponent implements OnInit {
  msg="Welcome to Angular"


  constructor() {}
  ngOnInit(): void {

  }

}
```

pipes.component.html

pipe > src > app > pipes > pipes.component.html > ...

Go to component

```html
<!--Upper and lower case -->
<p>Original : {{msg}}</p>
<p>Upper case : {{msg|uppercase}}</p>
<p>Lower case : {{msg|lowercase}}</p>

<!--Slice -->
<p>Original : {{msg}}</p>
<p>slice Eg:1 : {{msg|slice:11:20}}</p>
<p>slice Eg:2 : {{msg|slice:-7}}</p>
<p>slice Eg:3 : {{msg|slice:0:7}}</p>
```



Original : Welcome to Angular

Upper case : WELCOME TO ANGULAR

Lower case : welcome to angular

Original : Welcome to Angular

slice Eg:1 : Angular

slice Eg:2 : Angular

slice Eg:3 : Welcome

3) Currency Pipe

pipes.component.ts

pipe > src > app > pipes > pipes.component.ts > PipesComponent > ngOnInit

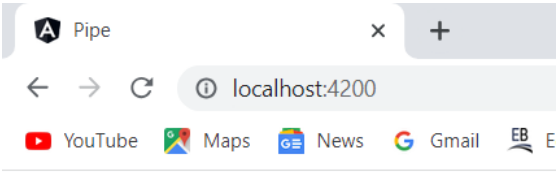
```ts
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-pipes',
  templateUrl: './pipes.component.html',
  styleUrls: ['./pipes.component.css']
})
export class PipesComponent implements OnInit {
  price=125;


  constructor() {}
  ngOnInit(): void {

  }

}
```

pipes.component.html

pipe > src > app > pipes > pipes.component.html > ...

Go to component

```html
<p>Price without format : {{price}}</p>
<p>Price with format : {{price|currency}}</p>
<p>Price with Indian format : {{price|currency:'INR'}}</p>
<p>Price with Indian format : {{price|currency:'INR':false}}</p>
```



Price without format : 125

Price with format : $125.00

Price with Indian format : ₹125.00

Price with Indian format : INR125.00

8. Sharing Data b/w Components

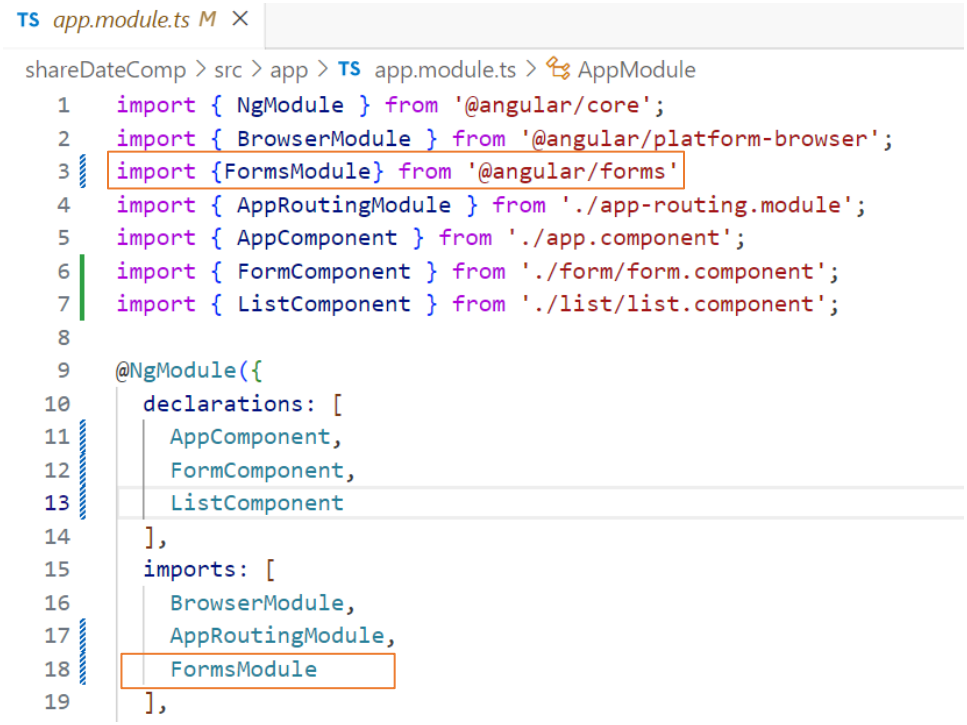
1) Parent comp to Child Comp

Step:1

Create to new application (ng new)

In App.Module.ts

Import FormsModule and make available in imports section.



Step:2

Create two new components form, list

In form.component.ts

Add cars array.

carName variable,

addCar() function

TS form.component.ts U

shareDateComp > src > app > form > TS form.component.ts > F

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

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent {
  cars:string[] = [];
  carName = "";

  addCar(){
    this.cars.push(this.carName);
    this.carName="";
    console.log(this,this.cars);

  }

}
```

Push carName input to cars array

Empty the input box after submit

Check input values are there in console.

form.component.html U

shareDateComp > src > app > form > form.component.html > app-list

Go to component

```html
<h5>Form Component</h5>
<form action="">
    <label for="">Car Name : </label>
    <input type="text" name="carname" id="" [(ngModel)]="carName">

    <button (click)="addCar()">Submit</button>
</form>
<br> <br>

<app-list [carsInput]="cars"></app-list>
```

Click event

Selector of Child comp(list)

Add selector of form component in App.component.html

app.component.html M

shareDateComp > src > app > app.component.ht

Go to component

```
1  <h3>App Component</h3>
2  <app-form></app-form>
```

Step:3

In list.component.ts

Import Input decorator from angular/core

Create cars string array same name in form.comp.ts

Add @Input decorator before the cars array.

list.component.ts U

shareDateComp > src > app > list > TS list.component.ts > ...

```
1   import { Component, Input } from '@angular/core';
2
3   @Component({
4     selector: 'app-list',
5     templateUrl: './list.component.html',
6     styleUrls: ['./list.component.css']
7   })
8   export class ListComponent {
9    @Input('carsInput') cars:string[] = [];
10  }
```

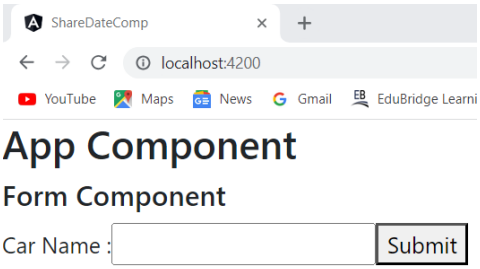
Made available in parent comp(form)

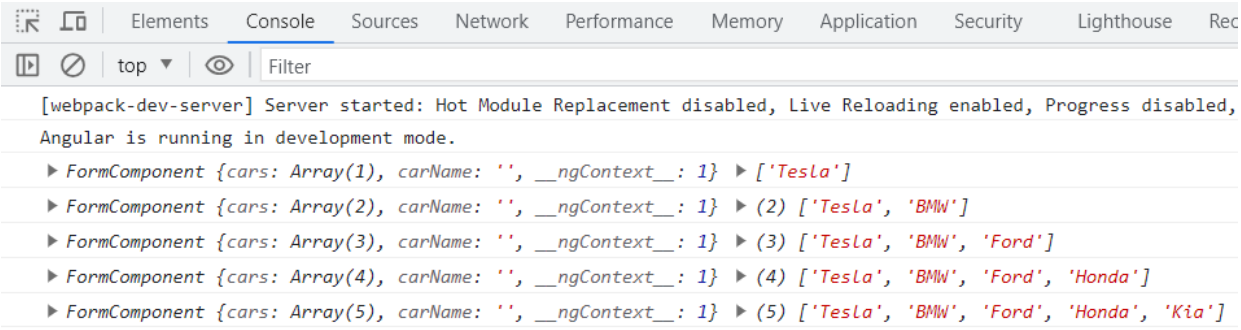
list.component.html U

shareDateComp > src > app > list > list.component.html > ...

Go to component

```
1   <h5>List Component</h5>
2   <div *ngIf="cars.length>0;then carList;else nodata "></div>
3   <ng-template #carList>
4       <ul>
5           <li *ngFor="let carName of cars">{{carName}}</li>
6       </ul>
7   </ng-template>
8   <ng-template #nodata>
9       <p>No Data Available...!</p>
10  </ng-template>
```





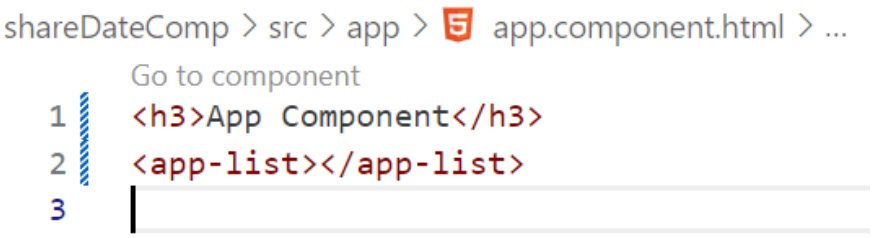
2) Child comp to parent comp

Step:1

Add child component selector(app.list) to App.component.html

Add parent component selector (app.form) to child component html (list.comp.html)

Create carDataUpdated function in list.comp.ts (child comp)



Output variable name

shareDateComp > src > app > list > list.component.html > ...

```
Go to component
<app-form (carAdded)="carDataUpdated($event)" ></app-form>
<h5>List Component</h5>

<div *ngIf="cars.length>0;then carList;else nodata "></div>
<ng-template #carList>
    <ul>
        <li *ngFor="let carName of cars">{{carName}}</li>
    </ul>
</ng-template>
<ng-template #nodata>
    <p>No Data Available...!</p>
</ng-template>
```

shareDateComp > src > app > list > TS list.component.ts > ...

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

@Component({
  selector: 'app-list',
  templateUrl: './list.component.html',
  styleUrls: ['./list.component.css']
})
export class ListComponent {
cars:string[] = [];

 carDataUpdated(carsData:string[]){
   this.cars = carsData;
 }
}
```

Step:2

Add Output decorator and EventEmiter in form.comp.ts (parent comp)

Create ouput decorator , name it (carAdded):EventEmitter<string[]>=new EventEmiter;

In addcars function create what to emit…

TS form.component.ts U

shareDateComp > src > app > form > TS form.component.ts > FormComponent

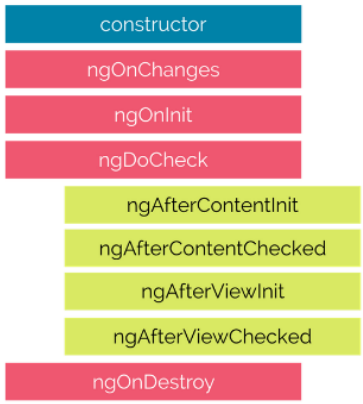
```ts
import { Component,Output,EventEmitter } from '@angular/core';

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
})
export class FormComponent {
  @Output() carAdded:EventEmitter<string[]>=new EventEmitter;
  cars:string[] = [];
  carName = "";

  addCar(){
    this.cars.push(this.carName);
    this.carName="";
    console.log(this,this.cars);
    this.carAdded.emit(this.cars);

  }

}
```

9. Component Life Cycle Hooks

Understand the different phases an Angular component goes through from being created to being destroyed.
Know how to hook into those phases and run your own code.
Know the order in which the different phases happen and what triggers each phase.

Phases

A component in Angular has a life-cycle, a number of different phases it goes through from birth to death.
We can hook into those different phases to get some pretty fine grained control of our application.
To do this we add some specific methods to our component class which get called during each of these life-cycle phases, we call those methods hooks.
The hooks are executed in this order:



Hooks for the Component

constructor
This is invoked when Angular creates a component or directive by calling new on the class.

ngOnChanges
Invoked every time there is a change in one of th input properties of the component.

ngOnInit
Invoked when given component has been initialized.
This hook is only called once after the first ngOnChanges

ngDoCheck
Invoked when the change detector of the given component is invoked. It allows us to implement our own change detection algorithm for the given component.

Important
ngDoCheck and ngOnChanges should not be implemented together on the same component.

ngOnDestroy
This method will be invoked just before Angular destroys the component.
Use this hook to unsubscribe observables and detach event handlers to avoid memory leaks.

Hooks for the Component's Children

ngAfterContentInit
Invoked after Angular performs any content projection into the component's view (see the previous lecture on Content Projection for more info).

ngAfterContentChecked
Invoked each time the content of the given component has been checked by the change detection mechanism of Angular.

ngAfterViewInit
Invoked when the component's view has been fully initialized.

ngAfterViewChecked
Invoked each time the view of the given component has been checked by the change detection mechanism of Angular.

1) ngOnChanges

Respond when Angular sets or resets data-bound input properties. The method receives a SimpleChanges object of current and previous property values.

Angular

Step:1

app.module.ts M

lifeCycleHooks > src > app > app.module.ts > ...

```ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import {FormsModule} from '@angular/forms'
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HeadingComponent } from './heading/heading.component';

@NgModule({
  declarations: [
    AppComponent,
    HeadingComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
```

Step:2

app.component.html M

lifeCycleHooks > src > app > app.component.html > app-heading

Go to component

```html
<h3>App component</h3>
Title : <input type="text" [(ngModel)]="title">
<br><br>

<app-heading [text]="title"></app-heading>
```

Step:3

Create new component

heading.component.html U

lifeCycleHooks > src > app > heading > heading.comp

Go to component

```html
<p>heading Component</p>
<h1>{{text}}</h1>
```

Step:4

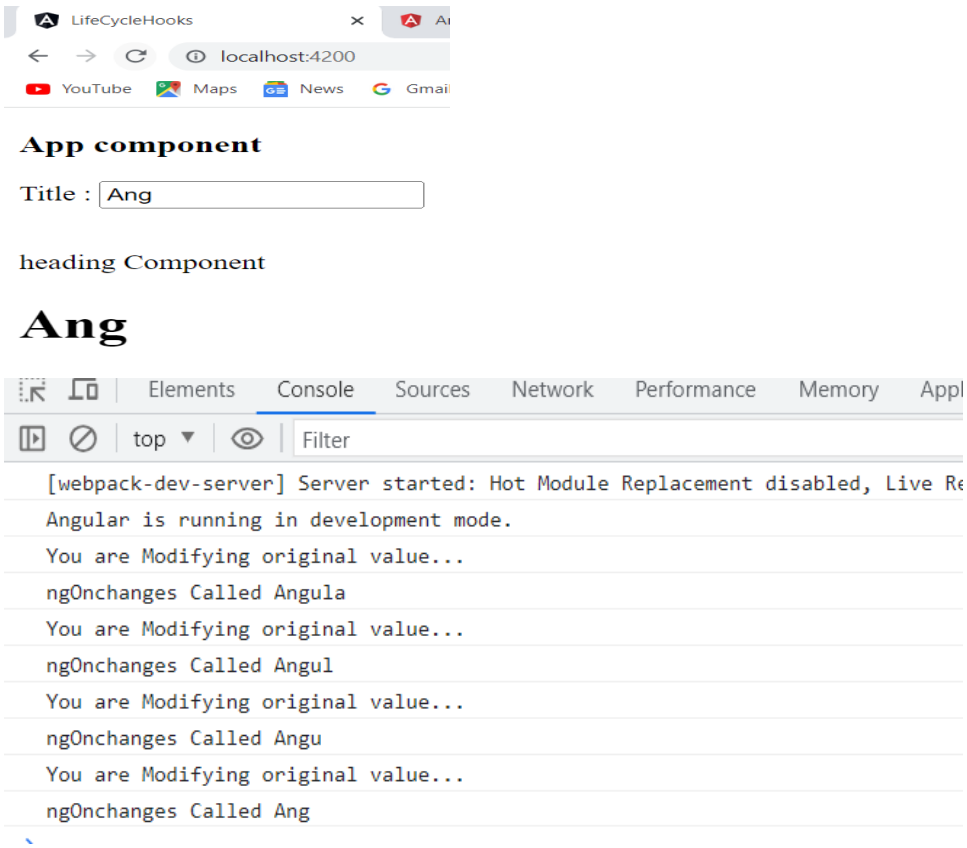
TS heading.component.ts U

lifeCycleHooks > src > app > heading > TS heading.component.ts > HeadingComponent > ngOnChanges

```
1  import { Component,Input,OnChanges } from '@angular/core';
2  import { SimpleChanges} from '@angular/core';
3  @Component({
4    selector: 'app-heading',
5    templateUrl: './heading.component.html',
6    styleUrls: ['./heading.component.css']
7  })
8  export class HeadingComponent implements OnChanges{
9    @Input() text='';
10   ngOnChanges(changes: SimpleChanges):void{
11     if(!changes['text'].isFirstChange()){
12       console.log("You are Modifying original value...");
13       console.log("ngOnchanges Called",changes['text'].currentValue);
14
15     }
```

Output:

It displayed each and every changes we made in that text box



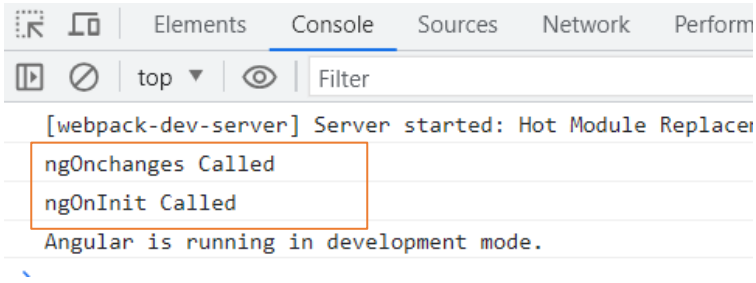
2) ngOnInit()

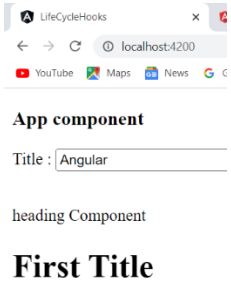
Initialize the directive or component after Angular first displays the data-bound properties and sets the directive or component's input properties. See details in Initializing a component or directive in this document.

TS heading.component.ts U

lifeCycleHooks > src > app > heading > TS heading.component.ts > HeadingComponent

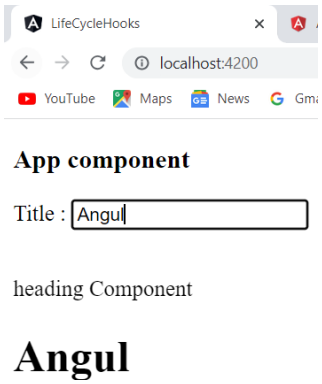
```
1   import { Component,Input,OnChanges, OnInit } from '@angular/core';
2   import { SimpleChanges} from '@angular/core';
3
4   @Component({
5     selector: 'app-heading',
6     templateUrl: './heading.component.html',
7     styleUrls: ['./heading.component.css']
8   })
9   export class HeadingComponent implements OnChanges, OnInit{
10    @Input() text='';
11
12    ngOnInit(): void {
13      console.log('ngOnInit Called');
14      this.text = "First Title"
15    }
16
17    ngOnChanges(changes: SimpleChanges):void{
18      console.log("ngOnchanges Called");
19    }
20
21  }
```





ngOnInit() is a better place to write “actual work code” that we need to execute as soon as the class is instantiated.

used to get data from database before the actual code work



3) ngDoCheck

helps to notice check array values are changed or not.

Step:1

app.component.ts M

lifeCycleHooks > src > app > app.component.ts > AppCom

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular';

  titles:string[] =[];

  addTitle(){
    this.titles.push(this.title);
    console.log(this.titles);

  }
}
```

app.component.html M

lifeCycleHooks > src > app > app.component.html > app-heading

Go to component

```html
<h3>App component</h3>
Title : <input type="text" [(ngModel)]="title">
<button (click)="addTitle()"> Add </button>
<br><br>


<app-heading [text]="titles"></app-heading>
```

Step:2

TS heading.component.ts U

lifeCycleHooks > src > app > heading > TS heading.component.ts > HeadingComponent > ngOnInit

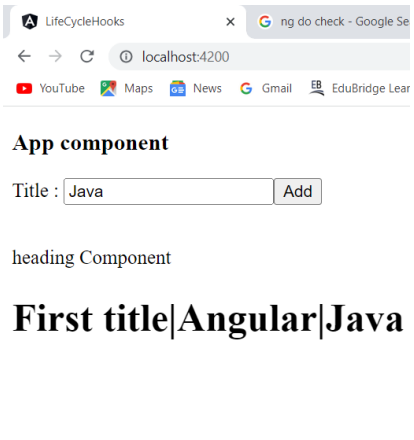
```ts
import { Component,Input,OnChanges, OnInit,DoCheck } from '@angular/core';
import { SimpleChanges} from '@angular/core';

@Component({
  selector: 'app-heading',
  templateUrl: './heading.component.html',
  styleUrls: ['./heading.component.css']
})
export class HeadingComponent implements OnChanges, OnInit, DoCheck{
  @Input() text:string[]=[];

  noOfTitles = 0;

  ngOnInit(): void {
    console.log('ngOnInit Called');
    this.text.push ('First title');
    this.noOfTitles = 1;
  }
  ngOnChanges(changes: SimpleChanges):void{
    console.log("ngOnchanges Called");
  }
  ngDoCheck(): void {
    if(this.text.length>this.noOfTitles){
      console.log('you are modifing original value');
      console.log('ngDocheck Called ');
      this.noOfTitles++;

    }
  }
}
```

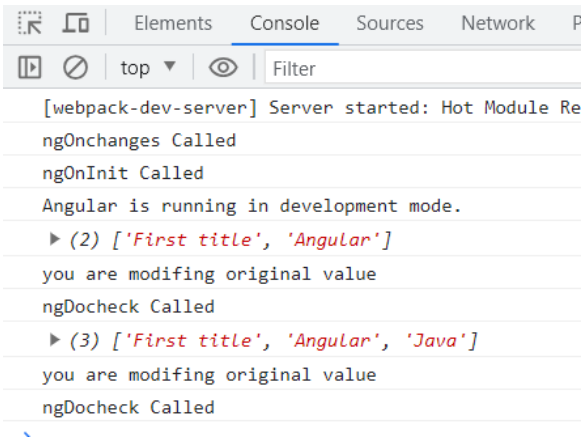
heading.component.html U

lifeCycleHooks > src > app > heading > heading.component.

Go to component

```html
<p>heading Component</p>
<h1>{{text.join('|')}}</h1>
```





4) ngOnDestory

A lifecycle hook that is called when a directive, pipe, or service is destroyed. Use for any custom cleanup that needs to occur when the instance is destroyed.

heading.component.ts U

lifeCycleHooks > src > app > heading > heading.component.ts > HeadingComponent > ngOnInit

```ts
import { Component,Input,OnChanges, OnInit, DoCheck, OnDestroy } from '@angular/core';
import { SimpleChanges} from '@angular/core';

@Component({
  selector: 'app-heading',
  templateUrl: './heading.component.html',
  styleUrls: ['./heading.component.css']
})
export class HeadingComponent implements OnChanges, OnInit, DoCheck, OnDestroy{
  @Input() text:string[]=[];

  noOfTitles = 0;

  ngOnInit(): void {
    console.log('ngOnInit Called');
    this.text.push ('First title');
    this.noOfTitles = 1;
  }
  ngOnChanges(changes: SimpleChanges):void{
    console.log("ngOnchanges Called");
  }
  ngDoCheck(): void {

    if(this.text.length>this.noOfTitles){
      console.log('you are modifing original value');
      console.log('ngDocheck Called ');
      this.noOfTitles++;
    }
  }
  ngOnDestroy(): void {
    console.log('ngOnDestroy Called ');

  }
}
```

app.component.html M

lifeCycleHooks > src > app > app.component.html > app-heading

```html
Go to component
<h3>App component</h3>
Title : <input type="text" [(ngModel)]="title">
<button (click)="addTitle()"> Add </button> <br>
Disable heading : <input type="checkbox" [(ngModel)]="disableHeading">
<br><br>


<app-heading *ngIf="!disableHeading" [text]="titles"></app-heading>
```

TS app.component.ts M

lifeCycleHooks > src > app > TS app.component.ts > ...

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular';

  titles:string[] =[];
  disableHeading = false;

  addTitle(){
    this.titles.push(this.title);
    console.log(this.titles);

  }
}
```

Before disable:

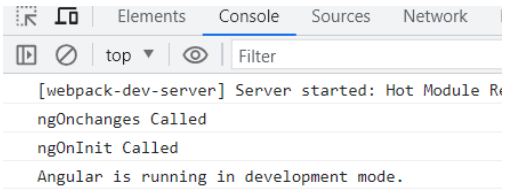
App component

Title : Angular Add

Disable heading :

heading Component

First title

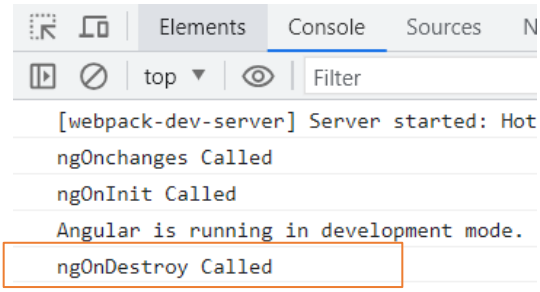


After Disable

App component

Title : Angular Add

Disable heading :



10. Forms

Handling user input with forms is the cornerstone of many common applications. Applications use forms to enable users to log in, to update a profile, to enter sensitive information, and to perform many other data-entry tasks.

Reactive forms and template-driven forms process and manage form data differently. Each approach offers different advantages.

Reactive forms

Provide direct, explicit access to the underlying form's object model. Compared to template-driven forms, they are more robust: they're more scalable, reusable, and testable. If forms are a key part of your application, or you're already using reactive patterns for building your application, use reactive forms.

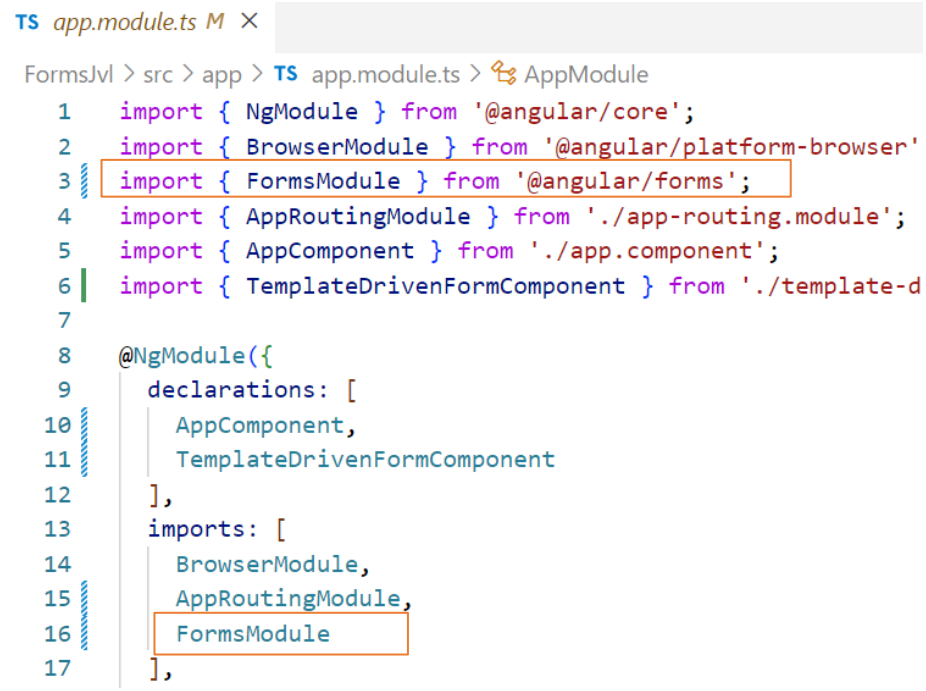
Template-driven forms

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 straightforward 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.

1) Template-Driven Forms

Step:1

Import FormsModule



Step:2

Create new component

Copy selector and paste in App.component.html

app.component.html M

FormsJvl > src > app > app.component.html > app-template-driven-form

Go to component

```
<h2>App Component</h2>
<br><br>
<app-template-driven-form></app-template-driven-form>
```

template-driven-form.component.html U

FormsJvl > src > app > template-driven-form > template-driven-form.component.html > div.container > form > div > p > label

Go to component

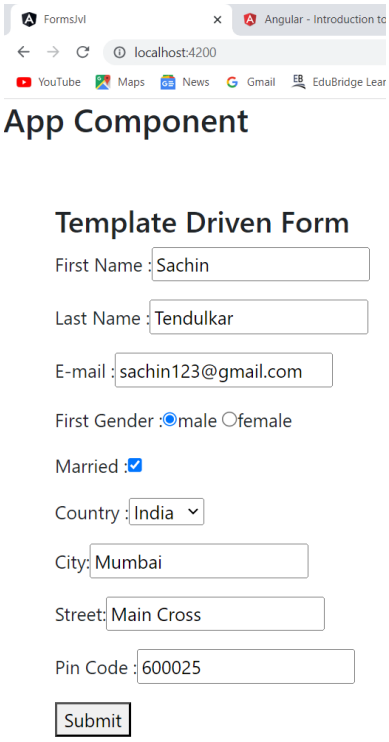
```
<div class="container">
    <h3>Template Driven Form</h3>
    <form #contactForm="ngForm" (submit)="onSubmit(contactForm)">
        <p>
            <label for="fname">First Name : </label>
            <input type="text" name="fname" id="fname" [(ngModel)]="contact.fname">
        </p>
        <p>
            <label for="lname">Last Name : </label>
            <input type="text" name="lname" id="lname" [(ngModel)]="contact.lname">
        </p>
        <p>
            <label for="email">E-mail : </label>
            <input type="email" name="email" id="email" [(ngModel)]="contact.email">
        </p>
        <p>
            <label for="gender">First Gender : </label>
            <input type="radio" value="male" name="gender" id="gender" [(ngModel)]="contact.gender">male
            <input type="radio" value="female" name="gender" id="gender" [(ngModel)]="contact.gender">female
        </p>
        <p>
            <label for="isMarried">Married : </label>
            <input type="checkbox" name="isMarried" id="isMarried" [(ngModel)]="contact.isMarried">
        </p>
        <p>
            <label for="country">Country : </label>
            <select name="country" id="country" [(ngModel)]="contact.country">
                <option selected="" value="">Select</option>
                <option selected="" [ngValue]="country.id" *ngFor="let country of countryList">{{country.name}}</option>
            </select>
        </p>
        <div ngModelGroup="address">
            <p>
                <label for="city">City: </label>
                <input type="text" name="city" id="city" [(ngModel)]="contact.address.city">
            </p>
            <p>
                <label for="street">Street:</label>
                <input type="text" name="street" id="street"  [(ngModel)]="contact.address.street">
            </p>
            <p>
                <label for="pincode">Pin Code :</label>
                <input type="number" name="pincode" id="pincode"  [(ngModel)]="contact.address.pincode">
            </p>

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

    </form>
</div>
```

TS template-driven-form.component.ts U ×

FormsJvl > src > app > template-driven-form > TS template-driven-form.component.ts > TemplateDr

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

@Component({
  selector: 'app-template-driven-form',
  templateUrl: './template-driven-form.component.html',
  styleUrls: ['./template-driven-form.component.css']
})
export class TemplateDrivenFormComponent implements OnInit {
  countryList:Country[]=[
    new Country('1','India'),
    new Country('2','USA'),
    new Country('3','UK'),

  ];
  contact!:Contact;
  ngOnInit(): void {
    this.contact={
      fname:"Sachin",
      lname:"Tendulkar",
      email:"sachin123@gmail.com",
      gender:"male",
      isMarried:true,
      country:'1',
      address:{
        city:"Mumbai",
        street:"Main Cross",
        pincode:"600025",
      }
    }
  }

  onSubmit(form:NgForm){
      console.log(form.value);

  }
}
class Country {
  id:String;
  name:string;

  constructor(id:string, name:string){
      this.id=id;
      this.name=name;
  }
}
class Contact{
  fname!:string;
  lname!:string;
  email!:string;
  gender!:string;
  isMarried!:boolean;
  country!:string;
  address! :{
    city:string;
    street:string;
    pincode:string;
  }

}
```

Helps to hold information



2) Reactive Forms

Reactive forms (also known as Model-driven forms) are one of the two ways to build Angular forms.

Reactive forms are forms where we define the structure of the form in the component class. i.e. we create the form model with Form Groups, Form Controls, and FormArrays. We also define the validation rules in the component class.

Step:1 Import FormsModule and ReactiveFormsModule

app.module.ts M

formJvl > src > app > app.module.ts > ...

```
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { FormsModule,ReactiveFormsModule } from '@angular/forms';
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { ReactiveFormComponent } from './reactive-form/reactive-form.component';
7
8
9  @NgModule({
10   declarations: [
11     AppComponent,
12     ReactiveFormComponent
13   ],
14   imports: [
15     BrowserModule,
16     AppRoutingModule,
17     FormsModule,
18     ReactiveFormsModule
19   ],
```

Step:2

Generate new component and add selector file in app.component.html

app.component.html M

formJvl > src > app > app.component.html > app-reactive-form

Go to component

```
1  <h1>App Component</h1>
2
3  <br><br>
4
5  <app-reactive-form></app-reactive-form>
```

Step:3

reactive-form.component.ts U

formJvl > src > app > reactive-form > reactive-form.component.ts > ReactiveFormComponent

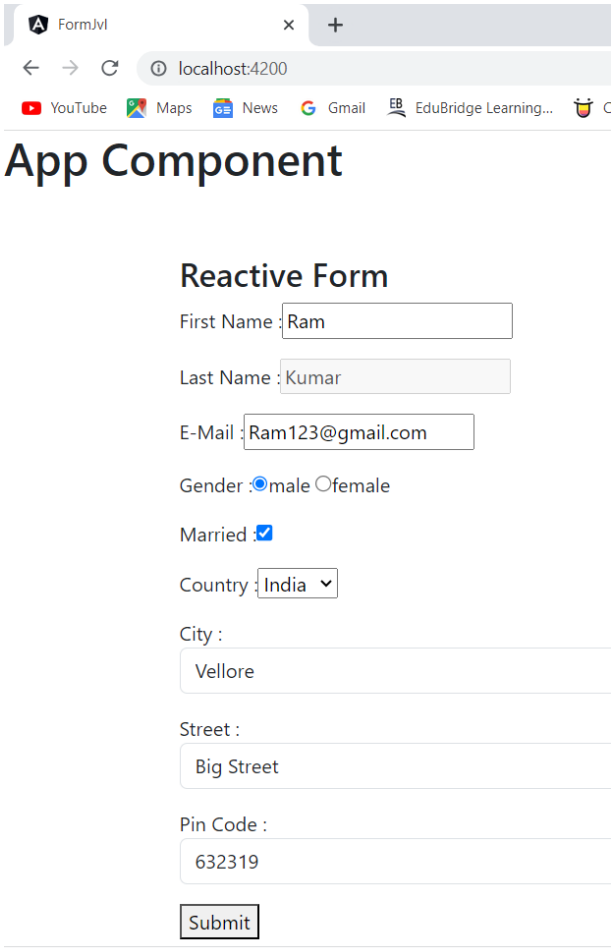
```
1  import { Component } from '@angular/core';
2  import { FormGroup, FormControl} from '@angular/forms'
3  @Component({
4    selector: 'app-reactive-form',
5    templateUrl: './reactive-form.component.html',
6    styleUrls: ['./reactive-form.component.css']
7  })
8  export class ReactiveFormComponent {
9
10   contactForm = new FormGroup({
11     fname : new FormControl('Ram'),
12     lname : new FormControl({
13       value: 'Kumar',
14       disabled: true
15     }),
16     email : new FormControl('Ram123@gmail.com'),
17     gender : new FormControl('male'),
18     isMarried : new FormControl(true),
19     country : new FormControl(1),
20     address : new FormGroup({
21       city : new FormControl('Vellore'),
22       street : new FormControl('Big Street'),
23       pincode : new FormControl(632319)
24     })
25   })
26
27   onSubmit(){
28     console.log(this.contactForm.value);
29
30   }
31
32
33 }
```

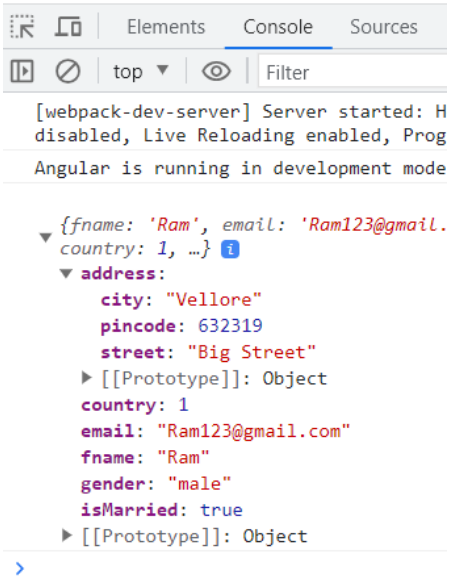
reactive-form.component.html U

formJvl > src > app > reactive-form > reactive-form.component.html > div.container > form > div

Go to component

```
<div class="container">
    <h3>Reactive Form</h3>

    <form [formGroup]="contactForm" (submit)="onSubmit()">
        <p>
            <label for="fname">First Name : </label>
            <input type="text" name="fname" id="fname" formControlName="fname">
        </p>

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

        <p>
            <label for="email">E-Mail : </label>
            <input type="email" name="email" id="email" formControlName="email">
        </p>

        <p>
            <label for="gender">Gender : </label>
            <input type="radio" name="gender" id="gender" value="male" formControlName="gender">male
            <input type="radio" name="gender" id="gender" value="female" formControlName="gender">female
        </p>
        <p>
            <label for="isMarried"> Married : </label>
            <input type="checkbox" name="isMarried" id="isMarried" formControlName="isMarried">
        </p>

         <p>
            <label for="country">Country : </label>
            <select name="country" id="country" formControlName="country">
                <option value="0">Select</option>
                <option value="1">India</option>
                <option value="2">USA</option>
                <option value="3">UK</option>
                <option value="4">Turky</option>
            </select>
        </p>
        <div formGroupName="address">
            <p>
            <label for="city"> City : </label>
            <input type="text" name="city" id="city" class="form-control" formControlName="city">
            </p>
            <p>
                <label for="street"> Street : </label>
                <input type="text" name="street" id="street" class="form-control" formControlName="street">
            </p>
            <p>
                <label for="pincode">Pin Code : </label>
                <input type="number" name="pincode" id="pincode" class="form-control" formControlName="pincode">
            </p>


        </div>

        <button>Submit</button>
    </form>

</div>
```





3) Validation using Template driven form

ngNativeValidate in form input tag helps to avoid FormsModule default ban of HTML validator and it helps to work HTML validator (required)

Step:1

app.module.ts M

FormsJvl > src > app > app.module.ts > AppModule

```
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { FormsModule } from '@angular/forms';
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { TemplateDrivenFormComponent } from './template-driven-form/template-driven-form.component';
7
8  @NgModule({
9    declarations: [
10     AppComponent,
11     TemplateDrivenFormComponent
12   ],
13   imports: [
14     BrowserModule,
15     AppRoutingModule,
16     FormsModule
17   ],
18   providers: [],
19   bootstrap: [AppComponent]
20 })
21 export class AppModule { }
```

Step:2

app.component.html M

FormsJvl > src > app > app.component.html > app-template-driven-form

Go to component

```
<h2>App Component</h2>
<br><br>
<app-template-driven-form></app-template-driven-form>
```

Step:3

Create a new component

template-driven-form.component.ts U

FormsJvl > src > app > template-driven-form > template-driven-form.component.ts > TemplateDrive

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

@Component({
  selector: 'app-template-driven-form',
  templateUrl: './template-driven-form.component.html',
  styleUrls: ['./template-driven-form.component.css']
})
export class TemplateDrivenFormComponent implements OnInit {
  countryList:Country[]=[
    new Country('1','India'),
    new Country('2','USA'),
    new Country('3','UK'),

  ];
  contact!:Contact;
  ngOnInit(): void {
    this.contact={
      fname:"",
      lname:"",
      email:"",
      gender:"",
      isMarried:false,
      country:'',
      address:{
        city:"",
        street:"",
        pincode:"",
      }
    }
  }

  onSubmit(form:NgForm){
      console.log(form.value);

  }
}
```

For individual validation check put the name of the individual

```
onSubmit(formcontrol:NgControl){
  console.log(formcontrol);
```

```typescript
class Country {
  id:String;
  name:string;

  constructor(id:string, name:string){
      this.id=id;
      this.name=name;
  }
}
class Contact{
  fname!:string;
  lname!:string;
  email!:string;
  gender!:string;
  isMarried!:boolean;
  country!:string;
  address! :{
    city:string;
    street:string;
    pincode:string;
  }

}
```

Step:4

For individual validation check put the name of the individual

Eg: onSubmit(fname)

template-driven-form.component.html U

FormsJvl > src > app > template-driven-form > template-driven-form.component.html > div.container > form > p > span > span.text-danger

Go to component

```
<form #contactForm="ngForm" (submit)="onSubmit(fname)">
```

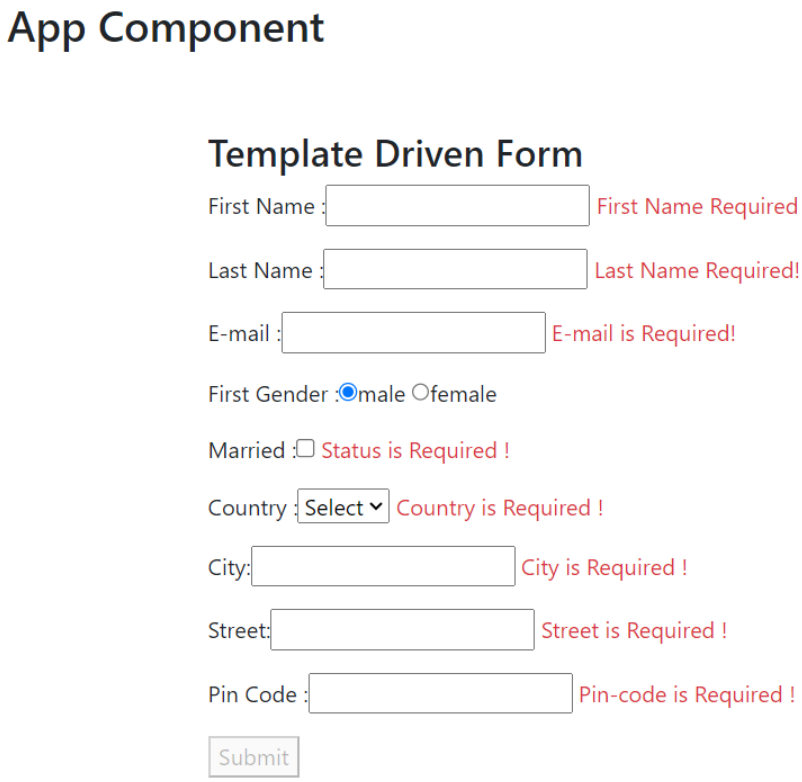
```html
<div class="container">
    <h3>Template Driven Form</h3>
    <form #contactForm="ngForm" (submit)="onSubmit(contactForm)">
        <p>
            <label for="fname">First Name : </label>
            <input type="text" name="fname" #fname="ngModel" id="fname" [(ngModel)]="contact.fname" required
                minlength="10">
            <span *ngIf="fname.valid == false && (fname.touched==true || fname.dirty)">
                <span class="text-danger" *ngIf="fname.errors?.['minlength']">
                    minumum length 10 Characters
                </span>
                <span class="text-danger" *ngIf="fname.errors?.['required']">
                    First Name Required
                </span>
            </span>
        </p>
        <p>
            <label for="lname">Last Name : </label>
            <input type="text" name="lname" #lname="ngModel" id="lname" [(ngModel)]="contact.lname" maxlength="15"
                pattern="^[a-zA-Z]+$" required>
            <span *ngIf="lname.valid == false && (lname.touched==true || lname.dirty)">
                <span class="text-danger" *ngIf="lname.errors?.['pattern']">
                    Characters only!
                </span>
                <span class="text-danger" *ngIf="lname.errors?.['required']">
                    Last Name Required!
                </span>
            </span>
        </p>
        <p>
            <label for="email">E-mail : </label>
            <input type="email" name="email" #email="ngModel" id="email" [(ngModel)]="contact.email" required>
            <span class="text-danger" *ngIf="email.valid == false && (email.touched==true || email.dirty)">
                E-mail is Required!
            </span>

        </p>
```

```
        <p>
            <label for="gender">First Gender : </label>
            <input type="radio" value="male" #gender="ngModel" name="gender" id="gender" [(ngModel)]="contact.gender"
                required>male
            <input type="radio" value="female" #gender="ngModel" name="gender" id="gender" [(ngModel)]="contact.gender"
                required>female
            <span class="text-danger" *ngIf="gender.valid == false && (gender.touched==true || gender.dirty)">
                <span class="text-danger" *ngIf="gender.errors?.['required']">
                    Gender is Required !
                </span>

            </span>
        </p>
        <p>
            <label for="isMarried">Married : </label>
            <input type="checkbox" name="isMarried" #isMarried="ngModel" id="isMarried" [(ngModel)]="contact.isMarried"
                required>
            <span class="text-danger" *ngIf="isMarried.valid == false && (isMarried.touched==true || isMarried.dirty)">
                <span class="text-danger" *ngIf="isMarried.errors?.['required']">
                    Status is Required !
                </span>

            </span>
        </p>
        <p>
            <label for="country">Country : </label>
            <select name="country" #country="ngModel" id="country" [(ngModel)]="contact.country" required>
                <option selected="" value="">Select</option>
                <option selected="" [ngValue]="country.id" *ngFor="let country of countryList">{{country.name}}</option>
            </select>
            <span class="text-danger" *ngIf="country.valid == false && (country.touched==true || country.dirty)">
                <span class="text-danger" *ngIf="country.errors?.['required']">
                    Country is Required !
                </span>
            </span>
        </p>
        <div ngModelGroup="address">
            <p>
                <label for="city">City: </label>
                <input type="text" name="city" #city="ngModel" id="city" [(ngModel)]="contact.address.city" required>
                <span class="text-danger" *ngIf="city.valid == false && (city.touched==true || city.dirty)">
                    <span class="text-danger" *ngIf="city.errors?.['required']">
                        City is Required !
                    </span>
                </span>
            </p>
            <p>
                <label for="street">Street:</label>
                <input type="text" name="street" #street="ngModel" id="street" [(ngModel)]="contact.address.street" required>
                <span class="text-danger" *ngIf="street.valid == false && (street.touched==true || street.dirty)">
                    <span class="text-danger" *ngIf="street.errors?.['required']">
                        Street is Required !
                    </span>
                </span>
            </p>
            <p>
                <label for="pincode">Pin Code :</label>
                <input type="number" name="pincode" #pincode="ngModel" id="pincode" [(ngModel)]="contact.address.pincode" required>
                <span class="text-danger" *ngIf="pincode.valid == false && (pincode.touched==true || pincode.dirty)">
                    <span class="text-danger" *ngIf="pincode.errors?.['required']">
                        Pin-code is Required !
                    </span>
                </span>
            </p>

        </div>
        <p>
            <button type="submit"[disabled]="!contactForm.valid">Submit</button>
        </p>

    </form>
</div>
```

For individual validation check put the name of the individual

`disabled]="contactForm.valid"`

Output:



4) Validation using Reactive form

Step:1

app.module.ts M

formJvl > src > app > app.module.ts > AppModule

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule,ReactiveFormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ReactiveFormComponent } from './reactive-form/reactive-form.component';


@NgModule({
  declarations: [
    AppComponent,
    ReactiveFormComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Step: 2

app.component.html M

formJvl > src > app > app.component.html > app-reactive-form

Go to component

```
1  <h1>App Component</h1>
2
3  <br><br>
4
5  <app-reactive-form></app-reactive-form>
```

Step:3

Create a component

reactive-form.component.ts U

formJvl > src > app > reactive-form > reactive-form.component.ts > ReactiveFormComponent > (get) pincode

```
1  import { Component } from '@angular/core';
2  import { FormGroup, FormControl, Validators} from '@angular/forms'
3  @Component({
4    selector: 'app-reactive-form',
5    templateUrl: './reactive-form.component.html',
6    styleUrls: ['./reactive-form.component.css']
7  })
8  export class ReactiveFormComponent {
9
10   get fname(){
11     return this.contactForm.get('fname');
12   }
13   get lname(){
14     return this.contactForm.get('lname');
15   }
16   get email(){
17     return this.contactForm.get('email');
18   }
19   get gender(){
20     return this.contactForm.get('gender');
21   }
22   get isMarried(){
23     return this.contactForm.get('isMarried');
24   }
25   get country(){
26     return this.contactForm.get('address')?.get('country');
27   }
28   get city(){
29     return this.contactForm.get('address')?.get('city');
30   }
31   get street(){
32     return this.contactForm.get('address')?.get('street');
33   }
34   get pincode(){
35     return this.contactForm.get('address')?.get('pincode');
36   }
```

```
38
39    contactForm = new FormGroup({
40      fname : new FormControl('',[Validators.required,Validators.minLength(8)]),
41      lname : new FormControl("",[Validators.required, Validators.maxLength(8),Validators.pattern("[a-zA-Z]+$")]),
42      email : new FormControl('',[Validators.required]),
43      gender : new FormControl('',[Validators.required]),
44      isMarried : new FormControl('',[Validators.requiredTrue]), //checkbox
45      country : new FormControl('',[Validators.required]),
46      address : new FormGroup({
47        city : new FormControl('',[Validators.required]),
48        street : new FormControl('',[Validators.required]),
49        pincode : new FormControl('',[Validators.required])
50      })
51    })
52
53    onSubmit(){
54      console.log(this.contactForm.value);
55
56    }
57
58
59  }
```

Step: 4

reactive-form.component.html U

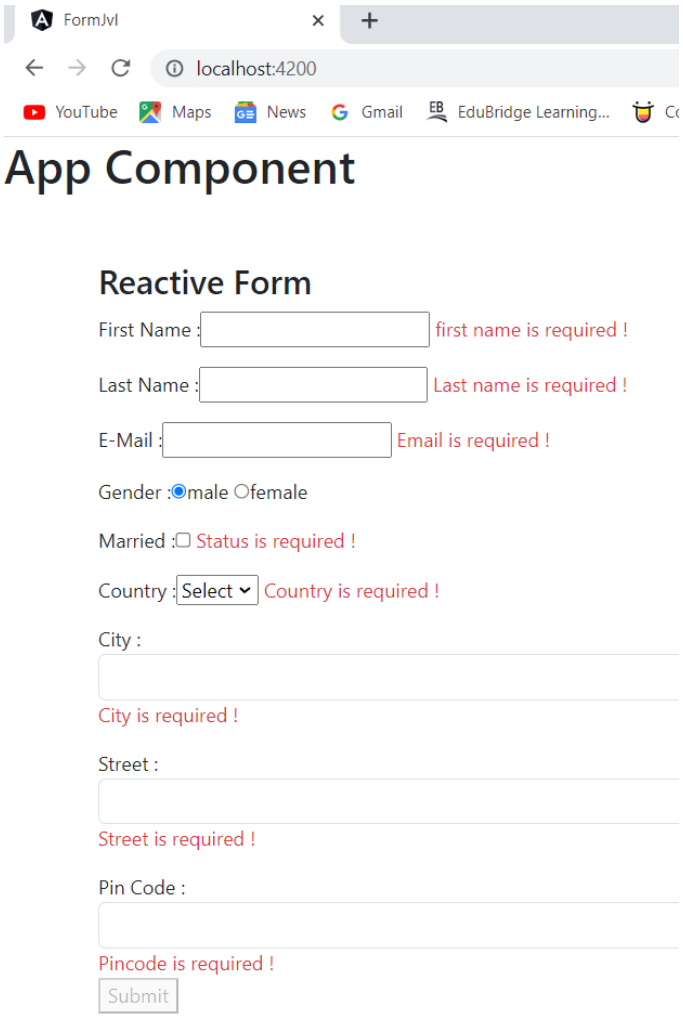
formJvl > src > app > reactive-form > reactive-form.component.html > div.container > form > div > div > span > span.text

Go to component

```
1  <div class="container">
2      <h3>Reactive Form</h3>
3
4      <form [formGroup]="contactForm" (submit)="onSubmit()">
5          <p>
6              <label for="fname">First Name : </label>
7              <input type="text" name="fname" id="fname" formControlName="fname">
8              <span *ngIf="fname?.valid==false && (fname?.touched==true || fname?.dirty==true)" >
9                  <span class="text-danger" *ngIf="fname?.errors?.['required']">
10                     first name is required !
11                 </span>
12                 <span class="text-danger" *ngIf="fname?.errors?.['minlength']">
13                     Minimum length is 8 character !
14                 </span>
15             </span>
16
17         </p>
18
19         <p>
20             <label for="lname">Last Name : </label>
21             <input type="text" name="lname" id="lname" formControlName="lname">
22             <span *ngIf="lname?.valid==false && (lname?.touched==true || lname?.dirty==true)" >
23                 <span class="text-danger" *ngIf="lname?.errors?.['required']">
24                     Last name is required !
25                 </span>
26                 <span class="text-danger" *ngIf="lname?.errors?.['maxlength']">
27                     maximum length is 8 character !
28                 </span>
29                 <span class="text-danger" *ngIf="lname?.errors?.['pattern']">
30                     only characters !
31                 </span>
32             </span>
33         </p>
34
```

```
    <p>
        <label for="email">E-Mail : </label>
        <input type="email" name="email" id="email" formControlName="email">
        <span *ngIf="email?.valid==false && (email?.touched==true || email?.dirty==true)" >
            <span class="text-danger" *ngIf="email?.errors?.['required']">
                Email is required !
            </span>
        </span>
    </p>

    <p>
        <label for="gender">Gender : </label>
        <input type="radio" name="gender" id="gender" value="male" formControlName="gender">male
        <input type="radio" name="gender" id="gender" value="female" formControlName="gender">female
        <span *ngIf="gender?.valid==false && (gender?.touched==true || gender?.dirty==true)" >
            <span class="text-danger" *ngIf="email?.errors?.['required']">
                gender is required !
            </span>
        </span>
    </p>
    <p>
        <label for="isMarried"> Married : </label>
        <input type="checkbox" name="isMarried" id="isMarried" formControlName="isMarried">
        <span *ngIf="isMarried?.valid==false && (isMarried?.touched==true || isMarried?.dirty==true)" >
            <span class="text-danger" *ngIf="isMarried?.errors?.['required']">
                Status is required !
            </span>
        </span>
    </p>

    <p>
        <label for="country">Country : </label>
        <select name="country" id="country" formControlName="country">
            <option >Select</option>
            <option value="1">India</option>
            <option value="2">USA</option>
            <option value="3">UK</option>
            <option value="4">Turky</option>
        </select>
        <span *ngIf="country?.valid==false && (country?.touched==true || country?.dirty==true)" >
            <span class="text-danger" *ngIf="country?.errors?.['required']">
                Country is required !
            </span>
        </span>
    </p>
    <div formGroupName="address">
        <p>
        <label for="city"> City : </label>
        <input type="text" name="city" id="city" class="form-control" formControlName="city">
        <span *ngIf="city?.valid==false && (city?.touched==true || city?.dirty==true)" >
            <span class="text-danger" *ngIf="city?.errors?.['required']">
                City is required !
            </span>
        </span>
        </p>
        <p>
            <label for="street"> Street : </label>
            <input type="text" name="street" id="street" class="form-control" formControlName="street">
            <span *ngIf="street?.valid==false && (street?.touched==true || street?.dirty==true)" >
                <span class="text-danger" *ngIf="street?.errors?.['required']">
                    Street is required !
                </span>
            </span>
        </p>
```

```
99              <div>
100                 <label for="pincode">Pin Code : </label>
101                 <input type="number" name="pincode" id="pincode" class="form-control" formControlName="pincode">
102                 <span *ngIf="pincode?.valid==false && (pincode?.touched==true || pincode?.dirty==true)" >
103                     <span class="text-danger" *ngIf="pincode?.errors?.['required']">
104                         Pincode is required !
105                     </span>
106                 </span>
107             </div>
108
109
110
111
112         </div>
113         <p>
114             <button type="submit" [disabled]="contactForm.valid==false">Submit</button>
115         </p>
116
117     </form>
118
119 </div>
```



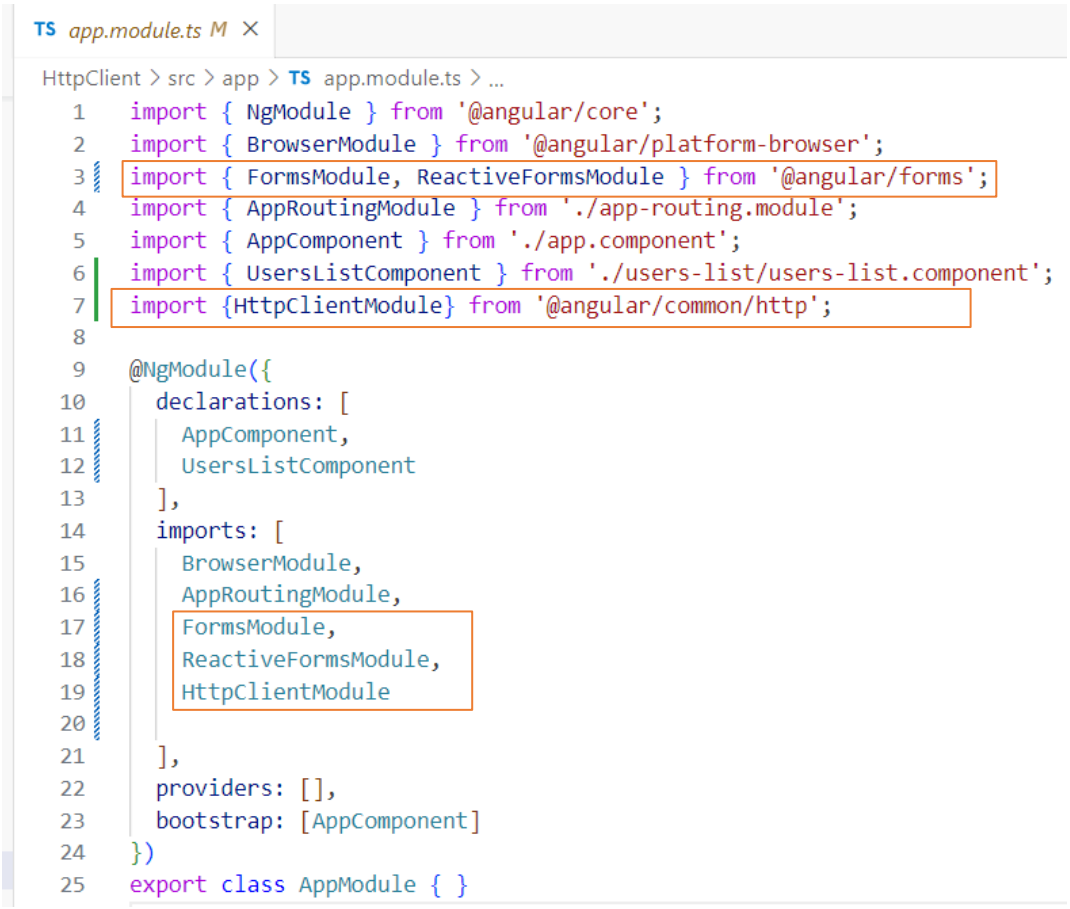
5) Http Client

Most front-end applications need to communicate with a server over the HTTP protocol, to download or upload data and access other back-end services. Angular provides a client HTTP API for Angular applications, the **HttpClient** service class in @angular/common/http.

The most commonly used HTTP request methods are GET, POST, PUT, PATCH, and DELETE.

Step:1

import HttpClientModule in app.module.ts



Step:2

Create new component and paste selector file of a new component to app.component.html

app.component.html M

HttpClient > src > app > app.component.html > app-users-list

Go to component

```
1  <h1 class="container">App Component</h1>
2
3  <app-users-list></app-users-list>
```

Step:3

users-list.component.ts U

HttpClient > src > app > users-list > users-list.component.ts > UsersListComponent > userForm

```
1  import { Component,OnInit } from '@angular/core';
2  import { HttpClient } from '@angular/common/http';
3  import { FormControl, FormGroup } from '@angular/forms';
4  @Component({
5    selector: 'app-users-list',
6    templateUrl: './users-list.component.html',
7    styleUrls: ['./users-list.component.css']
8  })
9  export class UsersListComponent implements OnInit {
10
11
12   userForm = new FormGroup(
13     {
14       name : new FormControl(''),
15       Email : new FormControl(''),
16     }
17   )
18
19     users:User[] = []
20     constructor(private http:HttpClient){
21
22     }
23     ngOnInit(){
24         this.getUsers().subscribe((response)=>{
25           console.log('response: ',response);
26           this.users = response
27         })
28
29     }
30
```

Fake server from

https://jsonplaceholder.typicode.com/

```
31    getUsers(){
32      return this.http.get<User[]>('https://jsonplaceholder.typicode.com/users');
33    }
34
35    addUser(){
36      return this.http.post<User>('https://jsonplaceholder.typicode.com/users',{
37        name : this.userForm.controls.name.value,
38        email : this.userForm.controls.Email.value
39      });
40    }
41    onSubmit(){
42      this.addUser().subscribe((response)=>{
43        this.users.push(response)
44      })
45
46    }
47  }
48
49  class User {
50    name!: string;
51    email!: string;
52  }
```

Step:4

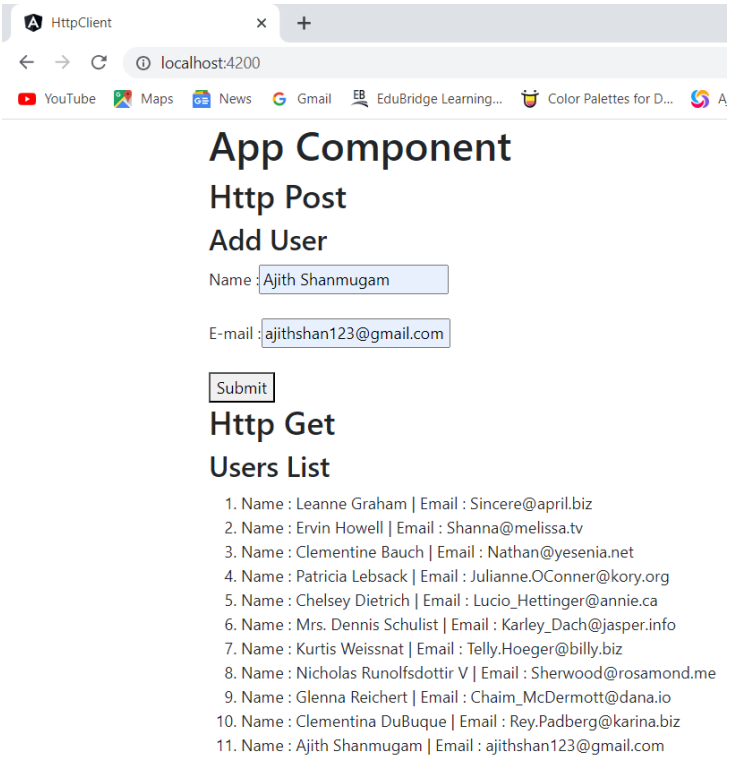
users-list.component.html U

HttpClient > src > app > users-list > users-list.component.html > ...

Go to component

```
1   <div class="container">
2       <h2>Http Post</h2>
3       <h3>Add User</h3>
4       <form [formGroup]="userForm" method="post" (submit)="onSubmit()">
5           <label for="name">Name : </label>
6           <input type="text" name="name" id="name" class="form-control" formControlName="name"><br><br>
7           <label for="Email">E-mail : </label>
8           <input type="email" name="Email" id="Email"  class="form-control" formControlName="Email"><br><br>
9           <button type="submit">Submit</button>
10      </form>
11
12
13  </div>
14
15
16
17  <div class="container">
18      <h2>Http Get</h2>
19      <h3>Users List</h3>
20
21
22      <ol>
23          <li *ngFor="let user of users">Name : {{user.name}} | Email : {{user.email}}</li>
24
25      </ol>
26  </div>
```

Output



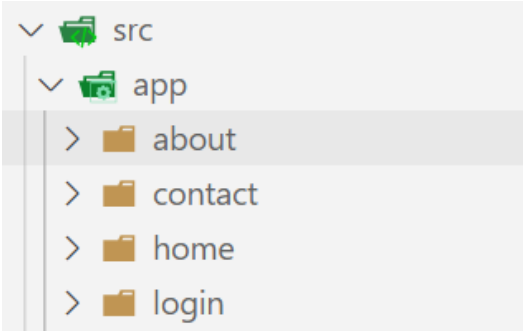
6) Angular Routing

Step:1

Create new application with routing -Yes

Step:2

Create new components with name of Home, Contact, About, Login



Step:3

In **app-routing.module.ts** create paths and import that.

src > app > TS app-routing.module.ts > ...

```ts
import { Component, NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { ContactComponent } from './contact/contact.component';
import { AboutComponent } from './about/about.component';
import { LoginComponent } from './login/login.component';

const routes: Routes = [
  {path:'home',component:HomeComponent},
  {path:'contact', component:ContactComponent},
  {path: 'about', component:AboutComponent},
  {path:'login', component:LoginComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Step:4

Create anchor tag replace the href with **rouerlink**

Create **router-outlet**

src > app > app.component.html > router-outlet

Go to component

```html
<div align="left">
<a routerLink="/home" routerLinkActive="Active">Home</a>   
<a routerLink="/contact" routerLinkActive="Active">Contact</a>   
<a routerLink="/about" routerLinkActive="Active">About</a>   
<a routerLink="/login" routerLinkActive="Active">Login</a>   
</div>

<router-outlet></router-outlet>
```

Output

Home Contact About Login

contact works!

Home Contact About Login

about works!