

Custom Directive:

So far, we have seen built in directives. Now we will see how to create custom directive. Let's consider a scenario where you are responsible for creating a custom directive which will be responsible to change colour of text.

In order to achieve this we will have to follow steps given below:

Step 1. We will have to create/add directive. To do so, we will have to use command.

ng g directive changecolor

Step 2. Once our directive will be created then open it. Here you will see that this directive class file is having new decorator that is @Directive. Which include selector. Here selector is playing same role means it's identifier for this directive. Using this selector, we can use this directive anywhere we want.

For implementing our requirement, we will have to use one class here and it's name is ElementRef.

So,

1. what is **ElementRef**? --ElementRef is a class that can hold a reference to a DOM element.
2. How to use it? –First we will have to add it in import at top of our directive.ts file and inject it using constructor.

```
TS changecolor.directive.ts X
src > app > TS changecolor.directive.ts > ChangecolorDirective > constructor
1  import { Directive , ElementRef } from '@angular/core';
2
3  @Directive({
4    selector: '[appChangecolor]'
5  })
6  export class ChangecolorDirective {
7
8    constructor(private element:ElementRef) { }
9
10 }
11
```

To complete our requirement I will have to add given below code here.

```
TS changecolor.directive.ts X  # app.component.css  <> app.component.html
src > app > TS changecolor.directive.ts > ChangecolorDirective
1  import { Directive ,ElementRef } from '@angular/core';
2
3  @Directive({
4    selector: '[appChangecolor]'
5  })
6  export class ChangecolorDirective {
7
8    constructor(private element:ElementRef) {
9      this.element.nativeElement.style.color='blue';
10     this.element.nativeElement.style.background='Yellow'
11   }
12
13 }
14
```

So, now I can say my custom directive is ready now. Wherever I will have to use this styling I can simply use this custom directive just by using it's selector.

Ex:

```
<h1 appChangeColor>Welcome to Citiustech</h1>
```

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.

Pipes are useful because you can use them throughout your application, while only declaring each pipe once.

App.component.ts:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'day7demos';
  userName='BhushanP';
  object: Object = {name: 'John', email: 'john@gmail.com',
    additionalInfo: {age: 33, phoneNumbers: [9975665249, 123456789,
987654321]}};
  a: number = 0.259;
  currentDate = new Date();
}
```

App.component.html

```

<h1>Welcome to Citiustech</h1>
<!-- Uppercase Pipe -->
<h2>Welcome {{userName | uppercase}}</h2>
<!-- Lowercase pipe -->
<h2>Welcome {{userName | lowercase}}</h2>
<!-- Json Pipe -->
<div>
  <p>Without JSON pipe:</p>
  <pre>{{object}}</pre>
  <p>With JSON pipe:</p>
  <pre>{{object | json}}</pre>
</div>

<!-- Percent Pipe -->
<div>
  <!--output '26%'-->
  <p>A: {{a | percent}}</p>
</div>

<!-- Date Pipe -->
<h2>{{currentDate}}</h2>
<h2>{{currentDate | date: 'short'}}</h2>
<h2>{{currentDate | date: 'medium'}}</h2>

```

For detail information about pipe visit.

<https://angular.io/api?query=pipe>

Custom Pipes

What we have seen for built in pipes, same way we can create our own pipe.

What is reason for having custom pipes?

Let's consider a scenario that there is data which we have to display but instead of displaying data as it we have to do some changes. Example: if string is very long and we have to display only 8 characters followed by ... then in order to achieve this we will create our pipe.

How to create custom pipe?

Either we can add using angular cli or we can create it manually.

Command to create new pipe is

ng g p pipe_name

```
PS C:\Users\BhushanP5\OneDrive - CitiusTech\Angular Calender Training\Day 8\Day 8> cd .\day7demos\  
PS C:\Users\BhushanP5\OneDrive - CitiusTech\Angular Calender Training\Day 8\Day 8\day7demos> ng g p cit  
ipipe  
CREATE src/app/citipipe.pipe.spec.ts (195 bytes)  
CREATE src/app/citipipe.pipe.ts (221 bytes)  
UPDATE src/app/app.module.ts (459 bytes)
```

Let's open our pipe and try to understand

```
1  import { Pipe, PipeTransform } from '@angular/core';  
2  
3  @Pipe({  
4    name: 'citipipe'  
5  })  
6  export class CitipipePipe implements PipeTransform {  
7  
8    transform(value: any) {  
9      // transform(value: any, limit:number) {  
10       // console.log(value);  
11       return value.substr(0, 5)+'...';  
12     }  
13  }
```

After creating pipe, we will get one pipe.


Pipe_name.pipe.ts. This file has **@Pipe** decorator in which we will have to provide name for our pipe. This class implements **PipeTransform** interface.

This interface has one method which we will have to implement that method is

transform(value:any){ }. In this method here is one parameter and also can have one more optional parameter. Whatever transformation we will have to implement that transformation has to develop in transform () method.

Let's try it by implementing it.

Step 1. We have created new pipe, open this pipe and add given below code.



```
<> app.component.html TS app.component.ts TS citipipe.pipe.ts X
day7demos > src > app > TS citipipe.pipe.ts > CitipipePipe > transform
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'citipipe'
5  })
6  export class CitipipePipe implements PipeTransform {
7
8    transform(value: any) {
9      // transform(value: any, limit:number) {
10     // console.log(value);
11     return value.substr(0, 5)+'...' ;
12   }
13
14 }
```

Now, add one property in app.component.ts.

```
app.component.html TS app.component.ts X TS citipipe.pipe.ts
day7demos > src > app > TS app.component.ts > AppComponent > greet
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'day7demos';
10  userName='BhushanP';
11  object: Object = {name: 'John', email: 'john@gmail.com',
12  additionalInfo: {age: 33, phoneNumbers: [9975665249, 123456789, 987654321]}};
13  a: number = 0.259;
14  currentDate = new Date();
15  greet='Good Afternoon Guys';
16 }
17
```

Open app.component.html

```
app.component.html X TS app.component.ts TS citipipe.pipe.ts
day7demos > src > app > app.component.html > h2
16 <!--output '26%'-->
17 <h2>A: {{a | percent}}</h2>
18 </div>
19
20 <!-- Date Pipe -->
21 <h2>{{currentDate}}</h2>
22 <h2>{{currentDate | date: 'short'}}</h2>
23 <h2>{{currentDate | date: 'medium'}}</h2>
24
25 <h2>{{greet | citipipe}}</h2>
```

Step 2. Now, we will implement pipe to accept parameter. For this we will create an array of employees name and we want to display only 5 characters from name and if name will be having more 5 characters then our pipes should show only 5 characters followed by three dots. ...

a. Add an array in **app.component.ts**

```
app.component.html TS app.component.ts X TS citipipe.pipe.ts
day7demos > src > app > TS app.component.ts > AppComponent > employees
7   })
8   export class AppComponent {
9     title = 'day7demos';
10    userName='BhushanP';
11    object: Object = {name: 'John', email: 'john@gmail.com',
12    additionalInfo: {age: 33, phoneNumbers: [9975665249, 123456789, 987654321]}};
13    a: number = 0.259;
14    currentDate = new Date();
15    greet='Good Afternoon Guys';
16    employees=[
17      'Paul Adams',
18      'Brian',
19      'Tom',
20      'Keanu Reeves',
21      'John Connor'
22    ]
}
```

b. Display this array in app.component.html file.

```
<h2>{{greet | citipipe}}</h2>
<ul *ngFor="let emp of employees">
  <li>{{emp}}</li>
</ul>
```

Now, add our pipe on it

```
<h2>{{greet | citipipe}}</h2>
<ul *ngFor="let emp of employees">
  <li>{{emp | citipipe}}</li>
</ul>
```

It will transform output.

c. Now, we will have to provide parameters.

Means number of characters we will have to provide dynamically.

Step 3. Now, our requirement says that if name of employee is less than limit then it shouldn't have three dots ...

Our final code will look like:


```
<> app.component.html TS app.component.ts TS citipipe.pipe.ts X
day7demos > src > app > TS citipipe.pipe.ts > CitipipePipe > transform
1 import { Pipe, PipeTransform } from '@angular/core';
2
3 @Pipe({
4   name: 'citipipe'
5 })
6 export class CitipipePipe implements PipeTransform {
7
8   // transform(value: any) {
9   transform(value: any, limit: number) {
10     // console.log(value);
11     // return value.substr(0, 5)+'...';
12     if (value.length > 8) {
13       return value.substr(0, limit) + '...';
14     } else {
15       return value;
16     }
17   }
18
19 }
```

Now, use our pipe with value as parameter as shown in given below picture.

```
<> app.component.html X TS app.component.ts TS citipipe.pipe.ts
day7demos > src > app > <> app.component.html > h2
21 <h2>{{currentDate | date: 'short'}}</h2>
22 <h2>{{currentDate | date: 'short'}}</h2>
23 <h2>{{currentDate | date: 'medium'}}</h2>
24
25 <h2>{{greet | citipipe:4}}</h2>
26 <ul *ngFor="let emp of employees">
27   <li>{{emp | citipipe:2}}</li>
28 </ul>
```

In this way, we can create our custom pipe.

Pure and Impure Pipe.

Whenever any pipe transforms our data that time angular execute that pipe.

Pure pipe are those pipes when pure change will be detected. So, pure pipes check when changes are detecting in values.

But impure pipes are getting executed on any change in our applications. So, impure pipes may affect performance of our application.

It's best practice to avoid impure pipes because it may affect performance of application.

Let's Understand it with an example.

Step 1. We will use array employees in `app.component.ts` file.

```
<> app.component.html • TS app.component.ts X TS citipipe.pipe.ts
day7demos > src > app > TS app.component.ts > AppComponent > employees
7   })
8   export class AppComponent {
9     title = 'day7demos';
10    userName='BhushanP';
11    object: Object = {name: 'John', email: 'john@gmail.com',
12    additionalInfo: {age: 33, phoneNumbers: [9975665249, 123456789, 987654321]}};
13    a: number = 0.259;
14    currentDate = new Date();
15    greet='Good Afternnon Guys';
16    employees=[
17      'Paul Adams',
18      'Brian',
19      'Tom',
20      'Keanu Reeves',
21      'John Conor'
22    ]
23  }
24
```

Step 2. Open app.component.html file and display this array using interpolation first then add text box and button to add employee runtime.

```
<> app.component.html • TS app.component.ts TS citipipe.pipe.ts
day7demos > src > app > <> app.component.html > button
21 <h2>{{currentDate}}</h2>
22 <h2>{{currentDate | date: 'short'}}</h2>
23 <h2>{{currentDate | date: 'medium'}}</h2>
24
25 <h2>{{greet | citipipe:4}}</h2>
26 <ul *ngFor="let emp of employees">
27   <li>{{emp | citipipe:2}}</li>
28 </ul>
29
30 <!-- pure and impure pipes -->
31 <input type="text" value="" #empName>
32 <button (click)="addEmp(empName.value)">Add</button>
33 <p>{{employees}}</p>
```

Step 3. implement this addEmp() method in app.component.ts file.

```
<> app.component.html ● TS app.component.ts X TS citipipe.pipe.ts
day7demos > src > app > TS app.component.ts > AppComponent > addEmp
10  username= 'brusnar';
11  object: Object = {name: 'John', email: 'john@gmail.com',
12  additionalInfo: {age: 33, phoneNumbers: [9975665249, 123456789, 987654
13  a: number = 0.259;
14  currentDate = new Date();
15  greet='Good Afternoon Guys';
16  employees=[
17    'Paul Adams',
18    'Brian',
19    'Tom',
20    'Keanu Reeves',
21    'John Connor'
22  ]
23  addEmp(value :any){
24    this.employees.push(value);
25  }
26  }
27
```

Step 4: Add new custompipe by using command.

ng g p custompipe.

After firing command, we will get a custom pipe.

Do one change in it. **Change return null to return value.join().**

```
<> app.component.html X TS app.component.ts TS custompipe.pipe.ts X TS citipipe.pipe.ts
day7demos > src > app > TS custompipe.pipe.ts > CustompipePipe > transform
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'custompipe'
5  })
6  export class CustompipePipe implements PipeTransform {
7
8    transform(value: unknown, ...args: unknown[]): unknown {
9      return value;
10    }
11  }
12
13
```

```
day7demos > src > app > TS custompipe.pipe.ts > CustompipePipe
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'custompipe',
5    pure:false
6  })
7  export class CustompipePipe implements PipeTransform {
8
9    transform(value: any, ...args: unknown[]): unknown {
10     return value.join();
11   }
12
13 }
```

Now, open app.component.html and add this pipe on our values.

```
<!-- pure and impure pipes -->
<input type="text" value="" #empName>
<button (click)="addEmp(empName.value)">Add</button>
<p>{{employees | custompipe}}</p>
```

Step 5: Now to go app.component.ts file and log array to console in addEmp() method.

```

employees=[
  'Paul Adams',
  'Brian',
  'Tom',
  'Keanu Reeves',
  'John Conor'
]
addEmp(value :any){
  this.employees.push(value);
  console.log(this.employees);
}
}

```

So now if we will add value in text box, it will be displayed in console but won't display on html.

Paul Adams,Brian,Tom,Keanu Reeves,John Conor

Console:

```

app.component.ts:25
(6) ['Paul Adams', 'Brian', 'Tom', 'Keanu Reeves', 'John Conor', 'Bhusha n']
app.component.ts:25
(7) ['Paul Adams', 'Brian', 'Tom', 'Keanu Reeves', 'John Conor', 'Bhusha n', 'Rakesh']

```

Now, go to custompipe and set property pure to false.

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'custompipe',
  pure:false
})
export class CustompipePipe implements PipeTransform {

  transform(value: any, ...args: unknown[]): unknown {
    return value.join();
  }
}
```

In this way pure and impure pipes works.

Angular Form:

- What Angular forms are?
- Types of Form building
- Form control Group control
- Simple example

Angular forms → All applications are made up of forms. You open any web application, you will must find any purpose form there it maybe registration, feedback, login, comment form. So, we can say forms are very important for web application.

In angular we can create very interactive forms. Angular supports two types of form.

1. Template driven form
2. Reactive form

Template Driven approach →

1. In this method conventional <form> tag is use to create form.
2. Angular automatically creates the object of form control with the help of directives.
3. Controls can be added to angular form using ngModel directive
4. Multiple controls can be grouped together using ngControlGroup module.
5. Form values can be generated using form.value method. Form data can be exported as json value and submit method will be called.
6. We can also provide validation by using basic html properties and objects. We will use directives to do custom validations.

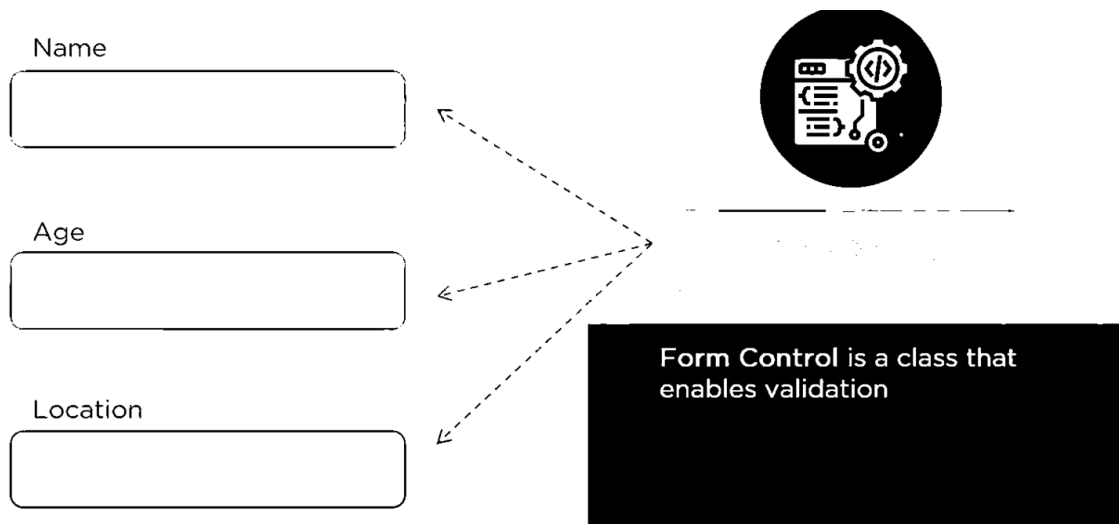
Reactive Form Approach →

1. Reactive form are code base forms.
2. Here components manages data flow.
Reactive form are code driven unlike template driven form.

3.No two way data binding use in reactive form.

Form Control and Form Group:

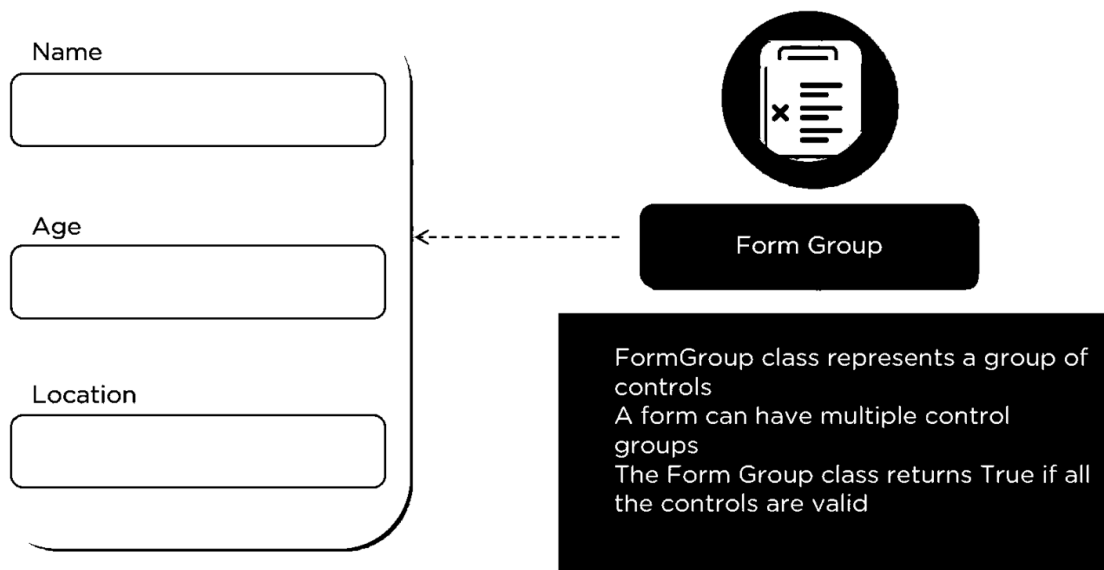
Form control:



Let's consider above form. We have three controls over there. So,

- a.form control is class that enables validation on these fields.
- b.For each input field an instance of this class will be created.
- c. These instances checks the values in these fields and observe whether they are touched, untouched, dirty, pristine, valid, invalid and so on.

Form Group



Let's try to understand it with demo.

Step 1. Add new component to our application.
regform.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
UPDATE src/app/app.module.ts (624 bytes)
PS C:\Users\BhushanP5\OneDrive - CitiusTech\Angular Calender Training\Day 8\Day 8\day7demos> ng g c regform
CREATE src/app/regform/regform.component.html (22 bytes)
```

Step 2. Next, we will have to import form module.
For this purpose open, app.module.ts file. And
import formModule at the top and add it to import
array as shown in given picture.

```
TS mydirective.directive.ts  app.component.html  TS app.module.ts X
src > app > TS app.module.ts > AppModule
8  import { MydirectiveDirective } from './mydirective.directive';
9  import { RegformComponent } from './regform/regform.component';
10 import { FormsModule } from '@angular/forms';
11
12 @NgModule({
13   declarations: [
14     AppComponent,
15     CitipipePipe,
16     CustompipePipe,
17     MydirectiveDirective,
18     RegformComponent
19   ],
20   imports: [
21     BrowserModule,
22     AppRoutingModule,
23     FormsModule
24   ],
25   providers: [],
26   bootstrap: [AppComponent]
27 })
```

Step 3. Now, open regForm.cmpoent.html file.

And create form with form tag of html.

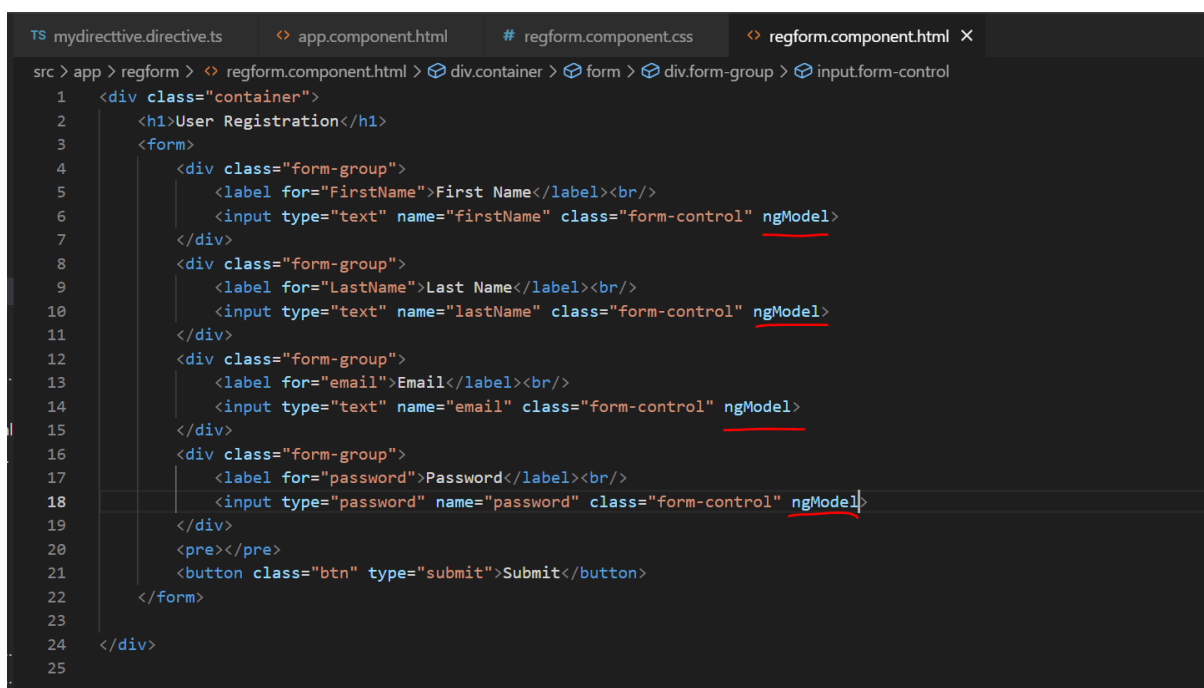
```
TS mydirective.directive.ts X  app.component.html  # regform.component.css  regform.component.html X
src > app > regform > regform.component.html > div.container > form > button.btn
1  <div class="container">
2    <h1>User Registration</h1>
3    <form>
4      <div class="form-group">
5        <label for="FirstName">First Name</label><br/>
6        <input type="text" name="firstName" class="form-control">
7      </div>
8      <div class="form-group">
9        <label for="LastName">Last Name</label><br/>
10       <input type="text" name="lastName" class="form-control">
11     </div>
12     <div class="form-group">
13       <label for="email">Email</label><br/>
14       <input type="text" name="email" class="form-control">
15     </div>
16     <div class="form-group">
17       <label for="password">Password</label><br/>
18       <input type="password" name="password" class="form-control">
19     </div>
20     <pre></pre>
21     <button class="btn" type="submit">Submit</button>
22   </form>
23
24 </div>
25
```

Open **regform.component.css** file and set

div{ text-align: center}

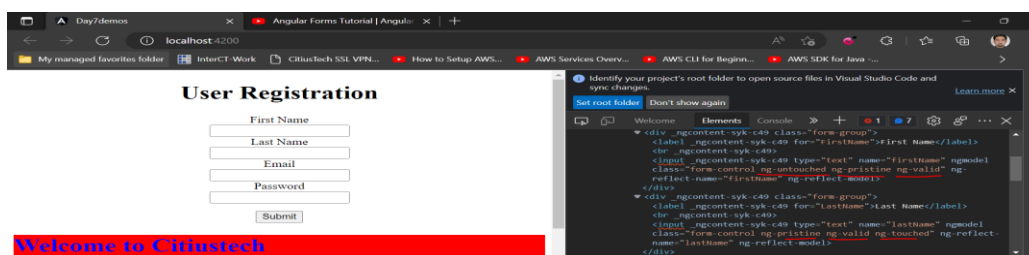
Step 4: Now, pen **regform.component.html** and we will have to add directive **ngModel**. We will have to add this directive to each input tag so it will provide form control to every field.

So, let's add it to all input fields.



```
src > app > regform > regform.component.html > div.container > form > div.form-group > input.form-control
1  <div class="container">
2    <h1>User Registration</h1>
3    <form>
4      <div class="form-group">
5        <label for="FirstName">First Name</label><br/>
6        <input type="text" name="firstName" class="form-control" ngModel>
7      </div>
8      <div class="form-group">
9        <label for="LastName">Last Name</label><br/>
10       <input type="text" name="lastName" class="form-control" ngModel>
11     </div>
12     <div class="form-group">
13       <label for="email">Email</label><br/>
14       <input type="text" name="email" class="form-control" ngModel>
15     </div>
16     <div class="form-group">
17       <label for="password">Password</label><br/>
18       <input type="password" name="password" class="form-control" ngModel>
19     </div>
20     <pre></pre>
21     <button class="btn" type="submit">Submit</button>
22   </form>
23
24 </div>
25
```

To observe, what is happening in background by adding **ngModel** directive right click in browser and click on inspect.



Here, we can see different types of classes like ng-pristine,ng-touched,ng-valid.

This indicates that after adding ngModel directive, angular recognize fields in form and add respective classes.

For this information about fields, we can implement validation.

Now, our next task is to provide some validation on submit. We will use html attributes to provide validation and by using objects on ngModel class we will specify respective message.

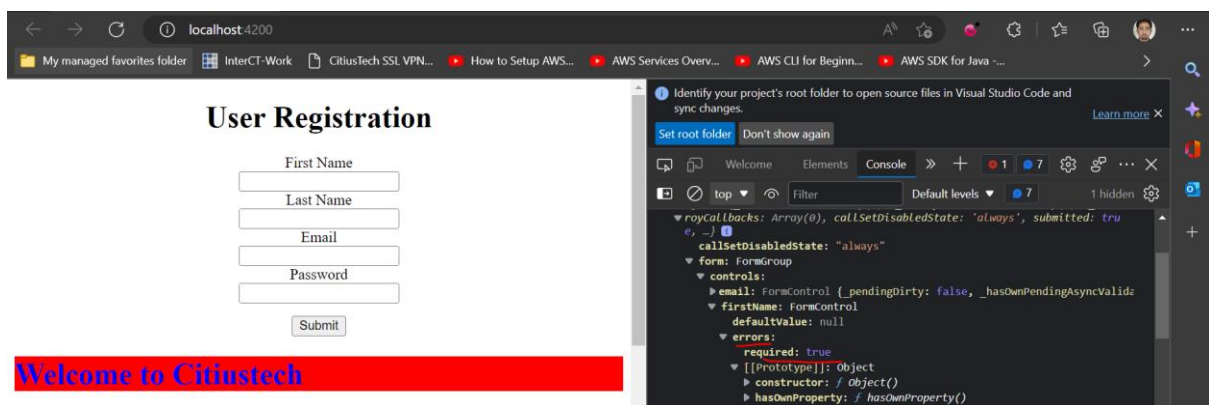
Consider a case:

Firstname field should must require input, minimum length of this field should be 3 and max will be 8. By keeping this in mind, we will add validation.

Observe given below code.

```
TS mydirective.directive.ts  app.component.html  # regform.component.css  regform.component.html X TS regform.component.ts
src > app > regform > regform.component.html > div.container > form > div.form-group > input.form-control
1 <div class="container">
2   <h1>User Registration</h1>
3   <form #login="ngForm" (ngSubmit)="submit(login)">
4     <div class="form-group">
5       <label for="FirstName">First Name</label><br/>
6       <input required minlength="3" maxlength="8" type="text" name="firstName" class="form-control" ngModel>
7     </div>
8     <div class="form-group">
9       <label for="LastName">Last Name</label><br/>
10      <input type="text" name="lastName" class="form-control" ngModel>
11    </div>
12    <div class="form-group">
13      <label for="email">Email</label><br/>
14      <input type="text" name="email" class="form-control" ngModel>
15    </div>
16    <div class="form-group">
17      <label for="password">Password</label><br/>
18      <input type="password" name="password" class="form-control" ngModel>
19    </div>
20    <pre></pre>
21    <button class="btn" type="submit">Submit</button>
22  </form>
23
24 </div>
25
```

Switch back to browser and inspect it now, it will show you invalid is true because we have specified required attribute so, this field should have some value.



So invalid is showing true. We can use these properties to raise validation.

src > app > regform > regform.component.html > div.container > form > div.form-group > div.alert

```
1 <div class="container">
2   <h1>User Registration</h1>
3   <form #login="ngForm" (ngSubmit)="submit(login)">
4     <div class="form-group">
5       <label for="FirstName">First Name</label><br/>
6       <input #name="ngModel" required minlength="3" maxlength="8" type="text" name="firstName" class="form-cont
7       <div *ngIf="name.touched && name.invalid" class="alert">
8         Please provide input in proper format
9     </div>
10  </div>
11  <div class="form-group">
12    <label for="LastName">Last Name</label><br/>
13    <input type="text" name="lastName" class="form-control" ngModel>
14  </div>
15  <div class="form-group">
16    <label for="email">Email</label><br/>
17    <input type="text" name="email" class="form-control" ngModel>
18  </div>
19  <div class="form-group">
20    <label for="password">Password</label><br/>
21    <input type="password" name="password" class="form-control" ngModel>
22  </div>
23  <pre></pre>
24  <button class="btn" type="submit">Submit</button>
25 </form>
26
```

