

ANGULARJS2

In today's web application development world, AngularJS is one of the most common and famous JS based frameworks. AngularJS has been very popular from the beginning for any kind of single page application development or any other web application. A few months ago, Google (AngularJS framework maintainers) released the new version of AngularJS i.e. AngularJS 2.0. Developers like us are eagerly awaiting the release of this final version of AngularJS.

Now, in this article and also in the coming sequential articles, I will discuss briefly about AngularJS 2. Before starting development in Angular JS, we first need to understand what is new in AngularJS 2.0 and what are new benefits of it.

What is Angular ?

The most common question before starting AngularJS i.e. what is AngularJS? In common word, AngularJS is

- A MVC Structured Framework
- A Single Page Application Framework
- A Client Side Templating
- A Language where written code can be easily tested by unit testing

Now the above key features of Angularjs contain main concepts of innovations of angularjs. The main idea of angularjs is the separation between html manipulation and javascript logic in the web pages, so that html page and javascript logic can be developed simultaneously. It always give us faster productivity from a developer point of view. Also angularjs provides us a structured javascript framework which can perform unit testing easily.

Why Angular JS 2.0 ?

Now before going to start the development in angular js 2.0, we first need to know why Google developed a new version of angularjs which is basically entirely different in structural concept or coding concept from its previous versions i.e. angular 1.x. The main reasons are -

- AngularJS 1.x is not so powerful in respect to the mobile application development. The new version of angularjs is totally focused on mobile development. Google takes care of some new things like performance, load time in respect of the mobile development.
- The earlier version of AngularJS is the modular based framework. Whereas in AngularJS 2.0 several modules have been removed resulting in better performance. Actually AngularJS 2.0 is a component based framework. In the current framework, controllers and \$scope have been removed. This two main building blocks of earlier versions have been replaced by components and directives.
- Angular2 targets the ECMAScript6 or ES6 and all the new versions of browsers so that applications developed by using this framework can be operated any browser of any device. In the previous versions of the ECMAScript, everything is defined as prototype. But in ES6 version, class can be defined as javascript objects. Also in ES6, we can implement inheritance of class objects just like OOPS.
- Angular 2.0 is basically based on TypeScript which is actually an extension of ECMAScript6.0. Basically Typescript is from Microsoft which means that in the near future AngularJS 2.0 will also be popular to the .NET domain users or developers. Normally Typescript files are compiled in the javascript files.

Different Between Angular 1.x and Angular 2.0

Angular 1.x	Angular 2.0
Structured MVC Framework	Component Based Framework
Separation of HTML and Business Logic in Javascript	It contains the same concept with more modular pattern.
	It support typescript

Below is the main key differences between Angular JS 1.x and Angular 2.0.

So in the above discussion, it is clear that for developing angular js 2.0 application or code, we need to write code in the Typescript language. So before going to write down the code, I will discuss about some basic ideas of Type Script. Actually Typescript is a superset of javascript. It means typescript is such a language which contains all the features of javascript including some extra new features which help us write down the code in a very easy way. The main features of Typescript are as below –

- It is a strongly type language
- It supports modules and classes
- It supports Template string
- It supports interfaces
- It supports generics

Some Sample Annotation of TypeScripts

JavaScript

```
var num = 5;  
var name = "Speros";  
var something = 123;  
var list = [1,2,3];
```

```
function square(num) {  
    return num * num;  
}
```

TypeScript

```
var num: number = 5;  
var name: string = "Speros"  
var something: any = 123;  
var list: Array<number> = [1,2,3];
```

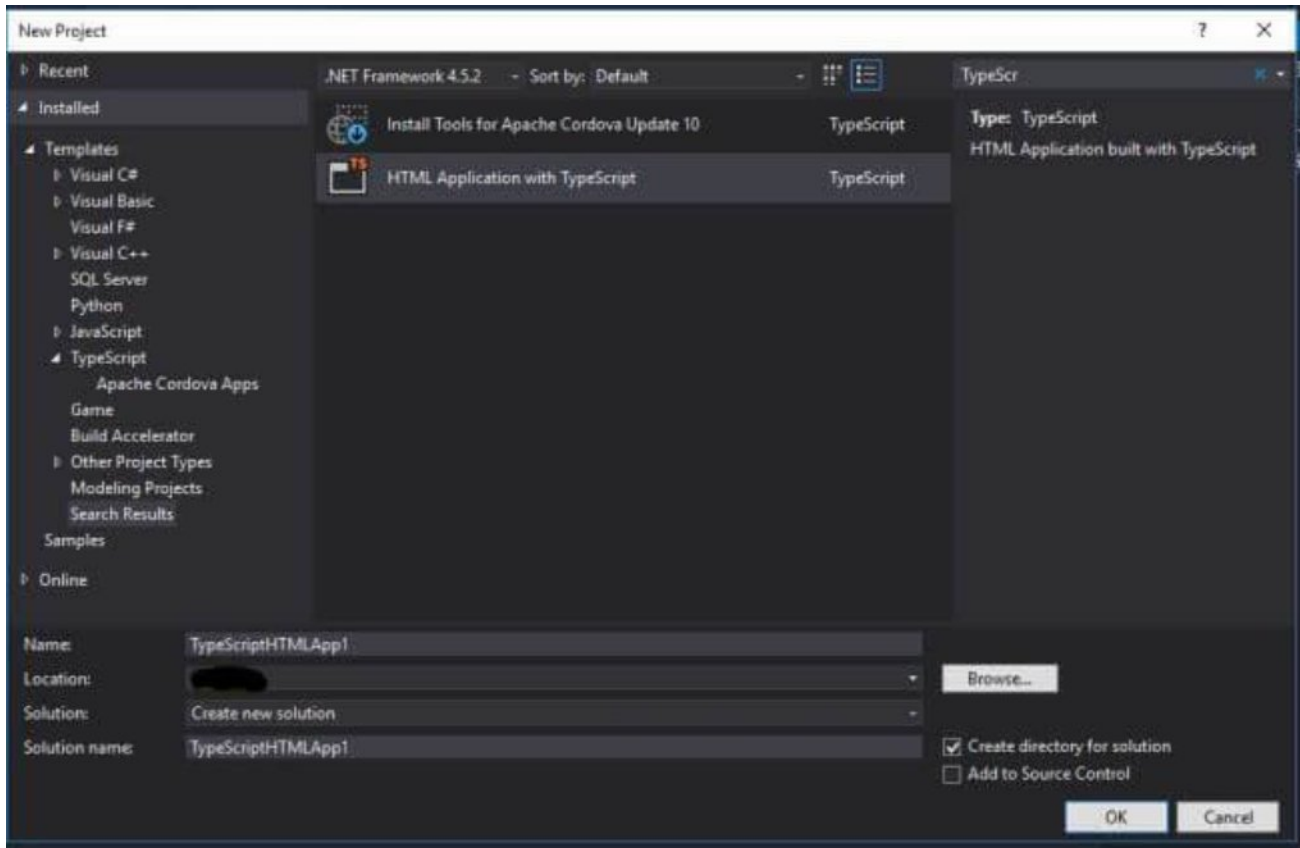
```
function square(num: number): number {  
    return num * num;  
}
```

Now, we are going to develop our first program in AngularJs2. For this, we first need to install NodeJS in our computer. The latest version of NodeJS can be downloaded from the this [URL](#).

Now open the Visual Studio 2015 and do the following steps.

Click on File -> New -> Projects and in the New Project window, Search TypeScript Project as mentioned in the below image.

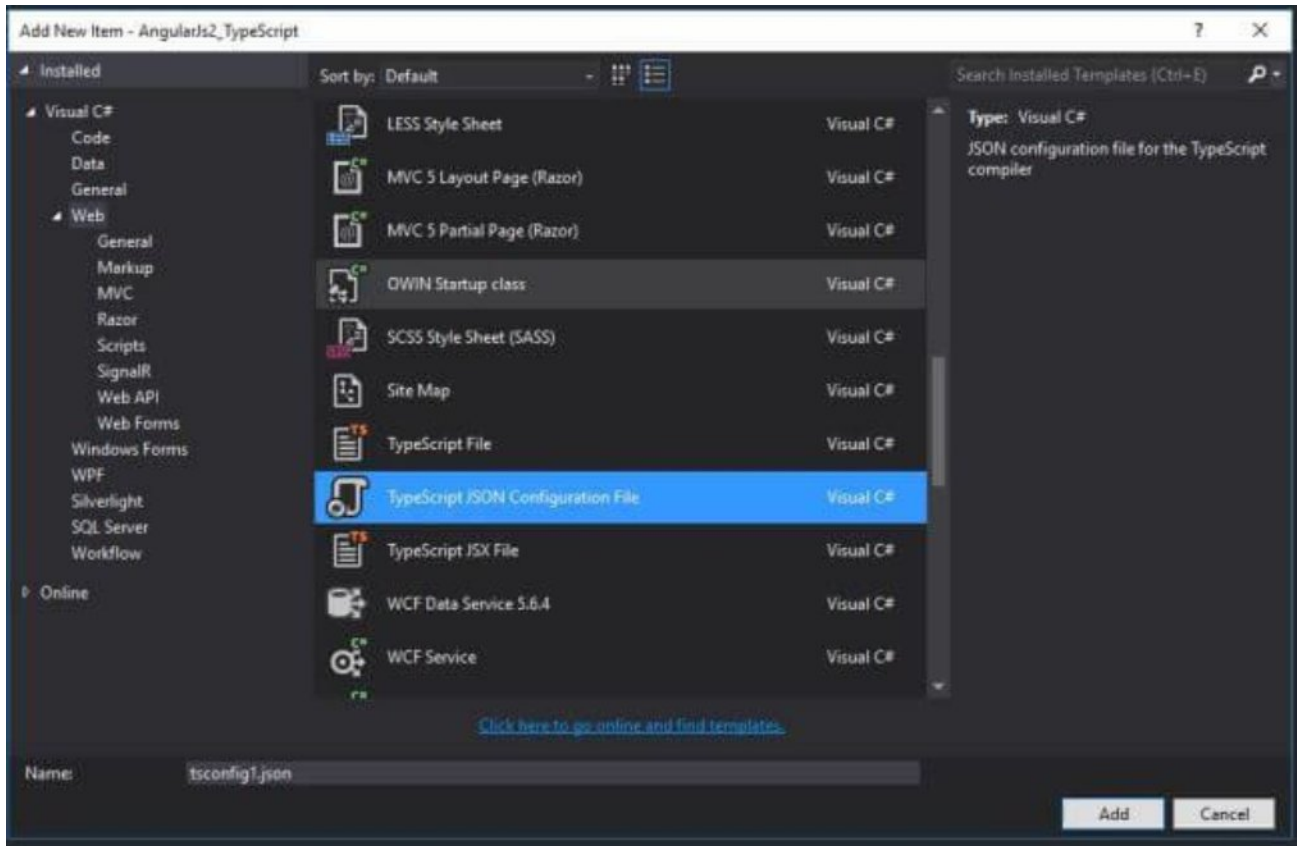
ANGULARJS2



A blank project file has been created.

Click on Project -> Add File and Select file Type TypeScript JSON Configuration File and then add the below code within the file -

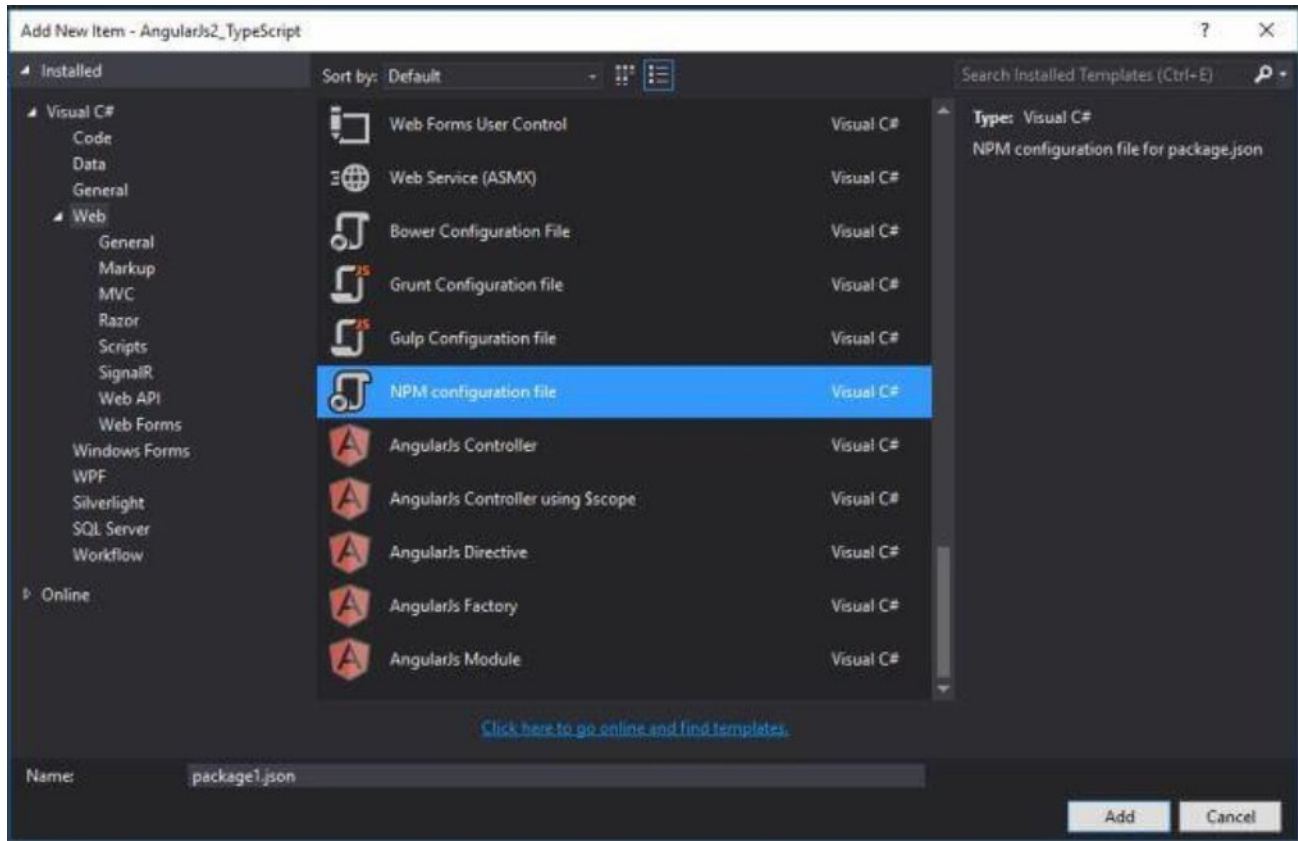
ANGULARJS2



```
1. {  
2.   "compilerOptions": {  
3.     "target": "es5",  
4.     "module": "commonjs",  
5.     "moduleResolution": "node",  
6.     "sourceMap": true,  
7.     "emitDecoratorMetadata": true,  
8.     "experimentalDecorators": true,  
9.     "removeComments": false,  
10.    "noImplicitAny": false  
11.  }  
12. }
```

Click on Project --> Add New Item and Select file Type NPM Configuration File and the below code in the file.

ANGULARJS2



```
1. {
2.   "name": "AngularJs2_TypeScript",
3.   "version": "1.0.0",
4.   "scripts": {
5.     "start": "tsc && concurrently \"tsc -w\" \"lite-server\" ",
6.     "lite": "lite-server",
7.     "postinstall": "typings install",
8.     "tsc": "tsc",
9.     "tsc:w": "tsc -w",
10.    "typings": "typings"
11.  },
12.  "license": [
13.    {
14.      "type": "MIT",
15.      "url": "https://github.com/angular/angular.io/blob/master/LICENSE"
16.    }
17.  ],
18.  "dependencies": {
19.    "@angular/common": "~2.1.0",
20.    "@angular/compiler": "~2.1.0",
21.    "@angular/core": "~2.1.0",
22.    "@angular/forms": "~2.1.0",
23.    "@angular/http": "~2.1.0",
24.    "@angular/platform-browser": "~2.1.0",
25.    "@angular/platform-browser-dynamic": "~2.1.0",
26.    "@angular/router": "~3.1.0",
27.    "@angular/upgrade": "~2.1.0",
28.    "angular-in-memory-web-api": "~0.1.5",
29.    "bootstrap": "^3.3.7",
30.    "core-js": "^2.4.1",
31.    "reflect-metadata": "^0.1.8",
32.    "rxjs": "5.0.0-beta.12",
33.    "systemjs": "0.19.39",
34.    "zone.js": "^0.6.25"
35.  },
}
```

ANGULARJS2

```
36.   "devDependencies": {
37.     "concurrently": "^3.0.0",
38.     "lite-server": "^2.2.2",
39.     "typescript": "^2.0.3",
40.     "typings": "^1.4.0"
41.   }
42. }
```

Click on Project -> Add New Item and Select file Type Javascript File with file name systemjs.config.js and the below code.

```
1.  /**
2.   * System configuration for Angular samples
3.   * Adjust as necessary for your application needs.
4.   */
5.  (function (global) {
6.    System.config({
7.      paths: {
8.        // paths serve as alias
9.        'npm:': './node_modules/'
10.     },
11.     // map tells the System loader where to look for things
12.     map: {
13.       // our output js is within the src folder mention folder name if all js file located in a
14.       // particular file
15.       app: '.',
16.       // angular bundles
17.       '@angular/core': 'npm:@angular/core/bundles/core.umd.js',
18.       '@angular/common': 'npm:@angular/common/bundles/common.umd.js',
19.       '@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',
20.       '@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-browser.umd.js',
21.       '@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-dynamic/bundles/platform-browser-dynamic.umd.js',
22.       '@angular/http': 'npm:@angular/http/bundles/http.umd.js',
23.       '@angular/router': 'npm:@angular/router/bundles/router.umd.js',
24.       '@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',
25.       // other libraries
26.       'rxjs': 'npm:rxjs',
27.       'angular-in-memory-web-api': 'npm:angular-in-memory-web-api',
28.     },
29.     // packages tells the System loader how to load when no filename and/or no extension
30.     packages: {
31.       app: {
32.         main: './main.js',
33.         defaultExtension: 'js'
34.       },
35.       rxjs: {
36.         defaultExtension: 'js'
37.       },
38.       'angular-in-memory-web-api': {
39.         main: './index.js',
40.         defaultExtension: 'js'
41.       }
42.     }
43.   })(this);
```

1. Now open the package.json file location in command prompt and execute the below command to load the required modules and supported files which are mentioned in the package.json file.

npm start

ANGULARJS2

Now add a TypeScript File named main.ts and the below code.

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
3. import { AppModule } from './app.module';
4.
5. const platform = platformBrowserDynamic();
6. platform.bootstrapModule(AppModule);
```

Here Import keyword is used to include the browser module into the program.

Now add another typescript file named app.module.ts.

```
1. import { NgModule } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { FirstProgComponent } from './src/app.component.firstprog';
4.
5. @NgModule({
6.   imports: [BrowserModule],
7.   declarations: [FirstProgComponent],
8.   bootstrap: [FirstProgComponent]
9. })
10. export class AppModule { }
```

Here @NgModule is the module annotations. I will discuss in detail about this in the next article.

Now add another typescript file named app.component.ts and add the below code.

```
1. import { Component } from '@angular/core';
2.
3. @Component({
4.   selector: 'my-app',
5.   template: '<h1>My First Angular App in Angular 2.0</h1><br>Debasish Saha'
6. })
7.
8. export class FirstProgComponent { }
```

Here @Component is the component annotation which defines a component which contains a selector property. Basically selector property tells the browser which component needs to load.

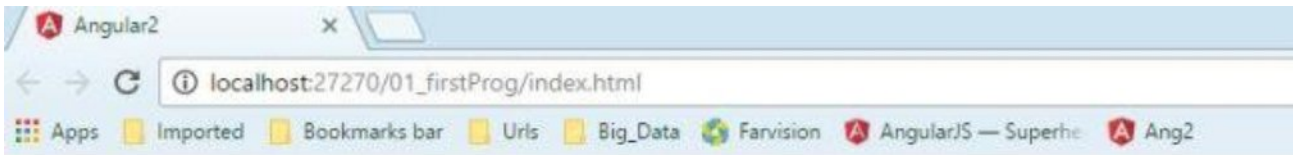
Now add html file named *index.html* and write down the below code.

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <title>Angular2</title>
5.     <meta charset="UTF-8">
6.     <meta name="viewport" content="width=device-width, initial-scale=1">
7.     <link rel="stylesheet" href="../styles.css">
8.
9.     <!-- Polyfill(s) for older browsers -->
10.    <script src="../node_modules/core-js/client/shim.min.js"></script>
11.    <script src="../node_modules/zone.js/dist/zone.js"></script>
12.    <script src="../node_modules/reflect-metadata/Reflect.js"></script>
13.    <script src="../node_modules/systemjs/dist/system.src.js"></script>
14.
15.    <script src="../systemjs.config.js"></script>
16.    <script>
17.      System.import('app').catch(function(err){ console.error(err); });
18.    </script>
```


ANGULARJS2

```
19.   </head>
20.
21.   <body>
22.     <my-app>Loading... </my-app>
23.
24.   </body>
25. </html>
```

Now build the project and run it in the browser.



My First Angular App in Angular 2.0

I am here to continue the discussion around [AngularJS 2.0](#). Today, we will discuss about Component in AngularJS 2. Also, in case you did not have a look at our previous articles of this series, go through the links, mentioned below.

- [AngularJS 2.0 From Beginning - Introduction of AngularJS 2.0](#)

In this article, we will cover the topics about component, as shown below.

1. What is component.
2. How to create a component.
3. About component function properties or argument.
4. Life Cycle of the component.
5. Nested component.

In my previous article, I discussed how to write a simple "Hello World" type program in Angular2. In that program, we developed a component class named FirstProgComponent with the `@component` decorator. Actually, component is the main building block of Angular2 Framework. In Angular2 application project, any number of components can be created and used within a single HTML file. Basically, Angular2 framework is a component based framework, which is the main difference from the previous version of AngularJS.

Actually, previous versions of AngularJS are based on the concept of controller and directives to populate the data and also responsible to develop any custom elements for the view but in Angular 2, components perform all the responsibilities that are performed by controllers and directives.

What is component ?

In Angular2, every thing is component. Component is the main building block of Angular2 framework. Basically, component is the main process in which we can define or design the views including the logic.

Steps to create a component

Although, we have already created our first component in the previous article. But in spite of that, I describe the steps for creating a component.

1. Create a typescript base class and export it. As it is very much clear from its name that export keyword is used to export any type of objects like class, functions etc. from an angular module. Export is basically just

ANGULARJS2

an identifier, which indicates that it can be import into any other script section, using import keyword or module in Angular2. Export or import both the modules are supported by ES6 module definitions.

2. Now, decorate the class with `@component` metadata decorator or annotations. Basically `@component` add some metadata to the class objects in order to provide some specific meaning. Basically, it is the declarative way to add metadata to the code.

Now, the first questions arise that where the `@component` is defined? As we all know, it is not a JavaScript keyword or method. The answer is that this keyword is provided by Angular2 framework itself. Basically Angular2 framework provides us some inbuilt module or provider for the perform our task. For use `@component` decorator or annotation in our class, we need to import the code module into our class file at the beginning.

```
import {component} from '@angular/core'
```

3. Import the required library or module to develop the component.
4. Now, include the component class within the `ngModule` in order it to be used by other component or Applications. To mention that component is a member of `ngModule`, you need to add the component name within the declaration field of `ngModule`.

Component Configuration

`@Component` decorator basically decorated a type script class as a component objects. Basically, it is a function, which takes different types of parameters. In the `@component` decorator, we can set the values of different properties to finalize the behavior of the components. The most used properties are shown below.

1. *selector*
A component can be used by the selector expression.
2. *template*
Basically, template is the part of the component, which is rendered in the Browser. We can directly define HTML tags or code within the template properties. Sometimes we called this as an inline template. For writing multiple lines of HTML code, all code need to be covered within tilt (```) symbol.
3. *templateUrl*
Another way of rendered HTML tags in the Browser. Here, we need to provide the HTML file name with its related file path. Some times, it is known as external template. It is better approach, if HTML is a part of the component is complex.
4. *moduleId*
It is used to resolve the related path of the template URL or style URL for the component objects.
5. *styles or styleUrls*
Component can use its own style by providing custom CSS or can refer to an external style sheet files, which can be used by multiple components at a time. For proving inline style, we need to use `styles` and to provide external file path or URL, we need to use `styleUrl`'s.
6. *providers*
We are in the real life Application. We need to use or inject different types of custom Service within the component to implement the business logic for the component. To use any custom Service within the component, we need to provide the Service instance within the provider. Basically, provider is an array type property, where multiple Service instance name can be provided by comma separation.

Life Cycles of the Component

The life cycle of the component is maintained by Angular2 himself. Below is the list of life cycle method of Angular2 components.

ANGULARJS2

- *constructor*
This method is executed before the execution of any one life cycle method. It is basically used for inject dependency.
- *ngOnChanges*
This method is called when an input control or binding value changes.
- *ngOnInit*
This method is called after the execution of first ngOnChanges.
- *ngDoCheck*
This method is called after every execution of ngOnChanges.
- *ngAfterContentInit*
This method executes after component content is initialized.
- *ngAfterContentChecked*
This method is executed after every check of the component check.
- *ngAfterViewInit*
This method executes after component views initialize.
- *ngAfterViewChecked*
This method executes after every check of component views.
- *ngOnDestroy*
This method executes before the component destroys.

Now, to create a component, first create a typescript file named app.component.template.ts and add the code, mentioned within it.

```
1. import {
2.     Component
3. } from '@angular/core';
4. @Component({
5.     moduleId: module.id,
6.     selector: 'hello-world',
7.     templateUrl: 'app.component.template.html',
8.     styles: ['h1{color:red}']
9. })
10. export class TemplateComponent {
11.     message: string = "Angular 2 Component with Template";
12.     author: string = "Debasis Saha";
13.     constructor() {}
14. }
```

In the code, mentioned above, I use template URL and style to show the text in different color. Now, for template HTML file, add HTML file named app.component.template.HTML and add the code, mentioned below.

```
1. <div>
2.     <h1>{{message}}</h1> <br />
3.     <h2>{{author}}</h2>
4. </div>
```

Now, run the code and the output is mentioned below.

Angular 2 Component with Template

Debasis Saha

In the example, mentioned above, I set the color of h1 tag as Red. Thus, when I run the code in the Browser, it will effect automatically. The same thing can be done by providing the style code within a CSS file and provide the file name with the relative file path to import the style class into the document.

Nested Component

Just like previous version of AngularJS, Angular2 also allows us to create nested component or component inside a component.

Let's see how it works. I first create a component called parentcomponent. Now, I also create two separate components named childComponent and EmailBoxComponent. Now, I want to use these two components within the parent component template. For doing this, I need to do the following.

- Import childcomponent and EmailBoxComponent within ngModule.
- Include the component name within declaration property of the ngmodule.
- Use child component, using their selector in the parent component template.

To demonstrate the nested component, I first create two child components. One of them simply displays some fixed text message within h2 tag. The another child component will be a simple search button with textbox. For doing this, we need to add one TypeScript file named as "app.component.child.ts" and write down the code, mentioned below.

```
1. import {
2.     Component
3. } from '@angular/core';
4. @Component({
5.     selector: 'child',
6.     template: '<h2>{{message}}</h2>'
7. })
8. export class ChildComponent {
9.     message: string = "This is Child Component";
10.    constructor() {}
11. }
```

Now, add another TypeScript file with name "app.component.emailbox.ts" and write down the code, mentioned below.

```
1. import {
2.     Component
3. } from '@angular/core';
4. @Component({
5.     selector: 'email-box',
6.     template: '<input placeholder="Email I" /><button class="btn-clear">Register</button>'
7. })
8. export class EmailBoxComponent {}
```

Now, create another TypeScript file named as "app.component.parent.ts" and add the code, mentioned below.

```
1. //components files
2. import {
3.     Component,
```

ANGULARJS2

```
4.     OnInit
5.   } from '@angular/core';
6.   //
7.   @Component({
8.     moduleId: module.id,
9.     selector: 'my-app',
10.    templateUrl: 'app.component.parent.html',
11.  })
12.  export class ParentComponent implements OnInit {
13.    constructor() {}
14.    ngOnInit() {}
15.  }
```

Now, create another HTML file named as "app.component.parent.html" and add the code, mentioned below.

```
1.  <div>
2.    <h1>This is Main Component</h1> <br />
3.    <child></child>
4.    <child></child>
5.    <child></child> <br />
6.    <email-box></email-box>
7.  </div>
```

Now, add another TypeScript file for module with name "app.module.ts" and write down the code, mentioned below.

```
1.
2.   import {
3.     NgModule
4.   } from '@angular/core';
5.   import {
6.     BrowserModule
7.   } from '@angular/platform-browser';
8.   import {
9.     ParentComponent
10.  } from './src/app.component.parent';
11.  import {
12.    ChildComponent
13.  } from './src/app.component.child';
14.  import {
15.    EmailBoxComponent
16.  } from './src/app.component.emailbox';
17.  @NgModule({
18.    imports: [BrowserModule],
19.    declarations: [ParentComponent, ChildComponent, EmailBoxComponent],
20.    bootstrap: [ParentComponent]
21.  })
22.  export class AppModule {}
```

Output

This is Main Component

This is Child Component

This is Child Component

This is Child Component

I am here to continue the discussion around [AngularJS 2.0](#). Today, we will discuss Data Binding in AngularJS 2. Also, in case you did not have a look at our previous articles of this series, go through the links, mentioned below.

- [AngularJS 2.0 From Beginning - Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning - Component \(Day 2\)](#)

In my previous article, I already discussed about the life cycle of the component. Now in this article, I will discuss about one of the main advantages of the Angular framework i.e. data binding. As per our previous experience regarding Angular 1.x series, data binding is one key feature which makes our development very easy with less coding for capturing or binding the data within the view interface. In Angular 1.x, data binding provides us two different mechanisms for binding the data; i.e., one-way binding and two-way binding.

Now, in Angular 2.0, there are couple of ways we can use for the data binding concept. They are,

1. Interpolation
2. Property Binding
3. Two Way Binding
4. Event Binding

Interpolation

Interpolation data binding is the most popular and easiest way of data binding in AngularJS. This mechanism is also available in earlier versions of the Angular framework. Actually, context between the braces is the template expression that AngularJS first evaluates and then converts into strings. Interpolation uses the braces expression i.e. `{{ }}` for rendering the bound value to the component template. It can be a static string or numeric value or an object of your data model.

As per our example, in AngularJS, we use it like below.

```
{{model.firstName}}
```

Here, model is the instance of the Controller objects. But in Angular 2, it is now much simpler where Controller instance name is removed from the expression, i.e.

```
{{firstName}}
```

ANGULARJS2

Property Binding

In AngularJS 2.0, a new binding mechanism is introduced, called Property Binding. But actually in nature, it is just same as interpolation or one-way binding. Some people call this process one way binding just like in the previous AngularJS concept. Property binding uses [] to send the data from the component to the HTML template. The most common way to use property binding is to assign any property of the HTML element tag into the [], with component property value, i.e.

```
<input type="text" [value]="data.name"/>
```

In the previous version of AngularJS, it could be done by using ng-bind directives. Basically, property binding can be used to bind any property, component, or a directive property. It can be used as

- Attribute binding
- Class binding
- Style binding

Now, for demonstrating the interpolation and one-way binding, we first create an interpolation component. For doing this, we need to do the following.

Step 1

Create one TypeScript file named as *app.component.interpolation.ts* and add the below code.

```
1. import { Component } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'interpolation',
6.   templateUrl: 'app.component.interpolation.html'
7. })
8.
9. export class InterpolationComponent {
10.   value1: number = 10;
11.   array1: Array<number> = [10, 22, 14];
12.   dt1: Date = new Date();
13.
14.   status: boolean = true;
15.
16.   returnString() {
17.     return "String return from function";
18.   }
19. }
```

Step 2

Now, add another HTML file named *app.component.interpolation.html* and add the below code.

```
1. <div>
2.   <span>Current Number is {{value1}}</span>
3.   <input [value]="value1" />
4.   <span>Status is {{status}}</span>
5.   <br /><br />
6.   <span>{{status ? "This is correct status" : "This is false status"}}</span>
7.   <br /><br />
8.   <span>{{returnString()}}</span>
9. </div>
```

ANGULARJS2

Step 3

Create another TypeScript file named as app.component.parent.ts and add the below code.

```
1. import { Component, OnInit } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'my-app',
6.   templateUrl: 'app.component.parent.html',
7. })
8.
9. export class ParentComponent implements OnInit {
10.   constructor() { }
11.
12.   ngOnInit() { }
13. }
```

Step 4

Again, create an HTML file named app.component.parent.html and add the below code.

```
1. <div>
2.   <h2>Demonstration of Interpolation</h2>
3.   <interpolation></interpolation>
4. </div>
```

Step 5

Lastly, create another TypeScript file named app.module.ts and add the below code.

```
1. import { NgModule } from '@angular/core';
2. import { FormsModule } from '@angular/forms';
3. import { BrowserModule } from '@angular/platform-browser';
4. import { ParentComponent } from './src/app.component.parent';
5. import { InterpolationComponent } from './src/app.component.interpolation';
6.
7. @NgModule({
8.   imports: [BrowserModule, FormsModule],
9.   declarations: [ParentComponent, InterpolationComponent],
10.  bootstrap: [ParentComponent]
11. })
12. export class AppModule { }
```

Now, run the code. The output is shown below.

Demonstration of Interpolation

Current Number is 10

Status is true

This is correct status

String return from function

ANGULARJS2

Event Binding

Event Binding is one of the new mechanisms introduced by AngularJS 2.0 as a new concept. In previous versions of AngularJS, we always used different types of directives like ng-click, ng-blur, for binding any particular event action of an HTML control. But in Angular 2, we need to use the same property of the HTML element (like click, change etc.) and use it within parentheses. So, in Angular 2, with properties, we use square brackets and in events, we use parentheses.

```
<button (click)="showAlert();">Click</button>
```

For doing this, add another TypeScript file named app.component.eventbinding.ts and add the below code.

```
1. import { Component } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'event-bind',
6.   templateUrl: 'app.component.eventbinding.html'
7. })
8.
9. export class EventBindingComponent {
10.  showAlert() {
11.    console.log('You clicked on the button...');
12.    alert("Click Event Fired...");
13.  }
14. }
```

Now, add another HTML file named app.component.eventbinding.html and add the below code.

```
1. <div>
2.   <input type="button" value="Click" class="btn-block" (click)="showAlert()" />
3.   <br /><br />
4.   <input type="button" value="Mouse Enter" class="btn-block" (mouseenter)="showAlert()" />
5. </div>
```

Place the <event-bind> selector in the parent HTML and add the component name in the module. Now, run the code. The output is shown below -

Demonstration of Event Binding

Click

Mouse Enter

localhost:27270 says:

Click Event Fired...

OK

Two Way Binding

ANGULARJS2

The most popular and widely used data binding mechanism is two-way binding in the Angular framework. Basically, two-way binding is used in the input type field or any form elements where user type or change of any Control value is done in the one side, and on the other side, the same is automatically updated into the Controller variables and vice versa. In Angular 1.x, we used `ng-model` directives with the element for this purpose.

```
<input type="text" ng-model="firstName"/>
```

Similarly, in Angular 2.0, we have a directive called `ngModel` and it can be used as below.

```
<input type="text" [(ngModel)]="firstName"/>
```

As we see, in Angular 2, the syntax is different because we use `[]` - since it is actually a property binding and parentheses are used for the event binding concept.

What is ngModel

As I already said in the previous section of the article, `ngModel` basically performs both property binding as well as event binding. Actually, property binding of `ngModel` (i.e. `[(ngModel)]`) performs the activity to update the input element with value, whereas `(ngModel)` (basically, fired in the `(ngModelChange)` event) instructs the outside world when any changes have occurred in the DOM Element.

For demonstrating two-way binding, add another TypeScript file named `app.component.twowaybind.ts` and add the below code.

```
1. import { Component } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'twowaybind',
6.   templateUrl: 'app.component.twowaybind.html'
7. })
8.
9. export class TwoWayBindComponent {
10.   val: string = '';
11. }
```

Now add another HTML file named `app.component.twowaybind.html` and add the below code.

```
1. <div>
2.   <input [(ngModel)]="val" type="text"/>
3.   <span>{{val}}</span>
4.   <br />
5.   <input [value]="val" (keyup)="num = $event.target.value" />
6. </div>
```

Now, place the `<twowaybind>` selector in the parent HTML and add the component name in the module. And then, run the code. The output is as below.

Demonstration of Two Way Binding

Welcome to Angular 2	Welcome to Angular 2
Welcome to Angular 2	

I am here to continue the discussion around [AngularJS 2.0](#). Today, we will discuss input data binding in AngularJS 2. Also, in case you did not have a look at the previous articles of this series, go through the links, mentioned below.

ANGULARJS2

- [AngularJS 2.0 From Beginning - Introduction of AngularJS 2.0](#)
- [AngularJS 2.0 From Beginning - Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning - Data Binding \(Day 3\)](#)

In my previous article, I already discussed about the different processes of data binding techniques in Angular 2.0. Now, in this article, I will discuss about the process through which we can pass information or values within a component. Since in Angular 2.0, components are the core objects of the Angular framework, it is very common concern for all of us that how we can pass the data into the component in order for us to dynamically manipulate or configure the component. In the world of web development, static components will not fulfill all our requirement. So for that reason, we need to develop dynamic component or non-static component which can take data from its parent component.

Component Hierarchy

Angular 2.0 is actually made of a nested or hierarchical component structure – that means Angular 2.0 basically begins with a root component which contains all the child components of the application. Components act as self-contained because normally, we want to encapsulate our component variables and functions; and we never want that the other code to arbitrary reaches our code or value to check the changes, done in the component. Also, as a developer, I always want that our component should not hamper any other component objects used in other areas or along with my components. At the same time, component may need to share data. In Angular 2.0, a component can receive any type of data from its parent component as a receiving component.

In the above structure, each cycle structure represents a component and technically, this presentation is called as graph – a data structure, consisting of nodes and its connecting edges. The line structure represents the data flow from one component and as we can see, the data flow is in one direction i.e. from top to downwards (means from parent component to child components). This kind of structure always has some important advantages :- i.e. it is very much predictable and also it is simple to traverse and it is very easy to understand what has happened when a change is made in the parent component. In Angular point of view, when data changes in the parent node, it is very easy to find what changes are occurred in the child component.

For implementing this concept, Angular 2.0 introduces a property of *@component* decorator, to pass the data from parent component to child components. In Angular 2, we also can specify the data type of input data so that parent component can always pass the proper data to the child component. For defining input property of a component, we need to remember the following points.

1. The input property is passed to the *@component* decorator to pass the data.
2. The component displays the input property in the view, but when we want to access the input property within the constructor of the constructor, then we found that it is not defined yet. This is because input properties are not available until the view has rendered which happened after the constructor function execution. Basically the values of input properties can be found either within *ngOnInit()* or *ngAfterViewInit()*.

To declare input property, we need to first import the input command from Angular core module and then, use *@Input* decorator for declaring that property. *@Input* decorator takes a string type argument for specify the custom property name to provide custom component attribute name.

For demonstrating this input, we basically create an email register component as a child component and then place that component within a parent component page to pass input values. For this, create the below mentioned files.

Step 1

Create a TypeScript file named *app.component.emailbox.ts* and write down the below code.

```
1. import { Component, Input } from '@angular/core';
2.
3. @Component({
4.   selector: 'email-box',
5.   template: '<input placeholder="{{email}}"/><button class="btn-clear">Register</button>'
6. })
```

ANGULARJS2

```
7.
8. export class EmailBoxComponent {
9.
10.     @Input('place-holder')
11.     email: string = "Type your email..";
12. }
```

Step 2

Now, create another TypeScript file called app.component.parent.ts and write down the below code.

```
1. import { Component, OnInit } from '@angular/core';
2.
3. @Component({
4.     moduleId: module.id,
5.     selector: 'my-app',
6.     templateUrl: 'app.component.parent.html',
7. })
8.
9. export class ParentComponent implements OnInit {
10.     constructor() { }
11.
12.     ngOnInit() { }
13. }
```

Step 3

Now, create an HTML file named app.component.parent.html and write down the below code.

```
1. <div>
2.     <h1 class="badge">For subscription, register you email address :</h1>
3.     <br />
4.     <br />
5.     <email-box></email-box><br />
6.     <email-box [place-holder]="'' Put your email ''"></email-box><br />
7.     <email-box place-holder="Put your office email"></email-box><br />
8. </div>
```

Here, place-holder basically is the input property which takes the place holder value of the text box.

Step 4

Now, create another TypeScript file named app.module.ts and write down the below code.

```
1. import { NgModule } from '@angular/core';
2. import { FormsModule } from '@angular/forms';
3. import { BrowserModule } from '@angular/platform-browser';
4.
5. import { ParentComponent } from './src/app.component.parent';
6. import { EmailBoxComponent } from './src/app.component.emailbox';
7.
8. @NgModule({
9.     imports: [BrowserModule, FormsModule],
10.    declarations: [ParentComponent, EmailBoxComponent],
11.    bootstrap: [ParentComponent]
12. })
13. export class AppModule { }
```

ANGULARJS2

Step 5

Now, create another TypeScript file named `main.ts` and write down the below code.

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
3. import { AppModule } from './app.module';
4.
5. const platform = platformBrowserDynamic();
6. platform.bootstrapModule(AppModule);
```

Step 6

Now, create another HTML file named `index.html` and write down the below code.

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <title>Angular2</title>
5.     <meta charset="UTF-8">
6.     <meta name="viewport" content="width=device-width, initial-scale=1">
7.     <link rel="stylesheet" href="../styles.css">
8.
9.     <!-- Polyfills for older browsers -->
10.    <script src="../node_modules/core-js/client/shim.min.js"></script>
11.    <script src="../node_modules/zone.js/dist/zone.js"></script>
12.    <script src="../node_modules/reflect-metadata/Reflect.js"></script>
13.    <script src="../node_modules/systemjs/dist/system.src.js"></script>
14.
15.    <script src="../systemjs.config.js"></script>
16.    <script>
17.      System.import('app').catch(function(err){ console.error(err); });
18.    </script>
19.  </head>
20.
21.  <body>
22.    <my-app>Loading... </my-app>
23.  </body>
24. </html>
```

Now, run the `index.html` file in the browser and see the output.

Here,

Now, we want to simply update the email register component with event handling. For this, do the following.

Step 1

First, add another TypeScript file named `app.component.emailboxevent.ts` and write down the below code.

```
1. import { Component, Input } from '@angular/core';
2.
3. @Component({
4.   selector: 'email-box-event',
5.   template: '<input placeholder="{{email}}" #emailbox/><button class="red" (click)="register(emailbox)">Register</button>',
6.   styles: ['.red {color: red;}']
7. })
8.
```

ANGULARJS2

```
9. export class EmailBoxEventComponent {
10.   @Input('place-holder')
11.   email: string = "Type your email...";
12.
13.   register(txtEmail) {
14.     console.log('Email Address resgister properly...');
15.     if (txtEmail.value == '') {
16.       alert('Put your email address first..');
17.       return;
18.     }
19.     else {
20.       alert('Your email address is : ' + txtEmail.value);
21.       txtEmail.value = '';
22.     }
23.   }
24. }
```

Step 2

Now, add another TypeScript file for another type of email register component with the name *app.component.emailregister.ts* and add the below code.

```
1. import { Component, Input } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'email-register',
6.   templateUrl: './app.component.emailregister.html',
7.   styleUrls: ['./style/style.css']
8. })
9.
10. export class EmailRegisterComponent {
11.   @Input('caption1') caption1: string = '';
12.   @Input() caption2: string = '';
13.
14.   name: string = '';
15.   email: string = '';
16.
17.   register(): void {
18.     if (this.name != '' && this.email != '') {
19.       alert(this.name + ' Your email address ' + this.email + ' has been register. ');
20.       this.clear();
21.     }
22.     else {
23.       alert('Fill all the fields')
24.     }
25.   }
26.
27.   clear(): void {
28.     this.name = '';
29.     this.email = '';
30.   }
31. }
```

Step 3

Now, add an HTML file named *app.component.emailregister.html* and add the below code.

```
1. <div style="width: 40%; ">
2.   <fieldset>
3.     <h2>Email Registration</h2>
4.     <table style="width: 100%">
```

ANGULARJS2

```
5.         <tr>
6.             <td><span>{{caption1}}</span></td>
7.             <td><input type="text" [(ngModel)]="name" /></td>
8.         </tr>
9.         <tr>
10.            <td><span>{{caption2}}</span></td>
11.            <td><input type="text" [(ngModel)]="email" /></td>
12.        </tr>
13.        <tr>
14.            <td><input type="button" value="Register" class="blue" (click)="register()" /></td>
15.            <td><input type="button" value="Cancel" class="red" (click)="clear()" /></td>
16.        </tr>
17.    </table>
18. </fieldset>
19. </div>
```

Step 4

Change the code of app.component.parent.html file as below.

```
1. <div>
2.     <h1 class="badge">For subscription, register you email address :</h1>
3.     <br />
4.     <br />
5.     <email-box></email-box><br />
6.     <email-box [placeholder]="'Put your email'"></email-box><br />
7.     <email-box placeholder="Put your office email"></email-box><br />
8. </div>
9.
10. <div>
11.     <br />
12.     <h2>Email Register control with Events</h2>
13.     <email-box-event [placeholder]="'Put your email'"></email-box-event><br />
14.     <hr />
15.     <br />
16.     <email-register [caption1]="'Your Name'" caption2="Email Id"></email-register>
17. </div>
```

Step 5

Change the code of app.module.ts file as below.

```
1. import { NgModule } from '@angular/core';
2. import { FormsModule } from '@angular/forms';
3. import { BrowserModule } from '@angular/platform-browser';
4.
5. import { ParentComponent } from './src/app.component.parent';
6. import { EmailBoxComponent } from './src/app.component.emailbox';
7. import { EmailBoxEventComponent } from './src/app.component.emailboxevent';
8. import { EmailRegisterComponent } from './src/app.component.emailregister';
9.
10. @NgModule({
11.     imports: [BrowserModule, FormsModule],
12.     declarations: [ParentComponent, EmailBoxComponent, EmailBoxEventComponent, EmailRegisterComponent],
13.     bootstrap: [ParentComponent]
14. })
15. export class AppModule { }
```

Now, write down the below code and the output as below.

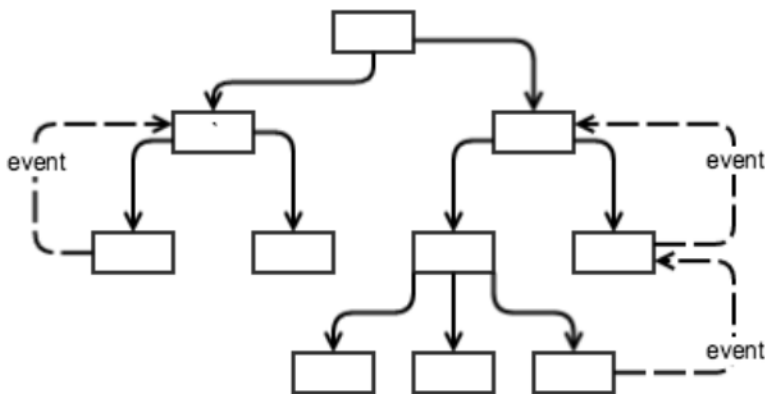
ANGULARJS2

I am here to continue the discussion around [AngularJS 2.0](#). Today, we will discuss the output property binding in AngularJS 2. Also, in case you did not have a look at the previous articles of this series, go through the links, mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)

In my previous article, I already discussed about the different processes of data binding techniques including input property binding mechanism in case of components within Angular 2.0. Now in this article, I will discuss about the process through which we can return the data to the parent component from the child component or execute any method in the parent component during any event execution of the child component. Since in Angular 2.0 the components are the core objects of the Angular framework, it is a very common concern for all of us regarding how we can pass the data or event into the parent component so that we can dynamically manipulate or configure the parent component while any changes are being done in the child component. In the world of web development, static components will not fulfill all our requirements, due to which we need to develop dynamic components or nonstatic components, which can take data from its parent component and also return some data or event to the parent component.

As I had already mentioned in my previous article, Angular 2.0 is actually made up a nested or hierarchical component structures, it means Angular 2.0 basically begins with a root component, which contains all the child components of the applications. The components act as a self contained entity because normally we want to encapsulate our component variables and functions, and we never want that other code to arbitrarily reach our code or value to check what changes can be done in the component. This architecture works fine for simply displaying data, but in the real world application, the data needs to flow both ways – i.e. back to the hierarchy – such as when the user makes changes to any data in the child component. In that case we need to tell the parent component about the changes, so the parent component updates its model and returns that data to the Server. Now, Angular 2 uses a different mechanism to send back data to the parent's events.



Now, whenever any action occurs within a child component, the child component fires an event which is traced by the parent component to intimate parent component. The parent component can perform separate actions against that event to perform its own logical operations.

If the child component wants to send the information back to its container; i.e., parent component, it can raise an event with the help of `@Output` decorator. Like the `@Input` decorator, we can always use `@Output` decorator to decorate any property of the child component. This property must be an event of the child component. The data to pass is called the event payload. In Angular 2, an event is defined with an `EventEmitter` object. Below is the sample example of an output event emitter property.

ANGULARJS2

```
1. @Output() click EventEmitter<string> = new EventEmitter<string>();
```

EventEmitter objects always accept any number of generic arguments. EventEmitter is an Angular2 abstraction and its only purpose is to emit the events in the components.

To declare output property, we need to first import the output and EventEmitter command from Angular Core module and then use @Output decorator to declare that property.

To demonstrate this output property, we will develop an email registration form. For this, please perform the steps given below.

Step 1

We create a typescript file named app.component.emailboxevent.ts and write down the code given below.

```
1. import { Component, Input, Output, EventEmitter } from '@angular/core';
2.
3. @Component({
4.   selector: 'emailboxevent',
5.   template: '<input placeholder="{{email}}" #emailbox/><button class="red" (click)="register(emailbox)">Register</button>',
6.   styles: ['.red {color:red;}']
7. })
8.
9. export class EmailBoxEventComponent {
10.   @Input('placeholder')
11.   email string = "Type your email..";
12.   @Output() emailregister EventEmitter<string> = new EventEmitter<string>();
13.
14.   register(txtEmail) {
15.     console.log('Email Address register properly...');
16.     if (txtEmail.value == '') {
17.       alert('Put your email address first..');
18.       return;
19.     }
20.     else {
21.       this.emailregister.emit(txtEmail.value);
22.     }
23.   }
24. }
```

Step 2

Now, add another typescript file named app.component.parent.ts and add the code given below.

```
1. //components files
2. import { Component, OnInit } from '@angular/core';
3. //
4.
5. @Component({
6.   moduleId: module.id,
7.   selector: 'myapp',
8.   templateUrl: 'app.component.parent.html',
9. })
10.
11. export class ParentComponent implements OnInit {
12.   constructor() { }
13.
14.   ngOnInit() { }
15.
16.   confirmregister(email string) void {
```

ANGULARJS2

```
17.         alert("Register Email Address is " + email);
18.     }
19. }
```

Step 3

Now, add HTML file named app.component.parent.html and write down the code given below in the file.

```
1. <div>
2.     <br />
3.     <h2>Email Register control with Events</h2>
4.     <emailboxevent [placeholder]="'Put your email'" (emailregister)="confirmregister($event)"></email
   boxevent><br />
5. </div>
```

Step 4

Now, add a style sheet file named style.css and add css codes given below in the files

```
1. .red {
2.     color black;
3.     backgroundcolor red;
4.     fontsize small;
5.     fontweight bold;
6. }
7.
8. .blue {
9.     color white;
10.    backgroundcolor blue;
11.    fontsize small;
12.    fontweight bold;
13.    fontfamily 'Franklin Gothic Medium', 'Arial Narrow', Arial, sansserif;
14.    fontstyle italic;
15. }
```

Step 5

Now, add another typescript file named app.module.ts and add the code given below.

```
1. import { NgModule } from '@angular/core';
2. import { BrowserModule } from '@angular/platformbrowser';
3. import { ParentComponent } from './src/app.component.parent';
4. import { EmailBoxEventComponent } from './src/app.component.emailboxevent';
5.
6.
7. @NgModule({
8.     imports [BrowserModule],
9.     declarations [ParentComponent, EmailBoxEventComponent],
10.    bootstrap [ParentComponent]
11. })
12. export class AppModule { }
```

Step 6

Now, add another HTML file named index.html and add the code given below.

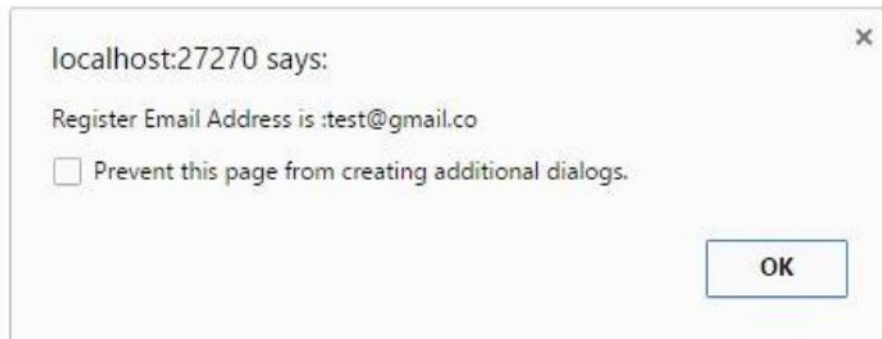
```
1. <!DOCTYPE html>
2. <html>
```

ANGULARJS2

```
3.   <head>
4.     <title>Angular2</title>
5.     <meta charset="UTF8">
6.     <meta name="viewport" content="width=device-width, initial-scale=1">
7.     <link rel="stylesheet" href="../styles.css">
8.
9.     <!-- Polyfill(s) for older browsers -->
10.    <script src="../node_modules/corejs/client/shim.min.js"></script>
11.    <script src="../node_modules/zone.js/dist/zone.js"></script>
12.    <script src="../node_modules/reflect-metadata/Reflect.js"></script>
13.    <script src="../node_modules/systemjs/dist/system.src.js"></script>
14.
15.    <script src="../systemjs.config.js"></script>
16.    <script>
17.      System.import('app').catch(function(err){ console.error(err); });
18.    </script>
19.  </head>
20.
21.  <body>
22.    <myapp>Loading... </myapp>
23.  </body>
24. </html>
```

Now, run the code in the browser and the output is as shown below. When we put an email address in the box and clicked on the register button. The child component emits the event to the parent with the given email value, which will be visible in an alert window.

Email Register control with Events

I am here to continue the discussion around [AngularJS 2.0](#). Today, we will discuss the output property binding in AngularJS 2. Also, in case, you did not have a look at the previous articles of this series, go through the links, mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)

In my previous article, I already discussed about the different process of data binding techniques including input property binding mechanism and output property for the Event Emitter in case of components within Angular 2.0.

ANGULARJS2

Now, in this article, I will discuss about the Directive concept in Angular 2. Before starting discussion about directive in Angular 2, first of all, we need to understand what directive is. As per Google, Directives are markers on a DOM object model or element (such as an attribute, element name, comment or CSS Class) that tells AngularJS's HTML compiler to attach a specified behavior to that DOM element or even to transform the DOM element and its child objects. Just like previous versions of Angular, Angular 2 also contains some built in directives and also allows us to create Custom Directives.

In Angular 2, basically the directives are also a component. Basically, there are three types of Directives in Angular 2, which are listed below.

- *Components* using `@Component` decorator is actually a directive with the template or template URL. I already discussed about this in my earlier articles in this series.
- *Attribute Directives* using `@Directive` decorator. It does not change DOM element but it changes the appearance of DOM element.
- *Structural Directives* using `@Directive` decorator. It can change DOM element or add any new DOM element or remove any DOM element.

In this article, we will discuss about the attribute Directives in Angular 2. The attribute Directive changes the appearance or behavior of a DOM element. These Directives look like regular HTML attributes in the templates and `ngModel` Directive, which is used for two-way is a perfect example of an attribute Directive. To create attribute Directives, we always need to use or inject the objects given below in our custom attribute Directive component class.

- *ElementRef*
While creating custom attribute Directive, we inject `ElementRef` in the constructor to access DOM element. Actually an `ElementRef` provides access to the underlying native element. `ElementRef` is a Service that grants us direct access to DOM element through its `nativeElement` property and we need to set the element's color, using the Browser DOM API.
- *Renderer*
While creating custom attribute directive, we inject `Renderer` in the constructor to access DOM element's style. Actually, we call the `renderer's setElementStyle` function. In this function, we pass the current DOM element with the help of `ElementRef` object and set the required attribute of the current element.
- *HostListener*
Some times, we may need to access an input property within the attribute directive, so that as per the given attribute directive, we can apply the related attribute within DOM element. For trap user actions, we can call different methods to handle the user actions. To access the method to operate user actions, we need to decorate the methods within the `@HostListener` method.

There are some inbuilt attribute directives in Angular2 like `ngClass`, `ngStyle`, `ngSwitch` etc. In the program given below, we demonstrate, how to use this attribute in our component.

Step 1

First, we add a TypeScript file named `app.component.colorchange.ts` and add the code given below.

```
1. import { Component, OnInit } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'attribute-directive',
6.   templateUrl: 'app.component.colorchange.html',
```

ANGULARJS2

```
7.     styles: [".red {color:red;}", ".blue {color:blue}", ".cyan {color : cyan}"]
8.   })
9.
10.  export class AttrDirectiveComponent implements OnInit {
11.    constructor() { }
12.    showColor: boolean = false;
13.    ngOnInit() { }
14.
15.    changeColor(): void {
16.      this.showColor = !this.showColor;
17.    }
18.  }
```

Step 2

Now, add HTML file named app.component.colorchange.html and add the code given below.

```
1. <div>
2.   <h3>This is a Attribute Directives</h3>
3.   <span [class.red]="true">Attribute Change</span><br />
4.   <span [ngClass]="{'blue': true}">Attribute Change by Using NgClass</span><br />
5.   <span [ngStyle]="{'font-size': '14px', 'color': 'green'}">Attribute Change by Using NgStyle</span>
6.   <br /><br />
7.   <span [class.cyan]="showColor">Attribute Change</span><br />
8.   <span [ngClass]="{'cyan': showColor}">Attribute Change by Using NgClass</span><br />
9.   <input type="button" value="Change Color" (click)="changeColor()" />
10.  <br /><br />
11.  <span [class.cyan]="showColor">Attribute Change</span><br />
12.  <span [ngClass]="{'cyan': showColor, 'red' : !showColor}">Attribute Change by Using NgClass</span>
    <br />
13. </div>
```

Step 3

Now, add another TypeScript file named app.module.ts and add the code given below.

```
1. import { NgModule } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { AttrDirectiveComponent } from './src/app.component.attrdirective';
4.
5. @NgModule({
6.   imports: [BrowserModule],
7.   declarations: [AttrDirectiveComponent],
8.   bootstrap: [AttrDirectiveComponent]
9. })
10. export class AppModule { }
```

Step 4

Now add another HTML file named index.html and add the code given below.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <title>Angular2</title>
5.   <meta charset="UTF-8">
6.   <meta name="viewport" content="width=device-width, initial-scale=1">
7.   <link rel="stylesheet" href="../styles.css">
8.   <!-- Polyfill(s) for older browsers -->
9.   <script src="../node_modules/core-js/client/shim.min.js"></script>
```

ANGULARJS2

```
10. <script src="../../node_modules/zone.js/dist/zone.js"></script>
11. <script src="../../node_modules/reflect-metadata/Reflect.js"></script>
12. <script src="../../node_modules/systemjs/dist/system.src.js"></script>
13. <script src="../../systemjs.config.js"></script>
14. <script>
15.     System.import('app').catch(function (err) { console.error(err); });
16. </script>
17. </head>
18. <body>
19.     <attribute-directive>Loading</attribute-directive>
20.
21. </body>
22. </html>
```

Now, run the code and output will be, as shown below. In the output, we create a button called change color on click of which it will execute a method and change the color of the element.

This is a Attribute Directives

Attribute Change
Attribute Change by Using NgClass
Attribute Change by Using NgStyle

Attribute Change
Attribute Change by Using NgClass

Attribute Change
Attribute Change by Using NgClass

This is a Attribute Directives

Attribute Change
Attribute Change by Using NgClass
Attribute Change by Using NgStyle

Attribute Change
Attribute Change by Using NgClass

Attribute Change
Attribute Change by Using NgClass

Now, we will create another table list type component, where on click on any table row, table background will be changed. For doing this, first we need to create a student list. For this, we will use ngSwitch directive.

Step 1

We need to add a TypeScript file named highlight.directive.ts, which basically performs the row highlight on row click.

```
1. import { Directive, ElementRef, Renderer, HostListener, Input } from "@angular/core";
2.
3. @Directive({
4.     selector: "[highlight]"
5. })
6.
7. export class Highlight {
8.     @Input("highlight") Color: string;
9.     constructor(private el: ElementRef, private renderer: Renderer) {
10.         this.renderer.setStyle(this.el.nativeElement, 'cursor', 'pointer');
11.     }
12.
13.     @HostListener("click")
14.
15.     onclick(): void {
16.         this.renderer.setStyle(this.el.nativeElement, 'backgroundColor', this.Color);
17.     }
18. }
```


ANGULARJS2

Step 2

Now, create another TypeScript file named app.component.list.ts and write down the code given below.

```
1. import { Component, OnInit, Directive } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'student-list',
6.   templateUrl: 'app.component.list.html'
7. })
8.
9. export class ListComponent implements OnInit {
10.   studentList: Array<any> = new Array<any>();
11.
12.   constructor() { }
13.   ngOnInit() {
14.     this.studentList = [
15.       { SrlNo: 1, Name: 'Raj i b Basak', Course: 'Bsc(Hons)', Grade: 'A' },
16.       { SrlNo: 2, Name: 'Raj i b Basak1', Course: 'BA', Grade: 'B' },
17.       { SrlNo: 3, Name: 'Raj i b Basak2', Course: 'BCom', Grade: 'A' },
18.       { SrlNo: 4, Name: 'Raj i b Basak3', Course: 'Bsc-Hons', Grade: 'C' },
19.       { SrlNo: 5, Name: 'Raj i b Basak4', Course: 'MBA', Grade: 'B' },
20.       { SrlNo: 6, Name: 'Raj i b Basak5', Course: 'MSc', Grade: 'B' },
21.       { SrlNo: 7, Name: 'Raj i b Basak6', Course: 'MBA', Grade: 'A' },
22.       { SrlNo: 8, Name: 'Raj i b Basak7', Course: 'MSc', Grade: 'C' },
23.       { SrlNo: 9, Name: 'Raj i b Basak8', Course: 'MA', Grade: 'D' },
24.       { SrlNo: 10, Name: 'Raj i b Basak9', Course: 'B. Tech', Grade: 'A' }
25.     ];
26.   }
27. }
```

Step 3

Create HTML file named app.component.list.html and add the code given below.

```
1. <div>
2.   <h2>Demonstrate ngSwi tch</h2>
3.   <table style="wi dth: 100%; border: sol id; border-col or: bl ue; border-wi dth: thi n; ">
4.     <thead>
5.       <tr >
6.         <td>Srl No</td>
7.         <td>Student Name</td>
8.         <td>Course</td>
9.         <td>Grade</td>
10.      </tr>
11.    </thead>
12.    <tbody>
13.      <tr *ngFor="l et student of studentList; " hi gh l ight="red">
14.        <td>
15.          <span>{{student. Srl No}}</span>
16.        </td>
17.        <td>
18.          <span >{{student. Name}}</span>
19.        </td>
20.        <td>
21.          <span >{{student. Course}}</span>
22.        </td>
23.        <td>
24.          <span >{{student. Grade}}</span>
25.        </td>
26.      </tr>
27.    </tbody>
28.  </table>
```

ANGULARJS2

```
29. </div>
```

Step 4

Now, include the app.component.list.ts within the app.component.colorchange.ts file, as shown below.

```
1. import { Component, OnInit } from '@angular/core';
2. import { ListComponent } from './app.component.list';
```

Step 5

Now, add HTML selector for student list within app.component.colorchange.html file, as shown below.

```
1. <div>
2.   <h3>This is a Attribute Directives</h3>
3.   <span [class.red]="true">Attribute Change</span><br />
4.   <span [ngClass]="{'blue': true}">Attribute Change by Using NgClass</span><br />
5.   <span [ngStyle]="{'font-size': '14px', 'color': 'green'}">Attribute Change by Using NgStyle</span>
6.   <br /><br />
7.   <span [class.cyan]="showColor">Attribute Change</span><br />
8.   <span [ngClass]="{'cyan': showColor}">Attribute Change by Using NgClass</span><br />
9.   <input type="button" value="Change Color" (click)="changeColor()" />
10.  <br /><br />
11.  <span [class.cyan]="showColor">Attribute Change</span><br />
12.  <span [ngClass]="{'cyan': showColor, 'red' : !showColor}">Attribute Change by Using NgClass</span>
13.  <br />
14.  <br />
15.  <student-list></student-list>
16. </div>
```

Step 6

Now, include the 2 component files within app.module.ts file, as shown below.

```
1. import { NgModule } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { AttrDirectiveComponent } from './src/app.component.attrdirective';
4. import { Highlight } from './src/highlight.directive';
5. import { ListComponent } from './src/app.component.list';
6.
7.
8. @NgModule({
9.   imports: [BrowserModule],
10.  declarations: [AttrDirectiveComponent, Highlight, ListComponent],
11.  bootstrap: [AttrDirectiveComponent]
12. })
13. export class AppModule { }
```

ANGULARJS2

Now, run the code, as shown below.

Demonstrate ngSwitch

Srl No	Student Name	Course	Grade
1	Rajib Basak	Bsc(Hons)	A
2	Rajib Basak1	BA	B
3	Rajib Basak2	BCom	A
4	Rajib Basak3	Bsc-Hons	C
5	Rajib Basak4	MBA	B
6	Rajib Basak5	MSc	B
7	Rajib Basak6	MBA	A
8	Rajib Basak7	MSc.	C
9	Rajib Basak8	MA	D
10	Rajib Basak9	B.Tech	A

I am here to continue the discussion around [AngularJS 2.0](#). Today, we will discuss Structural Directives in AngularJS 2. Also, in case, you did not have a look at the previous articles of this series, go through the links, mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)

In my previous article, I already discussed about the attribute directives in Angular 2.0. In that article, we discussed about some inbuilt attribute directives and also discussed about how to create custom attribute directives. Now in this article, I will discuss about another part of the directives -- structural directives. Components are simply directives with templates and they are the most commonly used in our application. Attribute directives are basically used to change the appearance of an element in our application, whereas structural directives can be used to modify the DOM by adding or removing any DOM elements and for this reason, it can be used in different scenarios.

In Angular 2.0, there are some in built structural directives i.e. `*ngIf`, `*ngFor`, `*ngSwitch` etc. Normally structural directives are prefixed with an asterisk which signals to Angular that the structure of the DOM element within the directive may change on depend of the some conditions.

ngIf

In Angular version 1.x, there was the `ng-show` and `ng-hide` directives which would show or hide the DOM elements on what the given expression evaluates by setting the display CSS property. In Angular 2.0, these two directives have been removed from the framework and a new directive named `ngIf` was introduced. The main difference of `ngIf` directive over `ng-show` or `ng-hide` is that it actually removes the element or components entirely from DOM. For `ng-show` or `ng-hide`, angular keeps the DOM elements/components in the page, so any component behaviors may keep running even if the component is not visible in the page. In Angular 2.0, `ng-show` or `ng-hide` directive is not available but we can obtain the same functionality by using the `[style.display]` property of any element. Now, one question always arises in our mind which is why angular removes component or elements from DOM in the case of `ngIf` directives? Actually, although in the earlier version, it hides the component or element, still the elements or

ANGULARJS2

components are attached with DOM. So it continues to fire its event listener. Also it keeps changing while the model data has been changed due to model binding. So in this way, this invisible components or elements use resources which might be useful for somewhere else. The performance and memory burden can be substantial and the user may not benefit at all. Setting `ngIf` value to false does effect the component resource consumptions. Angular removes the element from DOM, stops the change detection for the associated component, detaches it from DOM events, and destroys the components. The component can be garbage collected to free up memory. Components often have child components which themselves have children. All of them have been destroyed when `ngIf` destroys the common ancestor.

For demonstrating the `ngIf`, do the following steps,

Step 1

First Create the `app.module.ts` file and add the below code,

```
1. import { NgModule } from '@angular/core';
2. import { FormsModule } from '@angular/forms';
3. import { BrowserModule } from '@angular/platform-browser';
4.
5. import { ParentComponent } from './src/app.component.parent';
6. import { NgIfComponent } from './src/app.component.ngIf';
7.
8. @NgModule({
9.   imports: [BrowserModule, FormsModule],
10.  declarations: [ParentComponent, NgIfComponent],
11.  bootstrap: [ParentComponent]
12. })
13. export class AppModule { }
```

Step 2

Now create another ts file named `app.component.parent.ts` and add the below code,

```
1. import { Component, OnInit } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'structural-directive',
6.   templateUrl: 'app.component.parent.html',
7. })
8.
9. export class ParentComponent implements OnInit {
10.  constructor() { }
11.
12.  ngOnInit() { }
13. }
```

Step 3

Now create a html file named `app.component.parent.html` and add the below code,

```
1. <div>
2.   <h2 class="badge">Structural Directives Demonstrate</h2>
3.   <br />
4.   <toggle-text></toggle-text>
5. </div>
```

ANGULARJS2

Step 4

Now create another ts file named app.component.ngif.ts and add the below code,

```
1. import { Component, OnInit, Input } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'toggle-text',
6.   templateUrl: 'app.component.ngif.html'
7. })
8.
9. export class NgIfComponent implements OnInit {
10.   showInfo: boolean = false;
11.   caption: string = 'Show Text';
12.
13.   constructor() { }
14.   ngOnInit() { }
15.
16.   changeData(): void {
17.     this.showInfo = !this.showInfo;
18.     if (this.showInfo) {
19.       this.caption = 'Hide Text';
20.     }
21.     else {
22.       this.caption = 'Show Text';
23.     }
24.   }
25. }
```

Step 5

Now create another html file named app.component.ngif.html and add the below code,

```
1. <div>
2.   <input type="button" value="{{caption}}" (click)="changeData()"/>
3.   <br />
4.   <h2 *ngIf="showInfo"><span>Demonstrate of Structural Directives - *ngIf</span></h2>
5. </div>
```

Step 6

Now create another html file named index.html and add the below code,

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <title>Angular2 - Structural Directives</title>
5.     <meta charset="UTF-8">
6.     <meta name="viewport" content="width=device-width, initial-scale=1">
7.     <link rel="stylesheet" href=".. /styles.css">
8.
9.     <!-- Polyfill(s) for older browsers -->
10.    <script src=".. /node_modules/core-js/client/shim.min.js"></script>
11.    <script src=".. /node_modules/zone.js/dist/zone.js"></script>
12.    <script src=".. /node_modules/reflect-metadata/Reflect.js"></script>
13.    <script src=".. /node_modules/systemjs/dist/system.src.js"></script>
14.
15.    <script src=".. /systemjs.config.js"></script>
16.    <script>
17.      System.import('app').catch(function(err){ console.error(err); });
18.    </script>
```

ANGULARJS2

```
19.   </head>
20.
21.   <body>
22.     <structural -directive>Loading</structural -directive>
23.   </body>
24. </html>
```

Now run the code and the output as below,

Structural Directives Demonstrate

Show Text

Hide Text

Demonstrate of Structural Directives - *ngIf

ngFor

The ngFor directives instantiates a template once per item from an iterable. The context of each instantiated template inherits from the outer context with the given loop variable. ngFor provides several exported values that can be used to local variables :-

- *index* will be set to the current loop iteration for each template context
- *first* will be set to a boolean value indicating whether the item is the first one in the iteration.
- *last* will be set to a boolean value indicating whether the item is the last one in the iteration.
- *even* will be set to a boolean value indicating whether this item has an even index.
- *odd* will be set to a boolean value indicating whether this item has an odd index

Angular uses object identity to track insertions and deletions within the iterator and reproduce those changes in the DOM. This has important implications for animations and any stateful controls (such as <input> elements which accept user input) that are present. Inserted rows can be animated in, deleted rows can be animated out, and unchanged rows retain any unsaved state such as user input.

For demonstrate the ngFor directive, do the following code,

Step 1

Add a ts file named app.component.ngFor.ts and add the below code,

```
1. import { Component, OnInit, Directive } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'product-list',
6.   templateUrl: 'app.component.ngFor.html'
7. })
8.
9. export class NgForComponent implements OnInit {
10.   productList: Array<string> = ['iPhone', 'Galaxy 7.0', 'Blackberry 10Z'];
11.
12.   constructor() { }
13.   ngOnInit() { }
```

```
14. }
```

Step 2

Add a html file named app.component.ngFor.html and add the below code,

```
1. <div>
2.   <h2>Demonstrate ngFor</h2>
3.   <ul>
4.     <li *ngFor="let item of productList">
5.       {{item}}
6.     </li>
7.   </ul>
8. </div>
```

Now add the product-list selector tag within app.component.parent.html file and also take the reference of ngForComponent within the app.module.ts file and then run the code,

Demonstrate ngFor

- iPhone
- Galaxy 7.0
- Blackberry 10Z

ngSwitch

The ngSwitch directives is actually made of two directives, an attribute directive and a structural directive. It is similar to switch statement in javascript or other languages. ngSwitch stamps our nested views when their match expression value matches the value of the switch expression. The expression bound to the directives defines what will compared against in the switch structural directives. If an expression bound to ngSwitchCase matches the one given to ngSwitch, those components are created and the others destroyed. If none of the cases match, then components that have ngSwitchDefault bound to them will be created and the others destroyed. Note that multiple components can be matched using ngSwitchCase and in those cases all matching components will be created. Since components are created or destroyed be aware of the costs in doing so.

For demonstrating this, do the following,

Step 1

Add a ts file named app.component.ngSwitch.ts and add the below code,

```
1. import { Component, OnInit, Directive } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'student-list',
6.   templateUrl: 'app.component.ngSwitch.html'
7. })
8.
9. export class NgSwitchComponent implements OnInit {
10.   studentList: Array<any> = new Array<any>();
11.
12.   constructor() { }
13.   ngOnInit() {
14.     this.studentList = [
15.       { SrNo: 1, Name: 'Rajib Basak', Course: 'Bsc(Hons)', Grade: 'A' },
16.       { SrNo: 2, Name: 'Rajib Basak', Course: 'BA', Grade: 'B' },
```


ANGULARJS2

```
17.         { SrlNo: 3, Name: 'Raj i b Basak2', Course: 'BCom', Grade: 'A' },
18.         { SrlNo: 4, Name: 'Raj i b Basak3', Course: 'Bsc-Hons', Grade: 'C' },
19.         { SrlNo: 5, Name: 'Raj i b Basak4', Course: 'MBA', Grade: 'B' },
20.         { SrlNo: 6, Name: 'Raj i b Basak5', Course: 'MSc', Grade: 'B' },
21.         { SrlNo: 7, Name: 'Raj i b Basak6', Course: 'MBA', Grade: 'A' },
22.         { SrlNo: 8, Name: 'Raj i b Basak7', Course: 'MSc.', Grade: 'C' },
23.         { SrlNo: 9, Name: 'Raj i b Basak8', Course: 'MA', Grade: 'D' },
24.         { SrlNo: 10, Name: 'Raj i b Basak9', Course: 'B. Tech', Grade: 'A' }
25.     ];
26. }
27. }
```

Step 2

Add a html file named app.component.ngSwitch.html and add the below code,

```
1. <div>
2.     <h2>Demonstrate ngSwi tch</h2>
3.     <table style="width: 100%; border: sol id; border-col or: bl ue; border-wi dth: thi n;">
4.         <thead>
5.             <tr >
6.                 <td>Srl No</td>
7.                 <td>Student Name</td>
8.                 <td>Course</td>
9.                 <td>Grade</td>
10.            </tr>
11.        </thead>
12.        <tbody>
13.            <tr *ngFor="let student of studentList;" [ngSwi tch]="student.Grade">
14.                <td>
15.                    <span *ngSwi tchCase="' A' " [ngStyl e]="{' font-
16.                    si ze': ' 18px', ' col or': ' red' }">{{student. Srl No}}</span>
17.                    <span *ngSwi tchCase="' B' " [ngStyl e]="{' font-
18.                    si ze': ' 16px', ' col or': ' bl ue' }">{{student. Srl No}}</span>
19.                    <span *ngSwi tchCase="' C' " [ngStyl e]="{' font-
20.                    si ze': ' 14px', ' col or': ' green' }">{{student. Srl No}}</span>
21.                    <span *ngSwi tchDefaul t [ngStyl e]="{' font-
22.                    si ze': ' 12px', ' col or': ' bl ack' }">{{student. Srl No}}</span>
23.                </td>
24.                <td>
25.                    <span *ngSwi tchCase="' A' " [ngStyl e]="{' font-
26.                    si ze': ' 18px', ' col or': ' red' }">{{student. Name}}</span>
27.                    <span *ngSwi tchCase="' B' " [ngStyl e]="{' font-
28.                    si ze': ' 16px', ' col or': ' bl ue' }">{{student. Name}}</span>
29.                    <span *ngSwi tchCase="' C' " [ngStyl e]="{' font-
30.                    si ze': ' 14px', ' col or': ' green' }">{{student. Name}}</span>
31.                    <span *ngSwi tchDefaul t [ngStyl e]="{' font-
32.                    si ze': ' 12px', ' col or': ' bl ack' }">{{student. Name}}</span>
33.                </td>
34.                <td>
35.                    <span *ngSwi tchCase="' A' " [ngStyl e]="{' font-
36.                    si ze': ' 18px', ' col or': ' red' }">{{student. Course}}</span>
37.                    <span *ngSwi tchCase="' B' " [ngStyl e]="{' font-
38.                    si ze': ' 16px', ' col or': ' bl ue' }">{{student. Course}}</span>
39.                    <span *ngSwi tchCase="' C' " [ngStyl e]="{' font-
40.                    si ze': ' 14px', ' col or': ' green' }">{{student. Course}}</span>
41.                    <span *ngSwi tchDefaul t [ngStyl e]="{' font-
42.                    si ze': ' 12px', ' col or': ' bl ack' }">{{student. Course}}</span>
43.                </td>
44.                <td>
45.                    <span *ngSwi tchCase="' A' " [ngStyl e]="{' font-
46.                    si ze': ' 18px', ' col or': ' red' }">{{student. Grade}}</span>
47.                    <span *ngSwi tchCase="' B' " [ngStyl e]="{' font-
48.                    si ze': ' 16px', ' col or': ' bl ue' }">{{student. Grade}}</span>
```

ANGULARJS2

```
35.         <span *ngSwitchCase="'C'" [ngStyle]='{"font-  
    size': '14px', 'color': 'green'}">{{student.Grade}}</span>  
36.         <span *ngSwitchDefault [ngStyle]='{"font-  
    size': '12px', 'color': 'black'}">{{student.Grade}}</span>  
37.     </td>  
38. </tr>  
39. </tbody>  
40. </table>  
41. </div>
```

Step 3

Final code of app.module.ts file,

```
1. import { NgModule } from '@angular/core';  
2. import { FormsModule } from '@angular/forms';  
3. import { BrowserModule } from '@angular/platform-browser';  
4.  
5. import { ParentComponent } from './src/app.component.parent';  
6. import { NgIfComponent } from './src/app.component.ngIf';  
7. import { NgForComponent } from './src/app.component.ngFor';  
8. import { NgSwitchComponent } from './src/app.component.ngSwitch';  
9.  
10.  
11. @NgModule({  
12.   imports: [BrowserModule, FormsModule],  
13.   declarations: [ParentComponent, NgIfComponent, NgForComponent, NgSwitchComponent],  
14.   bootstrap: [ParentComponent]  
15. })  
16. export class AppModule { }
```

Step 4

Final code of app.component.parent.html file,

```
1. <div>  
2.   <h2 class="badge">Structural Directives Demonstrate</h2>  
3.   <br />  
4.   <toggle-text></toggle-text>  
5.   <br />  
6.   <product-list></product-list>  
7.   <br />  
8.   <student-list></student-list>  
9. </div>
```

Now run the code and output as below,

Demonstrate ngSwitch

Srl No	Student Name	Course	Grade
1	Rajib Basak	Bsc(Hons)	A
2	Rajib Basak1	BA	B
3	Rajib Basak2	BCom	A
4	Rajib Basak3	Bsc-Hons	C
5	Rajib Basak4	MBA	B
6	Rajib Basak5	MSc	B
7	Rajib Basak6	MBA	A
8	Rajib Basak7	MSc.	C
9	Rajib Basak8	MA	D
10	Rajib Basak9	B.Tech	A

ANGULARJS2

I am here to continue the discussion around [AngularJS 2.0](#). Today, we will discuss about pipes in AngularJS 2. Also, if, you did not have a look at the previous articles of this series, go through the links, mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJS 2.0 From Beginning - Structural Directives \(Day 7\)](#)

In my previous article, I already discussed about the structural Directives in Angular 2.0. In that article, we discussed about some inbuilt structural Directives. Now, in this article, I will discuss about pipes, one of the new flavors of Angular 2 architectures.

In Angular 1.x, we are familiar with the term filter. Filters are a great way of returning a new collection of data or formatting the new or doing any existing changes or mutating. Filters are basically just functions, which take a single value or a collection of values as an input and return a new value or collection of the value, which is based on the logical responsibilities. In Angular 2.0, the pipes are the modernized version of the filters. Most of the inbuilt filters of Angular 1.x have been converted as pipes in Angular 2.0 with some new pipes. Pipes are accessed in our templates in the same way that filters were--with the "pipe" character |. The table given below shows a comparison of pipes or filters in both Angular 1.x and Angular 2.0.

Filter / Pipes Available	Angular 1.x	Angular 2.0
currency	Yes	Yes
date	Yes	Yes
uppercase	Yes	Yes
lowercase	Yes	Yes
Json	Yes	Yes
limitTo	Yes	Yes
Number	Yes	
orderBy	Yes	
Filter	Yes	
async		Yes
decimal		Yes
percent		Yes

Thus, in this article, we will discuss how to use Angular 2.0 predefined pipes and also discuss how to create a custom pipe in Angular 2.0.

Basic Pipes

Most of the pipes provided by Angular 2.0 will be familiar to us, if we already worked in Angular 1.x. Actually pipes do not provide any new features in Angular 2.0. In Angular 2.0, we can use logic in the template. Also, we can execute or fire any function within the template to obtain its return value but actually pipe is a handsome way to handle this entire thing within the template. Also, it makes our code clean and structural. The pipe syntax starts with the actual input value followed by the pipe (|) symbol and then the pipe name. The parameters of that pipe can be sent separately by the colon (:) symbol. The order of execution of a pipe is right to left. Normally pipes works within our template and not in JavaScript code.

New pipes

The decimal and percent pipes are new in Angular 2.0. These take an argument that indicates the digit information, which should be used – that is how many integers and fraction digits; the number should be formatted with. The

ANGULARJS2

argument, which we pass for formatting follows this pattern: {minIntegerDigits}.{minFractionDigits} - {maxFractionDigits}.

Now, to show the inbuilt pipes in Angular 2.0, do the following.

Step 1

Create app.inbuildpipe.ts file and write down the code given below.

```
1. import { Component, OnInit } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'inbuild-pipe',
6.   templateUrl: 'app.component.inbuildpipe.html'
7. })
8.
9. export class InBuildPipeComponent implements OnInit {
10.   private todayDate: Date;
11.   private amount: number;
12.   private message: string;
13.
14.   constructor() { }
15.
16.   ngOnInit(): void {
17.     this.todayDate = new Date();
18.     this.amount = 100;
19.     this.message = "Angular 2 is a Component Based Framework";
20.   }
21.
22. }
```

Step 2

Now, add HTML file named app.component.inbuildpipe.html and add the code given below.

```
1. <div>
2.   <h1>Demonstrate of Pipe in Angular 2.0</h1>
3.
4.   <h2>Date Format</h2>
5.   Full Date : {{todayDate}}<br />
6.   Short Date : {{todayDate | date:'shortDate'}}<br />
7.   Medium Date : {{todayDate | date:'mediumDate'}}<br />
8.   Full Date : {{todayDate | date:'fullDate'}}<br />
9.   Time : {{todayDate | date:'HH:MM'}}<br />
10.
11.   <h2>Number Format</h2>
12.   No Formatting : {{amount}}<br />
13.   2 Decimal Place : {{amount | number:'2.2-2'}}
14.
15.   <h2>Currency Format</h2>
16.   No Formatting : {{amount}}<br />
17.   USD Doller($) : {{amount | currency:'USD':true}}<br />
18.   USD Doller : {{amount | currency:'USD':false}}<br />
19.   INR() : {{amount | currency:'INR':true}}<br />
20.   INR : {{amount | currency:'INR':false}}<br />
21.
22.   <h2>String Message</h2>
23.   Actual Message : {{message}}<br />
24.   Lower Case : {{message | lowercase}}<br />
25.   Upper Case : {{message | uppercase}}<br />
26.
27.   <h2> Percentage Pipes</h2>
```

ANGULARJS2

```
28.      2 Place Formatting : {{amount | percent :'.2'}}<br /><br />
29. </div>
```

Step 3

Now, add another TypeScript file named app.module.ts and add the code given below.

```
1. import { NgModule } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { InBui l dPi peComponent } from './src/app.component.inbui l dpi pe';
4.
5. @NgModule({
6.   imports: [BrowserModule],
7.   declarations: [InBui l dPi peComponent],
8.   bootstrap: [InBui l dPi peComponent]
9. })
10. export class AppModule { }
```

Step 4

Now, add another ts file named main.ts and add the code given below.

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
3. import { AppModule } from './app.module';
4.
5. const platform = platformBrowserDynamic();
6. platform.bootstrapModule(AppModule);
```

Step 5

Now, add another HTML file named index.html and add the code given below.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <title>Angular2</title>
5.   <meta charset="UTF-8">
6.   <meta name="viewport" content="width=device-width, initial-scale=1">
7.   <link rel="stylesheet" href="../styles.css">
8.   <!-- Polyfill(s) for older browsers -->
9.   <script src="../../node_modules/core-js/client/shim.min.js"></script>
10.  <script src="../../node_modules/zone.js/dist/zone.js"></script>
11.  <script src="../../node_modules/reflect-metadata/Reflect.js"></script>
12.  <script src="../../node_modules/systemjs/dist/system.src.js"></script>
13.  <script src="../../systemjs.config.js"></script>
14.  <script>
15.    System.import('app').catch(function (err) { console.error(err); });
16.  </script>
17. </head>
18. <body>
19.   <inbui l d-pi pe>Loadi ng</inbui l d-pi pe>
20.
21. </body>
22. </html>
```

Now, run the code and the output is shown below.

Demonstrate of Pipe in Angular 2.0

Date Format

Full Date : Thu Feb 16 2017 00:44:38 GMT+0530 (India Standard Time)
 Short Date : 2/16/2017
 Medium Date : Feb 16, 2017
 Full Date : Thursday, February 16, 2017
 Time : 00:02

Number Format

No Formatting : 100
 2 Decimal Place : 100.00

Currency Format

No Formatting : 100
 USD Doller(\$) : \$100.00
 USD Doller : USD100.00
 INR() : ₹100.00
 INR : INR100.00

String Message

Actual Message : Angular 2 is a Component Based Framework
 Lower Case : angular 2 is a component based framework
 Upper Case : ANGULAR 2 IS A COMPONENT BASED FRAMEWORK

Percentage Pipes

2 Place Formatting : 10,000.00%

Custom

Now, we define a custom pipe in Angular 2.0. To configure custom pipes, we need to use pipe object. For this, we need to define custom pipe with `@Pipe` decorator and use it by adding a pipes property to the `@View` decorator with the pipe class name. We use the transform method to do any logic, which is necessary to convert the value being passed in as an input value. We can get a hold of the arguments array as the second parameter and pass in as many as we like from the template.

Now, for doing a custom proper case pipe, we need to add a type script file named `app.component.propercase.ts` and add the code given below.

```
1. import { Pipe, PipeTransform } from "@angular/core"
2.
3. @Pipe({
4.   name: 'propercase'
5. })
6. export class ProperCasePipe implements PipeTransform {
7.   transform(value: string, reverse: boolean): string {
8.     if (typeof (value) == 'string') {
```

ANGULARJS2

```
9.         let intermediate = reverse == false ? value.toUpperCase() : value.toLowerCase();
10.        return (reverse == false ? intermediate[0].toLowerCase() :
11.                intermediate[0].toUpperCase()) + intermediate.substr(1);
12.    }
13.    else {
14.        return value;
15.    }
16. }
17. }
```

Now, change the code of app.module.ts file, as shown below.

```
1. import { NgModule } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { InBuildPipeComponent } from './src/app.component.inbuildpipe';
4. import { ProperCasePipe } from './src/app.component.propercasepipe';
5.
6.
7. @NgModule({
8.     imports: [BrowserModule],
9.     declarations: [InBuildPipeComponent, ProperCasePipe],
10.    bootstrap: [InBuildPipeComponent]
11. })
12. export class AppModule { }
```

Now, change the code of app.component.inbuildpipe.html file, as shown below.

```
1. <div>
2.     <h1>Demonstrate of Pipe in Angular 2.0</h1>
3.
4.     <h2>Date Format</h2>
5.     Full Date : {{todayDate}}<br />
6.     Short Date : {{todayDate | date:'shortDate'}}<br />
7.     Medium Date : {{todayDate | date:'mediumDate'}}<br />
8.     Full Date : {{todayDate | date:'fullDate'}}<br />
9.     Time : {{todayDate | date:'HH:MM'}}<br />
10.
11.    <h2>Number Format</h2>
12.    No Formatting : {{amount}}<br />
13.    2 Decimal Place : {{amount | number:'2.2-2'}}
14.
15.    <h2>Currency Format</h2>
16.    No Formatting : {{amount}}<br />
17.    USD Doller($) : {{amount | currency:'USD':true}}<br />
18.    USD Doller : {{amount | currency:'USD':false}}<br />
19.    INR() : {{amount | currency:'INR':true}}<br />
20.    INR : {{amount | currency:'INR':false}}<br />
21.
22.    <h2>String Message</h2>
23.    Actual Message : {{message}}<br />
24.    Lower Case : {{message | lowercase}}<br />
25.    Upper Case : {{message | uppercase}}<br />
26.
27.    <h2> Percentage Pipes</h2>
28.    2 Place Formatting : {{amount | percent : '.2'}}<br /><br />
29.
30.    Custom Pipe or Proper Case : {{message | propercase}}
31. </div>
```

Now, run the code.

ANGULARJS2

I am here to continue the discussion around [AngularJS 2.0](#). Today, we will discuss ViewChild in AngularJS 2. Also, in case you did not have a look at the previous articles of this series, go through the links mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJS 2.0 From Beginning - Structural Directives \(Day 7\)](#)
- [AngularJS 2.0 From Beginning - Pipes \(Day 8\)](#)

In my previous article, I already discussed about Pipes in Angular 2.0. In that article, we discussed about the Angular 2.0 predefined pipes and also discussed how to create custom pipes. Now in this article, I will discuss about ViewChild.

Basically, ViewChild is one of the new features introduced in Angular 2.0 framework. Since Angular 2.0 basically depends on component architecture, when we try to develop any web page or UI, it is most obvious that the page or UI must be a component, which basically contains a number of multiple different types of components within that component. In simple words, it is basically a parent component – child component based architecture. In this scenario, there are some situations when a parent component needs to interact with the child component. There are multiple ways to achieve this interaction between parent and child component. One of the ways is ViewChild. So when a parent component needs to execute or call any method of the child component, it can inject child component as a ViewChild within the parent component.

```
1. @ViewChild('calculator') private _calculator: ChildComponent;
```

For implementing ViewChild, we need to use @ViewChild decorator in the parent component. The @ViewChild decorator provides access to the class of child component from the parent component. The @ViewChild is a decorator function that takes the name of a component class as its input and finds its selector in the template of the containing component to bind to. @ViewChild can also be passed a template reference variable.

Now, for illustrating this, we will develop a calculator type UI. In this, we will develop two components.

- First component i.e. child component contains two textboxes from taking inputs and four buttons for performing four basic mathematical operations. After completion of the operations, each button will emit its final value to the parent component so that the parent component can show those values or perform any other operations on the basis of those value.
- Now, parent component wants to clear the values from both its own component and also from child component. For clearing the child component value, parent component will access the child component's clear() by using the @ViewChild decorator.

Now, to illustrate the ViewChild, follow the below lab exercise.

Step 1

First, create an html file named app.component.child.html and add the below code.

```
1. <div class="ibox-content">
2.   <div class="row">
3.     <div class="col-md-4">
4.       Enter First Number
5.     </div>
6.     <div class="col-md-8">
7.       <input type="number" [(ngModel)]="firstNumber" />
```


ANGULARJS2

```
8.         </div>
9.     </div>
10.    <div class="row">
11.        <div class="col-md-4">
12.            Enter Second Number
13.        </div>
14.        <div class="col-md-8">
15.            <input type="number" [(ngModel)]="secondNumber" />
16.        </div>
17.    </div>
18.    <div class="row">
19.        <div class="col-md-4">
20.        </div>
21.        <div class="col-md-8">
22.            <input type="button" value="+" (click)="add()" />
23.
24.            <input type="button" value="-" (click)="subtract()" />
25.
26.            <input type="button" value="X" (click)="multiply()" />
27.
28.            <input type="button" value="/" (click)="divide()" />
29.        </div>
30.    </div>
31. </div>
```

Step 2

Now, create a TypeScript file named app.component.child.ts and add the below code.

```
1. import { Component, OnInit, Output, EventEmitter } from '@angular/core';
2.
3. @Component({
4.     moduleId: module.id,
5.     selector: 'child',
6.     templateUrl: 'app.component.child.html'
7. })
8.
9. export class ChildComponent implements OnInit {
10.     private firstNumber: number = 0;
11.     private secondNumber: number = 0;
12.     private result: number = 0;
13.
14.     @Output() private addNumber: EventEmitter<number> = new EventEmitter<number>();
15.     @Output() private subtractNumber: EventEmitter<number> = new EventEmitter<number>();
16.     @Output() private multiplyNumber: EventEmitter<number> = new EventEmitter<number>();
17.     @Output() private divideNumber: EventEmitter<number> = new EventEmitter<number>();
18.
19.     constructor() { }
20.
21.     ngOnInit(): void {
22.     }
23.
24.     private add(): void {
25.         this.result = this.firstNumber + this.secondNumber;
26.         this.addNumber.emit(this.result);
27.     }
28.
29.     private subtract(): void {
30.         this.result = this.firstNumber - this.secondNumber;
31.         this.subtractNumber.emit(this.result);
32.     }
33.
34.     private multiply(): void {
35.         this.result = this.firstNumber * this.secondNumber;
```

ANGULARJS2

```
36.         this.multiplyNumber.emit(this.result);
37.     }
38.
39.     private divide(): void {
40.         this.result = this.firstNumber / this.secondNumber;
41.         this.divideNumber.emit(this.result);
42.     }
43.
44.     public clear(): void {
45.         this.firstNumber = 0;
46.         this.secondNumber = 0;
47.         this.result = 0;
48.     }
49. }
```

Step 3

Add another html file named app.component.parent.html files and add the below code.

```
1. <div>
2.   <h2>Simple Calculator</h2>
3.   <div>
4.     <child (addNumber)="add($event)" (subtractNumber)="subtract($event)" (multiplyNumber)="multiply($event)"
5.       (divideNumber)="divide($event)" #calculator></child>
6.   </div>
7.   <h3>Result</h3>
8.   <span>{{result}}</span>
9.   <br />
10.  <br />
11.  <input type="button" value="Reset" (click)="reset()" />
12. </div>
```

Step 4

Add another TS file named app.component.parent.ts and add the below code.

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2. import { ChildComponent } from './app.component.child';
3.
4. @Component({
5.   moduleId: module.id,
6.   selector: 'parent',
7.   templateUrl: 'app.component.parent.html'
8. })
9.
10. export class ParentComponent implements OnInit {
11.   private result: string = '';
12.
13.   @ViewChild('calculator') private _calculator: ChildComponent;
14.
15.   constructor() {
16.   }
17.
18.   ngOnInit(): void {
19.   }
20.
21.   private add(value: number): void {
22.     this.result = 'Result of Addition ' + value;
23.   }
24.
25.   private subtract(value: number): void {
```

ANGULARJS2

```
26.         this.result = 'Result of Subtraction ' + value;
27.     }
28.
29.     private multiply(value: number): void {
30.         this.result = 'Result of Multiply ' + value;
31.     }
32.
33.     private divide(value: number): void {
34.         this.result = 'Result of Division ' + value;
35.     }
36.
37.     private reset(): void {
38.         this.result = '';
39.         this._calculator.clear();
40.     }
41. }
```

Step 5

Now, add app.module.ts file and write down the below code.

```
1. import { NgModule } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { FormsModule } from "@angular/forms";
4. import { ParentComponent } from './src/app.component.parent';
5. import { ChildComponent } from './src/app.component.child';
6.
7. @NgModule({
8.     imports: [BrowserModule, FormsModule],
9.     declarations: [ParentComponent, ChildComponent],
10.    bootstrap: [ParentComponent]
11. })
12. export class AppModule { }
```

Step 6

Now, add main.ts file and write down the below code.

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
3. import { AppModule } from './app.module';
4.
5. const platform = platformBrowserDynamic();
6. platform.bootstrapModule(AppModule);
```

Step 7

Now, add the index.html file and write down the below code.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.     <title>Angular2 - ViewChild</title>
5.     <meta charset="UTF-8">
6.     <meta name="viewport" content="width=device-width, initial-scale=1">
7.     <link href="../../resources/style/style1.css" rel="stylesheet" />
8.     <!-- Polyfill(s) for older browsers -->
9.     <script src="../../node_modules/core-js/client/shim.min.js"></script>
10.    <script src="../../node_modules/zone.js/dist/zone.js"></script>
11.    <script src="../../node_modules/reflect-metadata/Reflect.js"></script>
```

ANGULARJS2

```
12. <script src="../../node_modules/systemjs/dist/system.src.js"></script>
13. <script src="../../systemjs.config.js"></script>
14. <script>
15.     System.import('app').catch(function (err) { console.error(err); });
16. </script>
17. </head>
18. <body>
19.     <parent>Loading</parent>
20.
21. </body>
22. </html>
```

Run the index.html file in browser and see the output.

Simple Calculator

Enter First Number

Enter Second Number

Result

Result of Addition 35

I am here to continue the discussion around [AngularJS 2.0](#). So far, we have discussed data binding, input properties, output properties, pipes, viewchild, and also directives in Angular 2.0. Now, in this article, I will discuss how to create a custom component like dynamic Grid using all the features which we discuss till now. Also, in case you did not have a look at the previous articles of this series, go through the links mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJS 2.0 From Beginning - Structural Directives \(Day 7\)](#)
- [AngularJS 2.0 From Beginning - Pipes \(Day 8\)](#)
- [AngularJS 2.0 From Beginning - Viewchild \(Day 9\)](#)

In this Angular 2 article series, we have already discussed basic components and directives concepts in Angular2. Now in this article, I will show you how to create custom components, such as records that display dynamic Grids for using the concept of directives, pipes, and other features of Angular 2.

Now, for creating the dynamic Grid, we first need to create a custom component which will represent the dynamic Grid components. For this purpose, we first add the TypeScript file named app.component.dynamicgrid.ts file and add the below code.

```
1. import { Component, OnInit, Input, Output } from '@angular/core';
```

ANGULARJS2

```
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'dynamic-grid',
6.   templateUrl: 'app.component.dynamicgrid.html'
7. })
8.
9. export class DynamicGridComponent implements OnInit {
10.
11.   private _source: Array<any> = new Array<any>();
12.
13.   @Input() private columns: Array<columnDef> = new Array<columnDef>();
14.
15.   constructor() { }
16.
17.   ngOnInit(): void {
18.     if (this.columns == null || this.columns == undefined) {
19.       alert("Column Definition of grid not provided.");
20.       return;
21.     }
22.   }
23.
24.   public bindData(data: Array<any>): void {
25.     if (data != null && data != undefined) {
26.       this._source = data;
27.     }
28.   }
29.
30.   public clearGrid(): void {
31.     this._source = [];
32.   }
33. }
34.
35. export class columnDef {
36.   caption: string;
37.   dataField: string;
38.   dataType: columnDataType;
39.   width: string;
40.   display: boolean = true;
41. }
42.
43. export enum columnDataType {
44.   Text,
45.   Number,
46.   Datetime,
47.   Integer,
48. }
```

In the above component, we created one Input Property, named columns, which is basically used for defining the column details such as caption and data field of the Grid columns. Also, we created two public methods within the Grid component called bindData() and clearGrid(). First function is used for binding data into the Grid component. This function takes an array type argument as data. The second method is used for clearing the Grid content.

In the above TypeScript file, we also defined two another classes named columnDef and columnDataType. These two classes are basically enum type classes which can be used for defining the column structure of the dynamic Grid component from the parent component.

Now, add an HTML file for dynamic Grid component templates named app.component.dynamicgrid.html and add the below code.

```
1. <div class="ibox-content">
2.   <div class="ibox-table">
3.     <div class="table-responsive">
4.       <table class="responsive-table table-striped table-bordered table-hover">
```

ANGULARJS2

```
5.         <thead>
6.             <tr>
7.                 <th *ngFor="let header of columns; let ind=index" [ngStyle]='{"width": header.
width}">
8.                     <span>
9.                         {{header.caption}}
10.                     </span>
11.                 </th>
12.             </tr>
13.         </thead>
14.         <tbody>
15.             <tr *ngFor="let s of _source; let i=index">
16.                 <td *ngFor="let cols of columns; let ind1=index" [ngSwitch]="cols.dataType" [
ngStyle]='{"width": cols.width}">
17.                     <span *ngSwitchCase="0">
18.                         {{s[cols.dataField]}}
19.                     </span>
20.                     <span *ngSwitchCase="1">
21.                         {{s[cols.dataField]}}
22.                     </span>
23.                     <span *ngSwitchCase="2">
24.                         {{s[cols.dataField] | date: "fullDate"}}
25.                     </span>
26.                     <span *ngSwitchCase="3">
27.                         {{s[cols.dataField]}}
28.                     </span>
29.                 </td>
30.             </tr>
31.         </tbody>
32.     </table>
33. </div>
34. </div>
35. </div>
```

In the above code, we used ngFor and ngSwitch for populating the dynamic table columns. Also, we used date pipes for date field value formatting.

Now, we need to create another component within which we use these dynamic Grid components. Actually, this contains an entry form and also the dynamic Grid component. For this purpose, we first create an HTML file named app.component.gridsetting.html and add the below code.

```
1. <div>
2.     <fieldset>
3.         <legend>
4.             Employee Information
5.         </legend>
6.         <table style="width: 100%; ">
7.             <tr>
8.                 <th>Employee Name</th>
9.                 <td><input type="text" maxLength="100" [(ngModel)]="_modelData.employeeName" /></td>
10.            </tr>
11.            <tr>
12.                <th>Department</th>
13.                <td><input type="text" maxLength="100" [(ngModel)]="_modelData.department" /></td>
14.            </tr>
15.            <tr>
16.                <th>Designation</th>
17.                <td><input type="text" maxLength="100" [(ngModel)]="_modelData.designation" /></td>
18.            </tr>
19.            <tr>
20.                <th>Date of Join</th>
21.                <td><input type="date" [(ngModel)]="_modelData.doj" /></td>
22.            </tr>
23.        </table>
```

ANGULARJS2

```
24.         <th>Salary</th>
25.         <td><input type="number" [(ngModel)]="_modelData.salary"/></td>
26.     </tr>
27. </tr>
28.         <td style="text-align: right;"></td>
29.         <td>
30.             <input type="button" value="Add" (click)="addData();" />
31.
32.             <input type="button" value="Clear" (click)="clearData()" />
33.
34.             <input type="button" value="Reset Grid" (click)="resetGrid()" />
35.         </td>
36.     </tr>
37. </table>
38. </fieldset>
39. </div>
40. <div>
41.     <h2>Employee Details</h2>
42.     <dynamic-grid [columns]="_columnDetails" #grid></dynamic-grid>
43. </div>
```

Now, add a TypeScript file named app.component.gridsetting.ts and add the below code.

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2. import { DynamicGridComponent, columnDef, columnDataType } from './app.component.dynamicgrid';
3.
4. @Component({
5.     moduleId: module.id,
6.     selector: 'grid-setting',
7.     templateUrl: 'app.component.gridsetting.html'
8. })
9.
10. export class GridSettingComponent implements OnInit {
11.     private _columnDetails: Array<columnDef>;
12.     private _modelData: any = {};
13.     private _gridData: Array<any> = new Array<any>();
14.     private _srlNo: number;
15.
16.     @ViewChild('grid') private _gridComponent: DynamicGridComponent;
17.
18.     constructor() {
19.         this._columnDetails = [
20.             { caption: 'Srl No', dataField: 'srlNo', dataType: columnDataType.Integer, width: '10%',
21.               display: true },
22.             { caption: 'Employee Name', dataField: 'employeeName', dataType: columnDataType.Text, width: '30%',
23.               display: true },
24.             { caption: 'Department', dataField: 'department', dataType: columnDataType.Text, width: '15%',
25.               display: true },
26.             { caption: 'Designation', dataField: 'designation', dataType: columnDataType.Text, width: '15%',
27.               display: true },
28.             { caption: 'Date Of Join', dataField: 'doj', dataType: columnDataType.DateTime, width: '15%',
29.               display: true },
30.             { caption: 'Salary', dataField: 'salary', dataType: columnDataType.Number, width: '15%',
31.               display: true }];
32.     }
33.
34.     ngOnInit(): void {
35.         this._srlNo = 1;
36.     }
37.
38.     private addData(): void {
39.         if (this.validateData()) {
40.             this._modelData.srlNo = this._srlNo;
41.             this._srlNo += 1;
42.             this._modelData.doj = new Date(this._modelData.doj);
43.             this._gridData.push(this._modelData);
44.         }
45.     }
46.
47.     private validateData(): boolean {
48.         if (this._modelData.srlNo < 1 || this._modelData.doj < new Date(2000, 0, 1)) {
49.             return false;
50.         }
51.         return true;
52.     }
53.
54.     private clearData(): void {
55.         this._modelData = {};
56.     }
57.
58.     private resetGrid(): void {
59.         this._gridData = [];
60.     }
61. }
```

ANGULARJS2

```
38.         this.clearData();
39.         this._gridComponent.bindData(this._gridData);
40.     }
41. }
42.
43. private clearData(): void {
44.     this._modelData = {};
45. }
46.
47. private validateData(): boolean {
48.     let status = true;
49.     if (this.isUndefined(this._modelData.employeeName)) {
50.         alert('Employee Name never blank');
51.         status = false;
52.     }
53.     else if (this.isUndefined(this._modelData.department)) {
54.         alert('Department never blank');
55.         status = false;
56.     }
57.     else if (this.isUndefined(this._modelData.designation)) {
58.         alert('Designation never blank');
59.         status = false;
60.     }
61.     else if (this.isUndefined(this._modelData.salary)) {
62.         alert('Salary never blank');
63.         status = false;
64.     }
65.     else if (this.isUndefined(this._modelData.doj)) {
66.         alert('Date of Join never blank');
67.         status = false;
68.     }
69.     return status;
70. }
71.
72. private isUndefined(data: any): boolean {
73.     return typeof (data) === "undefined";
74. }
75.
76. private resetGrid(): void {
77.     this._gridData = [];
78.     this._modelData = {};
79.     this._gridComponent.clearGrid();
80. }
81. }
```

In the above component, for accessing the public method of the Grid component, we use viewchild concept. Now, add another TypeScript file named app.module.ts and write down the below code.

```
1. import { NgModule } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { FormsModule } from "@angular/forms";
4. import { GridSettingComponent } from './src/app.component.gridsetting';
5. import { DynamicGridComponent } from './src/app.component.dynamicgrid';
6.
7. @NgModule({
8.     imports: [BrowserModule, FormsModule],
9.     declarations: [GridSettingComponent, DynamicGridComponent],
10.    bootstrap: [GridSettingComponent]
11. })
12. export class AppModule { }
```

Now, add another TypeScript file named main.ts and add the below code.

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
```


ANGULARJS2

```
3. import { AppModule } from './app.module';
4.
5. const platform = platformBrowserDynamic();
6. platform.bootstrapModule(AppModule);
```

Now, add another HTML file named index.html and add the below code.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <title>Angular2 - Custom Grid</title>
5.   <meta charset="UTF-8">
6.   <meta name="viewport" content="width=device-width, initial-scale=1">
7.   <link href="../../resources/style/style1.css" rel="stylesheet" />
8.   <!-- Polyfill(s) for older browsers -->
9.   <script src="../../node_modules/core-js/client/shim.min.js"></script>
10.  <script src="../../node_modules/zone.js/dist/zone.js"></script>
11.  <script src="../../node_modules/reflect-metadata/Reflect.js"></script>
12.  <script src="../../node_modules/systemjs/dist/system.src.js"></script>
13.  <script src="../../systemjs.config.js"></script>
14.  <script>
15.    System.import('app').catch(function (err) { console.error(err); });
16.  </script>
17. </head>
18. <body>
19.   <grid-setting>Loading</grid-setting>
20.
21. </body>
22. </html>
```

Now, run the code. The output is shown below.

Employee Information

Employee Name

Department

Designation

Date of Join

Salary

mm/dd/yyyy

Add

Clear

Reset Grid

Employee Details

Srl No	Employee Name	Departmentr	Designation	Date Of Join	Salary
1	Souvik Dutta	Accounts	Clerk	Saturday, January 1, 2011	15000
2	Romen Dutta	IT Admin	Manager	Thursday, January 6, 2011	35000

I am here to continue the discussion around [AngularJS 2.0](#). So far we have discussed about data binding, input properties, output properties, pipes, viewchild, and also about directives in Angular 2.0. Now in this article, I will discuss how to create a Service in Angular 2.0. Also, in case you did not have a look at the previous articles of this series, go through the links mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJs 2.0 From Beginning - Output Property Binding \(Day 5\)](#)

ANGULARJS2

- [AngularJs 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJs 2.0 From Beginning - Structural Directives \(Day 7\)](#)
- [AngularJs 2.0 From Beginning - Pipes \(Day 8\)](#)
- [AngularJs 2.0 From Beginning - Viewchild \(Day 9\)](#)
- [AngularJs 2.0 From Beginning - Dynamic Grid \(Day 10\)](#)

An Angular 2 Service is simply a JavaScript function, including its related properties and methods which can perform a particular task or a group of tasks. Actually, Service is a mechanism to use shared responsibilities within one or multiple components. As we already know, we can create components in Angular 2 and nest multiple components together within a component using selector, once our components are nested, we need to manipulate some data within the multiple components. In this case, Service is the best way to handle the situation. Service is the best place where we can take data from other sources or write down some calculations. Similarly, Service can be shared between multiple components as per our need.

Angular 2.0 has greatly simplified the concept of Service over Angular 1.x. In Angular 1, there were service, factory, provider, delegate, value etc. and it was not always clear when to use which one. Angular 2 simply changes the concept of Service. There are two steps for creating a Service in Angular 2.0.

1. Create a class with @Injectable decorator.
2. Register the class with provider or inject the class by using dependency injection.

@Injectable

@Injectable is actually is a decorator. Decorators are a proposed extension in JavaScript. In short, decorator provides the facility of modifying or using methods, classes, properties and parameters. Injectables are just normal classes (normal objects) and as such, they have no special lifecycle. When an object of your class is created, the class's constructor is called, so that's what your "OnInit" would be.

As for the destruction, a service does not really get destroyed. The only thing that might happen is that it gets garbage collected once there is no longer a reference to it, which likely happens after the dependency injector is removed itself. But you generally have no control over it, and there is no concept of a deconstructor in JavaScript.

@Injectable() lets Angular know that a *class* can be used with the dependency injector. @Injectable() is not *strictly* required if the class has other Angular decorators on it or does not have any dependencies. What is important is that any class that is going to be injected with Angular *is decorated*. However, the best practice is to decorate injectables with @Injectable(), as it makes more sense to the reader.

```
1. @Injectable()
2. export class SampleService {
3.     constructor() {
4.         console.log('Sample service is created');
5.     }
6. }
```

What is Dependency Injection?

Actually, dependency injection is an important and useful application design pattern. Angular 2.0 has its own dependency injection framework. Basically, it is a coding pattern in which classes receive their dependencies from external sources rather than creating them.

Dependency Injection in Angular 2.0

Dependency injection has always been one of Angular's biggest features and selling points. It allows us to inject dependencies in different components across our applications, without needing to know how those dependencies are created, or what dependencies they need themselves. However, it turns out that the current dependency injection system in Angular 1.x has some problems that need to be solved in Angular 2.x, in order to build the next generation framework.

ANGULARJS2

Dependency Injection basically consists of three things,

1. Injector – The Injector object that exposes APIs to us to create instances of dependencies
2. Provider – A Provider is like a commander that tells the injector how to create an instance of a dependency. A provider takes a token and maps that to a factory function that creates an objects.
3. Dependency – A Dependency is the type of which an object should be created.

Now, for demonstrating the Service, we will create a Service called `StudentService` which is basically storing the student details data. When we add any new student information from entry form, that data is also included in the `StudentService`. Also, on requirement, we can retrieve all the students list from the Service. For this purpose, we need to add a TypeScript file called `app.service.student.ts` and add the below code.

```
1. import { Injectable } from "@angular/core";
2.
3. @Injectable()
4. export class StudentService {
5.     private _studentList: Array<any> = [];
6.
7.     constructor() {
8.         this._studentList = [{name: 'Ami t Roy', age: 20, ci ty: 'Kol kata', dob: '01-01-1997' }];
9.     }
10.
11.     returnStudentData(): Array<any> {
12.         return this._studentList;
13.     }
14.
15.     addStudentData(i tem: any): void {
16.         this._studentLi st.push(i tem);
17.     }
18. }
```

In the above Service, we defined a private array type variable called "`_studentList`" which basically stores all the students details. Also, we created two public methods named `returnStudentData` (for fetching student list from component) and `addStudentData` for new students' entry within the list.

Now, we need to define the student form component. For this, we need to add an HTML file named `app.component.student.html` and add the below code.

```
1. <div>
2.     <h2>Student Form</h2>
3.     <table style="width: 80%; ">
4.         <tr>
5.             <td>Student Name</td>
6.             <td><input type="text" [(ngModel)]="_model.name" /></td>
7.         </tr>
8.         <tr>
9.             <td>Age</td>
10.            <td><input type="number" [(ngModel)]="_model.age" /></td>
11.        </tr>
12.        <tr>
13.            <td>Ci ty</td>
14.            <td><input type="text" [(ngModel)]="_model.ci ty" /></td>
15.        </tr>
16.        <tr>
17.            <td>Student DOB</td>
18.            <td><input type="date" [(ngModel)]="_model.dob" /></td>
19.        </tr>
20.        <tr>
21.            <td></td>
22.            <td>
23.                <input type="button" value="Submi t" (cli ck)="submi t()" />
24.                <input type="button" value="Reset" (cli ck)="reset()" />
25.            </td>
26.        </tr>
27.    </table>
28. </div>
```

ANGULARJS2

```
26.         </tr>
27.     </table>
28.     <h3>Student Details</h3>
29.     <div class="ibox-content">
30.         <div class="ibox-table">
31.             <div class="table-responsive">
32.                 <table class="responsive-table table-striped table-bordered table-hover">
33.                     <thead>
34.                         <tr>
35.                             <th style="width: 40%; ">
36.                                 <span>Student's Name</span>
37.                             </th>
38.                             <th style="width: 15%; ">
39.                                 <span>Age</span>
40.                             </th>
41.                             <th style="width: 25%; ">
42.                                 <span>City</span>
43.                             </th>
44.                             <th style="width: 20%; ">
45.                                 <span>Date of Birth</span>
46.                             </th>
47.                         </tr>
48.                     </thead>
49.                     <tbody>
50.                         <tr *ngFor="let item of _source; let i=index">
51.                             <td><span>{{item.name}}</span></td>
52.                             <td><span>{{item.age}}</span></td>
53.                             <td><span>{{item.city}}</span></td>
54.                             <td><span>{{item.dob}}</span></td>
55.                         </tr>
56.                     </tbody>
57.                 </table>
58.             </div>
59.         </div>
60.     </div>
61.
62. </div>
```

Now, add another TS file named app.component.student.ts and add the below code.

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2. import { StudentService } from './app.service.student';
3.
4. @Component({
5.     moduleId: module.id,
6.     selector: 'student',
7.     templateUrl: 'app.component.student.html',
8.     providers: [StudentService]
9. })
10.
11. export class StudentComponent implements OnInit {
12.
13.     private _model: any = {};
14.     private _source: Array<any>;
15.
16.     constructor(private _service: StudentService) {
17.         this._source = this._service.returnStudentData();
18.     }
19.
20.     ngOnInit(): void {
21.     }
22.
23.     private submit(): void {
24.         if (this.validate()) {
25.             this._service.addStudentData(this._model);
26.             this.reset();
```

ANGULARJS2

```
27.     }
28.   }
29.
30.   private reset(): void {
31.     this._model = {};
32.   }
33.
34.   private validate(): boolean {
35.     debugger;
36.     let status: boolean = true;
37.     if (typeof (this._model.name) === "undefined") {
38.       alert('Name is Blank');
39.       status = false;
40.       return;
41.     }
42.     else if (typeof (this._model.age) === "undefined") {
43.       alert('Age is Blank');
44.       status = false;
45.       return;
46.     }
47.     else if (typeof (this._model.city) === "undefined") {
48.       alert('City is Blank');
49.       status = false;
50.       return;
51.     }
52.     else if (typeof (this._model.dob) === "undefined") {
53.       alert('dob is Blank');
54.       status = false;
55.       return;
56.     }
57.     return status;
58.   }
59. }
```

In this above component, we used provider attribute for injecting the StudentService within the component.

Now, add another TypeScript file named app.module.ts and add the below code.

```
1. import { NgModule } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { FormsModule } from "@angular/forms";
4. import { StudentComponent } from '../src/app.component.student';
5.
6. @NgModule({
7.   imports: [BrowserModule, FormsModule],
8.   declarations: [StudentComponent],
9.   bootstrap: [StudentComponent]
10. })
11. export class AppModule { }
```

Now, add another TypeScript file named main.ts and add the below code.

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
3. import { AppModule } from './app.module';
4.
5. const platform = platformBrowserDynamic();
6. platform.bootstrapModule(AppModule);
```

Now, add another HTML file named index.html and add the below code.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
```

ANGULARJS2

```
4. <title>Angular2 - Service</title>
5. <meta charset="UTF-8">
6. <meta name="viewport" content="width=device-width, initial-scale=1">
7. <link href="../../../resources/style/style1.css" rel="stylesheet" />
8. <!-- Polyfill(s) for older browsers -->
9. <script src="../../../node_modules/core-js/client/shim.min.js"></script>
10. <script src="../../../node_modules/zone.js/dist/zone.js"></script>
11. <script src="../../../node_modules/reflect-metadata/Reflect.js"></script>
12. <script src="../../../node_modules/systemjs/dist/system.src.js"></script>
13. <script src="../../../systemjs.config.js"></script>
14. <script>
15.     System.import('app').catch(function (err) { console.error(err); });
16. </script>
17. </head>
18. <body>
19.     <student>Loading</student>
20.
21. </body>
22. </html>
```

Now, when we run the index.html file in the browser, we get the following output.

Student Form

Student Name	<input type="text"/>
Age	<input type="text"/>
City	<input type="text"/>
Student DOB	<input type="text" value="mm/dd/yyyy"/>
<input type="button" value="Submit"/> <input type="button" value="Reset"/>	

Student Details

Student's Name	Age	City	Date of Birth
Amit Roy	20	Kolkata	01-01-1997
Sunil Sharma	19	New Delhi	1998-10-06

I am here to continue the discussion around [AngularJS 2.0](#). So far, we discussed about data binding, input properties, output properties, pipes, viewchild, and also about directives in Angular 2.0. Now in this article, I will discuss how to use ngContent or transclusion in Angular 2.0. Also, in case you did not have a look at the previous articles of this series, go through the links mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJS 2.0 From Beginning - Structural Directives \(Day 7\)](#)
- [AngularJS 2.0 From Beginning - Pipes \(Day 8\)](#)
- [AngularJS 2.0 From Beginning - Viewchild \(Day 9\)](#)
- [AngularJS 2.0 From Beginning - Dynamic Grid \(Day 10\)](#)
- [AngularJS 2.0 From Beginning - Service \(Day 11\)](#)

In my previous article, I already discussed about the injectable Service in Angular 2.0. In this article, we will discuss about the content template or ng-content in Angular 2.0.

In Angular 1.0, there is a concept of Transclusion. Actually, transclusion in an Angular 1.x is represent the content replacement such as a text node or html, and injecting it into a template at a specific entry time. Same thing in

ANGULARJS2

Angular 2.0 is totally forbidden. This is now done in Angular 2.0 through modern web APIs, such as shadow DOM which is known as content projection.

Content Projection

So now, we know what we are looking from an Angular 1.x perspective so that we can easily migrate the same in Angular 2.0. Actually, projection is a very important concept in Angular. It enables developer to develop or build reusable components and make the application more scalable and flexible.

In Web Components, we *had* the `<content>` element, which was recently deprecated, which acted as a Shadow DOM insertion point. Angular 2 allows Shadow DOM through the use of ViewEncapsulation. Early alpha versions of Angular 2 adopted the `<content>` element, however due to the nature of a bunch of Web Component helper elements being deprecated, it was changed to `<ng-content>`. Actually ViewEncapsulation defines whether the template and styles defined within the component can affect the whole application or vice versa. Angular provides three encapsulation strategies,

- *Emulated (default)*

styles from main HTML propagate to the component. Styles defined in this component's `@Component` decorator are scoped to this component only.

- *Native*

styles from main HTML do not propagate to the component. Styles defined in this component's `@Component` decorator are scoped to this component only.

- *None*

styles from the component propagate back to the main HTML and therefore are visible to all components on the page. Be careful with apps that have None and Native components in the application. All components with None encapsulation will have their styles duplicated in all components with Native encapsulation.

To illustrate ng-content that, suppose we have a children component.

```
1. import { Component } from '@angular/core';
2.
3. @Component({
4.   selector: 'child',
5.   template: `
6.     <div style="border: 1px solid blue; padding: 1rem;">
7.       <h4>Child Component</h4>
8.       <ng-content></ng-content>
9.     </div>
10. `
11. })
12. export class ChildComponent {
13. }
```

Then, when we use ChildComponent in the template.

```
1. <child>
2.   <p>My <i>dynamic</i> content.</p>
3. </child>
```

ANGULARJS2

This is telling Angular that for any markup that appears between the opening and closing tag of `<child>`, to place inside of `<ng-content></ng-content>`. When doing this, we can have other components, markup, etc. projected here and the ChildComponent does not need to know about or care what is being provided.

But what if we have multiple `<ng-content></ng-content>` and want to specify the position of the projected content to certain ng-content? For example, for the previous ChildComponent, if we want to format the projected content into an extra area1 and area2 section. Then in the template, we can use directives, say, `<area1>` to specify the position of projected content to the ng-content with `select="area1"`.

For demonstrating the concept of ngContent, we will create a modal component where modal body and footer section dynamically added by the HTML tags.

For example, we will create the following files with the below code.

File Name - app.component.modal.html

```
1. <div class="modal" id="myModal" tabIndex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-
   hidden="true" [ngStyle]='{display: display}'">
2.   <div class="modal-dialog">
3.     <div class="modal-content animated bounceInRight">
4.       <div class="modal-header">
5.         <button type="button" class="close" (click)="fnClose()"></button>
6.         <h3 class="modal-title">{{header}}</h3>
7.       </div>
8.       <div class="modal-body">
9.         <ng-content select="content-body"></ng-content>
10.      </div>
11.      <div class="modal-footer">
12.        <ng-content select="content-footer"></ng-content>
13.      </div>
14.    </div>
15.  </div>
16. </div>
```

File Name - app.component.modal.ts

```
1. import { Component, OnInit, ViewChild, Input } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'modal-window',
6.   templateUrl: 'app.component.modal.html'
7. })
8.
9. export class ModalComponent implements OnInit {
10.   @Input() private display: string = 'none';
11.   @Input('header-caption') private header: string = 'Modal';
12.
13.   constructor() {
14.   }
15.
16.   ngOnInit(): void {
17.   }
18.
19.   private fnClose(): void {
20.     this.display = 'none';
21.   }
22.
23.   showModal(): void {
24.     this.display = 'block';
25.   }
26. }
```


ANGULARJS2

```
27.     close(): void {
28.         this.fnClose();
29.     }
30.
31.     setModalTitle(args: string): void {
32.         this.header = args;
33.     }
34. }
```

File Name - app.component.parent.html

```
1. <div>
2.     <h2>Demonstrate Modal Window using ngContent</h2>
3.     <input type="button" value="Show Modal" class="btn-group" (click)="fnOpenModal()" />
4.     <br />
5.     <modal-window [header-caption]="caption" #modal>
6.         <content-body>
7.             <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum tincidunt est vitae ultrices accumsan. Aliquam ornare lacus adipiscing, posuere lectus et, fringilla augue.</p>
8.             <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum tincidunt est vitae ultrices accumsan. Aliquam ornare lacus adipiscing, posuere lectus et, fringilla augue.</p>
9.             <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum tincidunt est vitae ultrices accumsan. Aliquam ornare lacus adipiscing, posuere lectus et, fringilla augue.</p>
10.            <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum tincidunt est vitae ultrices accumsan. Aliquam ornare lacus adipiscing, posuere lectus et, fringilla augue.</p>
11.            <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum tincidunt est vitae ultrices accumsan. Aliquam ornare lacus adipiscing, posuere lectus et, fringilla augue.</p>
12.        </content-body>
13.        <content-footer>
14.            <input type="button" class="btn-default active" class="btn btn-primary" value="Modal Close" (click)="fnHideModal();" />
15.        </content-footer>
16.    </modal-window>
17. </div>
```

File Name - app.component.parent.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2. import { ModalComponent } from './app.component.modal';
3.
4. @Component({
5.     moduleId: module.id,
6.     selector: 'parent-content',
7.     templateUrl: 'app.component.parent.html'
8. })
9.
10. export class ParentComponent implements OnInit {
11.
12.     private caption: string = 'Custom Modal';
13.     @ViewChild('modal') private _ctrlModal: ModalComponent;
14.
15.     constructor() {
16.     }
17.
18.     ngOnInit(): void {
19.     }
20.
21.     private fnOpenModal(): void {
22.         this._ctrlModal.showModal();
23.     }
24.
25.     private fnHideModal(): void {
26.         this._ctrlModal.close();
27.     }
28. }
```

ANGULARJS2

```
27.     }  
28. }
```

File Name - app.module.ts

```
1. import { NgModule, NO_ERRORS_SCHEMA } from '@angular/core';  
2. import { BrowserModule } from '@angular/platform-browser';  
3. import { FormsModule } from "@angular/forms";  
4. import { ParentComponent } from './src/app.component.parent';  
5. import { ModalComponent } from './src/app.component.modal';  
6.  
7. @NgModule({  
8.   imports: [BrowserModule, FormsModule],  
9.   declarations: [ParentComponent, ModalComponent],  
10.  bootstrap: [ParentComponent],  
11.  schemas: [NO_ERRORS_SCHEMA]  
12. })  
13. export class AppModule { }
```

File Name - main.ts

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';  
2.  
3. import { AppModule } from './app.module';  
4.  
5. const platform = platformBrowserDynamic();  
6. platform.bootstrapModule(AppModule);
```

File Name - index.html

```
1. <!DOCTYPE html>  
2. <html>  
3. <head>  
4.   <title>Angular2 - ngContent</title>  
5.   <meta charset="UTF-8">  
6.   <meta name="viewport" content="width=device-width, initial-scale=1">  
7.   <link href="../../resources/style/bootstrap.css" rel="stylesheet" />  
8.   <link href="../../resources/style/style1.css" rel="stylesheet" />  
9.  
10. </head>  
11. <body>  
12.   <parent-content>Loading</parent-content>  
13.   <!-- Polyfill(s) for older browsers -->  
14.   <script src="../../resources/js/jquery-2.1.1.js"></script>  
15.   <script src="../../resources/js/bootstrap.js"></script>  
16.  
17.   <script src="../../node_modules/core-js/client/shim.min.js"></script>  
18.   <script src="../../node_modules/zone.js/dist/zone.js"></script>  
19.   <script src="../../node_modules/reflect-metadata/Reflect.js"></script>  
20.   <script src="../../node_modules/systemjs/dist/system.src.js"></script>  
21.   <script src="../../systemjs.config.js"></script>  
22.   <script>  
23.     System.import('app').catch(function (err) { console.error(err); });  
24.   </script>  
25. </body>  
26. </html>
```

Now, execute the program and output as below.

Demonstrate Modal Window using ngContent

Show Modal

Demonstrate Modal Window using

Show Modal



I am here to continue the discussion around [AngularJS 2.0](#). So far, we have discussed about data binding, input properties, output properties, pipes, viewchild, and also about directives in Angular 2.0. Now in this article, I will discuss about the route concept in Angular 2.0. Also, in case you did not have a look at the previous articles of this series, go through the links mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJS 2.0 From Beginning - Structural Directives \(Day 7\)](#)
- [AngularJS 2.0 From Beginning - Pipes \(Day 8\)](#)
- [AngularJS 2.0 From Beginning - Viewchild \(Day 9\)](#)
- [AngularJS 2.0 From Beginning - Dynamic Grid \(Day 10\)](#)
- [AngularJS 2.0 From Beginning - Service \(Day 11\)](#)
- [AngularJS 2.0 From Beginning - ngContent \(Day 12\)](#)

In my previous article, I already discussed about the transclusion in Angular 2.0. In this article, we will discuss about the route concept in Angular 2.0.

Angular 2.0 brings many improved modules to the Angular framework including a new router called the Component Router. The component router is totally configurable and feature packed router. Features included are standard view routing, nested child routes, named routes, and route parameters.

Why Routing ?

Routing allows us to specify some aspects of the application's state in the URL. Unlike with Server-side front-end solutions, this is optional - we can build full application without ever changing the URL. Adding routing, however, allows the user to go straight into certain aspects of the application. This is very convenient as it can keep your application linkable and bookmarkable, and allows users to share links with others.

Routing allows you to -

ANGULARJS2

- Maintain the state of application
- Implement modular applications
- Implement the application based on the roles (certain roles have access to certain URLs)

Route Definition Objects

The Routes type is an array of routes that defines the routing for the application. This is where we can set up the expected paths, the components we want to use, and what we want our application to understand them as.

Each route can have different attributes. Some of the common attributes are.

- Path - URL to be shown in the browser when application is on the specific route
- Component - component to be rendered when the application is on the specific route
- redirectTo - redirects route if needed; each route can have either component or redirect attribute defined in the route (covered later in this chapter)
- pathMatch - optional property that defaults to 'prefix'; determines whether to match full URLs or just the beginning. When defining a route with empty path string set pathMatch to 'full', otherwise it will match all paths.
- children - array of route definitions objects representing the child routes of this route (covered later in this chapter)

To use Routes, create an array of route configurations.

```
1. const routes: Routes = [  
2.   { path: 'component-one', component: ComponentOne },  
3.   { path: 'component-two', component: ComponentTwo }  
4. ];
```

RouterModule

RouterModule.forRoot takes the Routes array as an argument and returns a *configured* router module. The following sample shows how we import this module in an app.routes.ts file.

```
1. import { RouterModule, Routes } from '@angular/router';  
2.  
3. const routes: Routes = [  
4.   { path: 'component-one', component: ComponentOne },  
5.   { path: 'component-two', component: ComponentTwo }  
6. ];  
7.  
8. export const routing = RouterModule.forRoot(routes);
```

We, then, import our routing configuration in the root of our application.

```
1. import { routing } from './app.routes';  
2.  
3. @NgModule({  
4.   imports: [  
5.     BrowserModule,  
6.     routing  
7.   ],  
8.   declarations: [  
9.     AppComponent,  
10.    ComponentOne,  
11.    ComponentTwo  
12.  ],  
13.  bootstrap: [ AppComponent ]  
14. })  
15. export class AppModule {
```

Redirecting the Router to Another Route

When your application starts, it navigates to the empty route by default. We can configure the router to redirect to a named route by default.

```

1. export const routes: Routes = [
2.   { path: '', redirectTo: 'component-one', pathMatch: 'full' },
3.   { path: 'component-one', component: ComponentOne },
4.   { path: 'component-two', component: ComponentTwo }
5. ];

```

The pathMatch property, which is required for redirects, tells the router how it should match the URL provided in order to redirect to the specified route. Since pathMatch: full is provided, the router will be redirected to component-one if the entire URL matches the empty path (").

When starting the application, it will now automatically navigate to the route for component-one.

RouterLink

Add links to routes using the RouterLink directive. For example, the following code defines a link to the route at path component-one.

```

1. <a routerLink="/component-one">Component One</a>

```

Alternatively, you can navigate to a route by calling the navigate function on the router.

```

1. this.router.navigate(['/component-one']);

```

Dynamically Adding Route Components

Rather than defining each route's component separately, use RouterOutlet which serves as a component placeholder; Angular dynamically adds the component for the route being activated into the <router-outlet></router-outlet> element.

<router-outlet></router-outlet>

In the above example, the component corresponding to the route specified will be placed after the <router-outlet></router-outlet> element when the link is clicked.

Now, to demonstrate the basic route, please see below.

app.component.cmputer.html

```

1. <div class="panel-body">
2.   <table class="table table-striped table-bordered">
3.     <thead>
4.       <tr>
5.         <th>Name</th>
6.         <th>Company</th>
7.         <th class="text-right">Quantity</th>
8.         <th class="text-right">Price</th>
9.       </tr>
10.    </thead>
11.    <tbody>
12.      <tr *ngFor="let item of data">
13.        <td>{{item.name}}</td>
14.        <td>{{item.company}}</td>
15.        <td class="text-right">{{item.quantity}}</td>

```

ANGULARJS2

```
16.         <td class="text-right">{{item.price | currency}}</td>
17.     </tr>
18. </tbody>
19. </table>
20. </div>
```

app.component.computer.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2.
3. @Component({
4.     moduleId: module.id,
5.     selector: 'computer',
6.     templateUrl: 'app.component.computer.html'
7. })
8.
9. export class ComputerComponent implements OnInit {
10.
11.     private data: Array<any> = [];
12.
13.     constructor() {
14.         this.data = [{ name: 'HP Pavilion 15"', company: 'HP', quantity: '10', price: '42000.00', specification: 'Intel Core i3 2 GB Ram 500 GB HDD with Windows 10' },
15.             { name: 'Lenovo Flex 2"', company: 'Lenovo', quantity: '20', price: '32000.00', specification: 'Intel Core i3 2 GB Ram 500 GB HDD with DOS OS' },
16.             { name: 'Lenovo Yova 500"', company: 'Lenovo', quantity: '20', price: '70000.00', specification: 'Intel Core i7 8 GB Ram 1TB HDD with Windows 8.1' }]
17.     }
18.
19.     ngOnInit(): void {
20.     }
21.
22. }
```

app.component.mobile.html

```
1. <div class="panel-body">
2.     <table class="table table-striped table-bordered">
3.         <thead>
4.             <tr>
5.                 <th>Name</th>
6.                 <th>Company</th>
7.                 <th class="text-right">Quantity</th>
8.                 <th class="text-right">Price</th>
9.             </tr>
10.        </thead>
11.        <tbody>
12.            <tr *ngFor="let item of data">
13.                <td>{{item.name}}</td>
14.                <td>{{item.company}}</td>
15.                <td class="text-right">{{item.quantity}}</td>
16.                <td class="text-right">{{item.price | currency: 'INR': true}}</td>
17.            </tr>
18.        </tbody>
19.    </table>
20. </div>
```

app.component.mobile.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2.
3. @Component({
4.     moduleId: module.id,
5.     selector: 'mobile',
```

ANGULARJS2

```
6.     templateUrl: 'app.component.mobile.html'
7.   })
8.
9.   export class MobileComponent implements OnInit {
10.
11.     private data: Array<any> = [];
12.
13.     constructor() {
14.       this.data = [{ name: 'Galaxy Tab 3', company: 'Samsung', quantity: '10', price: '25000.00' },
15.
16.         { name: 'Galaxy Tab 5', company: 'Samsung', quantity: '50', price: '55000.00' },
17.         { name: 'G4', company: 'LG', quantity: '10', price: '40000.00' },
18.         { name: 'Canvas 3', company: 'Micromax', quantity: '25', price: '18000.00' }];
19.     }
20.
21.     ngOnInit(): void {
22.     }
23. }
```

app.component.tv.html

```
1. <div class="panel-body">
2.   <table class="table table-striped table-bordered">
3.     <thead>
4.       <tr>
5.         <th>Name</th>
6.         <th>Company</th>
7.         <th class="text-right">Quantity</th>
8.         <th class="text-right">Price</th>
9.       </tr>
10.    </thead>
11.    <tbody>
12.      <tr *ngFor="let item of data">
13.        <td>{{item.name}}</td>
14.        <td>{{item.company}}</td>
15.        <td class="text-right">{{item.quantity}}</td>
16.        <td class="text-right">{{item.price | currency}}</td>
17.      </tr>
18.    </tbody>
19.  </table>
20. </div>
```

app.component.tv.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'tv',
6.   templateUrl: 'app.component.tv.html'
7. })
8.
9. export class TvComponent implements OnInit {
10.
11.   private data: Array<any> = [];
12.
13.   constructor() {
14.     this.data = [{ name: 'LED TV 20"', company: 'Samsung', quantity: '10', price: '11000.00' },
15.       { name: 'LED TV 24"', company: 'Samsung', quantity: '50', price: '15000.00' },
16.       { name: 'LED TV 32"', company: 'LG', quantity: '10', price: '32000.00' },
17.       { name: 'LED TV 48"', company: 'SONY', quantity: '25', price: '28000.00' }];
18.   }
19.
20.   ngOnInit(): void {
```

ANGULARJS2

```
21.     }  
22.  
23. }
```

app.component.home.html

```
1. <div class="row">  
2.   <div class="panel -body">  
3.     Home Page  
4.   <br />  
5.   <h3 class="panel -heading"><span>{{message}}</span></h3>  
6. </div>  
7. </div>
```

app.component.home.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';  
2.  
3. @Component({  
4.   moduleId: module.id,  
5.   selector: 'home',  
6.   templateUrl: 'app.component.home.html'  
7. })  
8.  
9. export class HomeComponent implements OnInit {  
10.  
11.   private message: string = '';  
12.   constructor() {  
13.     this.message = 'Click link to move other page';  
14.   }  
15.  
16.   ngOnInit(): void {  
17.   }  
18.  
19. }
```

app.component.homepage.html

```
1. <div class="rowDiv panel panel -primary">  
2.   <h2 class="panel -heading">Demonstration of Angular 2.0 Basic Route</h2>  
3.   <table style="width: 40%; ">  
4.     <tr class="table-bordered">  
5.       <td><a routerLink="/home" class="btn-block">Home</a></td>  
6.       <td><a routerLink="/mobile">Mobile</a></td>  
7.       <td><a routerLink="/tv">TV</a></td>  
8.       <td><a routerLink="/computer">Computers</a></td>  
9.     </tr>  
10.   </table>  
11. </div>  
12. <div class="rowDiv navbar-form">  
13.   <router-outlet></router-outlet>  
14. </div>
```

app.component.homepage.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';  
2.  
3. @Component({  
4.   moduleId: module.id,  
5.   selector: 'home-page',  
6.   templateUrl: 'app.component.homepage.html'
```


ANGULARJS2

```
7. })
8.
9. export class HomePageComponent implements OnInit {
10.
11.     constructor() {
12.     }
13.
14.     ngOnInit(): void {
15.     }
16.
17. }
```

app.routes.ts

```
1. import { Routes, RouterModule } from '@angular/router';
2. import { HomeComponent } from './app.component.home';
3. import { MobileComponent } from './app.component.mobile';
4. import { TvComponent } from './app.component.tv';
5. import { ComputerComponent } from './app.component.computer';
6.
7. export const routes: Routes = [
8.     { path: '', redirectTo: 'home', pathMatch: 'full' },
9.     { path: 'home', component: HomeComponent },
10.    { path: 'tv', component: TvComponent },
11.    { path: 'mobile', component: MobileComponent },
12.    { path: 'computer', component: ComputerComponent }
13. ];
14. ];
15.
16. export const appRoutingProviders: any[] = [
17.
18. ];
19.
20. export const routing = RouterModule.forRoot(routes);
```

app.module.ts

```
1. import { NgModule, NO_ERRORS_SCHEMA } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { FormsModule } from "@angular/forms";
4.
5. import { routing, appRoutingProviders } from './src/app.routes';
6. import { HomePageComponent } from './src/app.component.homepage';
7. import { HomeComponent } from './src/app.component.home';
8. import { MobileComponent } from './src/app.component.mobile';
9. import { TvComponent } from './src/app.component.tv';
10. import { ComputerComponent } from './src/app.component.computer';
11.
12. @NgModule({
13.     imports: [BrowserModule, FormsModule, routing],
14.     declarations: [HomePageComponent, HomeComponent, MobileComponent, TvComponent, ComputerComponent]
15. },
16. {
17.     bootstrap: [HomePageComponent],
18.     schemas: [NO_ERRORS_SCHEMA],
19.     providers: [appRoutingProviders],
20. })
21. export class AppModule { }
```

main.ts

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
3. import { AppModule } from './app.module';
```

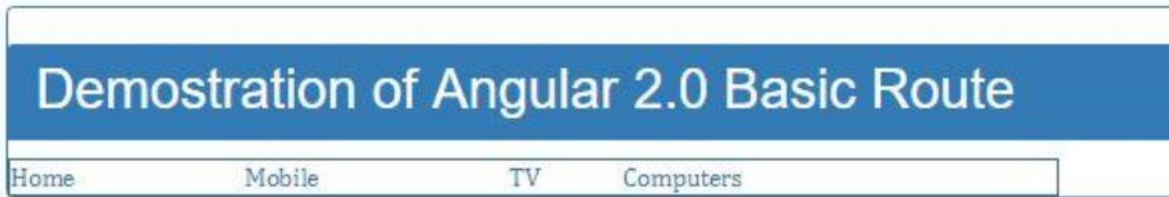
ANGULARJS2

```
4.  
5. const platform = platformBrowserDynamic();  
6. platform.bootstrapModule(AppModule);
```

index.html

```
1. <!DOCTYPE html>  
2. <html>  
3. <head>  
4.   <title>Angular2 - Basic router example</title>  
5.   <meta charset="UTF-8">  
6.   <meta name="viewport" content="width=device-width, initial-scale=1">  
7.   <link href="../../../resources/style/bootstrap.css" rel="stylesheet" />  
8.   <link href="../../../resources/style/style1.css" rel="stylesheet" />  
9.   <!-- Polyfill(s) for older browsers -->  
10.  <script src="../../../resources/js/jquery-2.1.1.js"></script>  
11.  <script src="../../../resources/js/bootstrap.js"></script>  
12.  
13.  <script src="../../../node_modules/core-js/client/shim.min.js"></script>  
14.  <script src="../../../node_modules/zone.js/dist/zone.js"></script>  
15.  <script src="../../../node_modules/reflect-metadata/Reflect.js"></script>  
16.  <script src="../../../node_modules/systemjs/dist/system.src.js"></script>  
17.  <script src="../../../systemjs.config.js"></script>  
18.  <script>  
19.    System.import('app').catch(function (err) { console.error(err); });  
20.  </script>  
21.  <!-- Set the base href, demo only! In your app: <base href="/" -->  
22.  <script>document.write('<base href="' + document.location + '" />');</script>  
23. </head>  
24. <body>  
25.   <home-page>Loading</home-page>  
26. </body>  
27. </html>
```

Now, run the code. The output is shown below.



Home Page

Click link to move other page

Demostration of Angular 2.0 Basic Route			
Home	Mobile	TV	Computers

Name	Company	Quantity	Price
Galaxy Tab 3	Samsung	10	₹25,000.00
Galaxy Tab 5	Samsung	50	₹55,000.00
G4	LG	10	₹40,000.00
Canvas 3	Micromax	25	₹18,000.00

I am here to continue the discussion around [AngularJS 2.0](#). So far, we have discussed about data binding, input properties, output properties, pipes, viewchild, directives, and services including basic routes in Angular 2.0. Now in this article, I will discuss how to pass parameters within routes and also nested routes in Angular 2.0. In case you did not have a look at the previous articles of this series, go through the links mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJS 2.0 From Beginning - Structural Directives \(Day 7\)](#)
- [AngularJS 2.0 From Beginning - Pipes \(Day 8\)](#)
- [AngularJS 2.0 From Beginning - Viewchild \(Day 9\)](#)
- [AngularJS 2.0 From Beginning - Dynamic Grid \(Day 10\)](#)
- [AngularJS 2.0 From Beginning - Service \(Day 11\)](#)
- [AngularJS 2.0 From Beginning - ngContent \(Day 12\)](#)
- [AngularJS 2.0 From Beginning - Route Part I \(Day 13\)](#)

In my previous article, I already discussed about the basic route in Angular 2.0. Now in this article, we discuss about the child route or nested route in Angular 2.0.

What is Nested Child Routes?

Child/Nested routing is a powerful new feature in the new Angular router. We can think of our application as a tree structure, components nested in more components. We can think the same way with our routes and URLs.

ANGULARJS2

So, we have the following routes, / and /about. Maybe our about page is extensive and there are a couple of different views we would like to display as well. The URLs would look something like /about and /about/ item. The first route would be the default about page but more routes would offer another view with more details.

Defining Child Routes

When some routes are only be accessible and viewed within other routes, it may be appropriate to create them as child routes.

For example -

The product details page may have a tabbed navigation section that shows the product overview by default. When the user clicks the "Technical Specs" tab, the section shows the specs instead.

If the user clicks on the product with ID 3, we want to show the product details page with the overview.

```
1. localhost: 3000/product-details/3/overview
```

When the user clicks "Technical Specs",

```
1. localhost: 3000/product-details/3/specs
```

Overview and specs are child routes of product-details/:id. They are only reachable within product details.

Passing Optional Parameters

Query parameters allow you to pass optional parameters to a route, such as pagination information.

For example, on a route with a paginated list, the URL might look like the following to indicate that we've loaded the second page.

```
1. localhost: 3000/product-list?page=2
```

The key difference between query parameters and [route parameters](#) is that route parameters are essential to determine route whereas query parameters are optional.

Passing Query Parameters

Use the [queryParams] directive along with [routerLink] to pass query parameters. For example,

```
1. <a [routerLink]="['product-list']" [queryParams]='{ page: 99 }">Go to Page 99</a>
```

Alternatively, we can navigate programmatically using the Router Service.

```
1. goToPage(pageNum) {  
2.  
3.   this.router.navigate(['/product-list'], { queryParams: { page: pageNum } });  
4.  
5. }
```

Reading Query Parameters

Similar to reading [route parameters](#), the Router service returns an [Observable](#) we can subscribe to to read the query parameters.

ANGULARJS2

```
1. import { Component } from '@angular/core';
2. import { ActivatedRoute, Router } from '@angular/router';
3.
4. @Component({
5.   selector: 'product-list',
6.   template: `<!-- Show product list -->`
7. })
8. export default class ProductList {
9.   constructor(
10.    private route: ActivatedRoute,
11.    private router: Router) {}
12.
13.   ngOnInit() {
14.     this.sub = this.route
15.       .queryParams
16.       .subscribe(params => {
17.         // Defaults to 0 if no query param provided.
18.         this.page = +params['page'] || 0;
19.       });
20.   }
21.
22.   ngOnDestroy() {
23.     this.sub.unsubscribe();
24.   }
25.
26.   nextPage() {
27.     this.router.navigate(['product-list'], { queryParams: { page: this.page + 1 } });
28.   }
29. }
```

For demonstrating this concept, we simply replicate the program of the previous article. Here, we change one thing, that is, computer component will contain two more sub components called desktop and laptop which will be loaded via child route concept.

For doing this, we need to add the following files.

app.component.desktop.html

```
1. <div class="panel-body">
2.   <table class="table table-striped table-bordered">
3.     <thead>
4.       <tr>
5.         <th>Name</th>
6.         <th>Company</th>
7.         <th class="text-right">Quantity</th>
8.         <th class="text-right">Price</th>
9.       </tr>
10.    </thead>
11.    <tbody>
12.      <tr *ngFor="let item of data">
13.        <td>{{item.name}}</td>
14.        <td>{{item.company}}</td>
15.        <td class="text-right">{{item.quantity}}</td>
16.        <td class="text-right">{{item.price | currency}}</td>
17.      </tr>
18.    </tbody>
19.  </table>
20. </div>
```

app.component.desktop.ts

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

ANGULARJS2

```
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'desktop',
6.   templateUrl: 'app.component.desktop.html'
7. })
8.
9. export class DesktopComponent implements OnInit {
10.
11.   private data: Array<any> = [];
12.
13.   constructor() {
14.     this.data = [{ name: 'HP Pavilion 15"', company: 'HP', quantity: '10', price: '42000.00', specification: 'Intel Core i3 2 GB Ram 500 GB HDD with Windows 10' }]
15.   }
16.
17.   ngOnInit(): void {
18.   }
19.
20. }
```

app.component.laptop.html

```
1. <div class="panel-body">
2.   <table class="table table-striped table-bordered">
3.     <thead>
4.       <tr>
5.         <th>Name</th>
6.         <th>Company</th>
7.         <th class="text-right">Quantity</th>
8.         <th class="text-right">Price</th>
9.       </tr>
10.    </thead>
11.    <tbody>
12.      <tr *ngFor="let item of data">
13.        <td>{{item.name}}</td>
14.        <td>{{item.company}}</td>
15.        <td class="text-right">{{item.quantity}}</td>
16.        <td class="text-right">{{item.price | currency}}</td>
17.      </tr>
18.    </tbody>
19.  </table>
20. </div>
```

app.component.laptop.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'laptop',
6.   templateUrl: 'app.component.laptop.html'
7. })
8.
9. export class LaptopComponent implements OnInit {
10.
11.   private data: Array<any> = [];
12.
13.   constructor() {
14.     this.data = [
15.       { name: 'Lenovo Flex 2"', company: 'Lenovo', quantity: '20', price: '32000.00', specification: 'Intel Core i3 2 GB Ram 500 GB HDD with DOS OS' },

```

ANGULARJS2

```
16.      { name: 'Lenovo Yova 500"', company: 'Lenovo', quantity: '20', price: '70000.00', specification: 'Intel Core i7 8 GB Ram 1TB HDD with Windows 8.1' }]
17.    }
18.
19.    ngOnInit(): void {
20.    }
21.
22. }
```

app.component.computer.html

```
1. <div>
2.   <h2>Computer Details</h2>
3.   <h3>Special Discount Rate : {{rebate}}</h3>
4.   <table style="width: 20%; ">
5.     <tr class="table-bordered">
6.       <td><a class="btn-block" (click)="fnNavigateUrl('desktop')">Desktop</a></td>
7.       <td><a (click)="fnNavigateUrl('laptop')">Laptop</a></td>
8.     </tr>
9.   </table>
10.  <div style="color: red; margin-top: 1rem;">
11.    Nested Component Route Outlet
12.  </div>
13.  <div class="rowDiv navbar-form">
14.    <router-outlet></router-outlet>
15.  </div>
16. </div>
```

app.component.computer.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2. import { Router, ActivatedRoute } from '@angular/router';
3.
4. @Component({
5.   moduleId: module.id,
6.   selector: 'computer-details',
7.   templateUrl: 'app.component.computer.html'
8. })
9.
10. export class ComputerComponent implements OnInit {
11.
12.   private data: Array<any> = [];
13.   private rebate;
14.   private sub: any;
15.
16.   constructor(private _router: Router, private route: ActivatedRoute) {
17.   }
18.
19.   ngOnInit(): void {
20.     this.sub = this.route.params.subscribe(params => {
21.       this.rebate = +params['id'];
22.     });
23.   }
24.
25.   private fnNavigateUrl(key: string): void {
26.     this._router.navigate(['/computer/' + key]);
27.   }
28.
29.   private ngOnDestroy() {
30.     this.sub.unsubscribe();
31.   }
32. }
```

ANGULARJS2

app.component.mobile.html

```
1. <div class="panel-body">
2.   <table class="table table-striped table-bordered">
3.     <thead>
4.       <tr>
5.         <th>Name</th>
6.         <th>Company</th>
7.         <th class="text-right">Quantity</th>
8.         <th class="text-right">Price</th>
9.       </tr>
10.    </thead>
11.    <tbody>
12.      <tr *ngFor="let item of data">
13.        <td>{{item.name}}</td>
14.        <td>{{item.company}}</td>
15.        <td class="text-right">{{item.quantity}}</td>
16.        <td class="text-right">{{item.price | currency: 'INR': true}}</td>
17.      </tr>
18.    </tbody>
19.  </table>
20. </div>
```

app.component.mobile.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2.
3. @Component({
4.   moduleId: module.id,
5.   selector: 'mobile',
6.   templateUrl: 'app.component.mobile.html'
7. })
8.
9. export class MobileComponent implements OnInit {
10.
11.   private data: Array<any> = [];
12.
13.   constructor() {
14.     this.data = [{ name: 'Galaxy Tab 3', company: 'Samsung', quantity: '10', price: '25000.00' },
15.       { name: 'Galaxy Tab 5', company: 'Samsung', quantity: '50', price: '55000.00' },
16.       { name: 'G4', company: 'LG', quantity: '10', price: '40000.00' },
17.       { name: 'Canvas 3', company: 'Micromax', quantity: '25', price: '18000.00' }];
18.   }
19.
20.   ngOnInit(): void {
21.   }
22.
23. }
```

app.component.tv.html

```
1. <div class="panel-body">
2.   <table class="table table-striped table-bordered">
3.     <thead>
4.       <tr>
5.         <th>Name</th>
6.         <th>Company</th>
7.         <th class="text-right">Quantity</th>
8.         <th class="text-right">Price</th>
9.       </tr>
10.    </thead>
```


ANGULARJS2

```
11.         <tbody>
12.             <tr *ngFor="let item of data">
13.                 <td>{{item.name}}</td>
14.                 <td>{{item.company}}</td>
15.                 <td class="text-right">{{item.quantity}}</td>
16.                 <td class="text-right">{{item.price | currency}}</td>
17.             </tr>
18.         </tbody>
19.     </table>
20. </div>
```

app.component.tv.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2.
3. @Component({
4.     moduleId: module.id,
5.     selector: 'tv',
6.     templateUrl: 'app.component.tv.html'
7. })
8.
9. export class TvComponent implements OnInit {
10.
11.     private data: Array<any> = [];
12.
13.     constructor() {
14.         this.data = [{ name: 'LED TV 20"', company: 'Samsung', quantity: '10', price: '11000.00' },
15.             { name: 'LED TV 24"', company: 'Samsung', quantity: '50', price: '15000.00' },
16.             { name: 'LED TV 32"', company: 'LG', quantity: '10', price: '32000.00' },
17.             { name: 'LED TV 48"', company: 'SONY', quantity: '25', price: '28000.00' }];
18.     }
19.
20.     ngOnInit(): void {
21.     }
22.
23. }
```

app.component.home.html

```
1. <div class="row">
2.     <div class="panel-body">
3.         Home Page
4.     <br />
5.     <h3 class="panel-heading"><span>{{message}}</span></h3>
6. </div>
7. </div>
```

app.component.home.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2.
3. @Component({
4.     moduleId: module.id,
5.     selector: 'home',
6.     templateUrl: 'app.component.home.html'
7. })
8.
9. export class HomeComponent implements OnInit {
10.
11.     private message: string = '';
12.     constructor() {
13.         this.message = 'Click link to move other page';
```

ANGULARJS2

```
14.     }
15.
16.     ngOnInit(): void {
17.     }
18.
19. }
```

app.component.homepage.html

```
1. <div class="rowDiv panel panel-primary">
2.   <h2 class="panel-heading">Demonstration of Angular 2.0 Basic Route</h2>
3.   <table style="width: 40%; ">
4.     <tr class="table-bordered">
5.       <td><a class="btn-block" (click)="fnNavigateUrl('home')">Home</a></td>
6.       <td><a (click)="fnNavigateUrl('mobile')">Mobile</a></td>
7.       <td><a (click)="fnNavigateUrl('tv')">TV</a></td>
8.       <td><a (click)="fnNavigateUrl('computer')">Computers</a></td>
9.     </tr>
10.  </table>
11. </div>
12. <div class="rowDiv navbar-form">
13.   <router-outlet></router-outlet>
14. </div>
```

app.component.homepage.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2. import { Router } from '@angular/router';
3.
4. @Component({
5.   moduleId: module.id,
6.   selector: 'home-page',
7.   templateUrl: 'app.component.homepage.html'
8. })
9.
10. export class HomePageComponent implements OnInit {
11.
12.   constructor(private _router: Router) {
13.   }
14.
15.   ngOnInit(): void {
16.   }
17.
18.   private fnNavigateUrl(key: string): void {
19.     if (key == 'computer') {
20.       this._router.navigate(['/' + key, 10]);
21.     }
22.     else
23.       this._router.navigate(['/' + key]);
24.   }
25. }
```

app.routes.ts

```
1. import { Routes, RouterModule } from '@angular/router';
2. import { HomeComponent } from './app.component.home';
3. import { MobileComponent } from './app.component.mobile';
4. import { TvComponent } from './app.component.tv';
5. import { ComputerComponent } from './app.component.computer';
6. import { DesktopComponent } from './app.component.desktop';
7. import { LaptopComponent } from './app.component.laptop';
```

ANGULARJS2

```
8.
9. export const routes: Routes = [
10.   { path: '', redirectTo: 'home', pathMatch: 'full' },
11.   { path: 'home', component: HomeComponent },
12.   { path: 'tv', component: TvComponent },
13.   { path: 'mobile', component: MobileComponent },
14.   {
15.     path: 'computer/:id', component: ComputerComponent,
16.     children: [
17.       { path: '', redirectTo: 'computer/laptop', pathMatch: 'full' },
18.       { path: 'computer/desktop', component: DesktopComponent },
19.       { path: 'computer/laptop', component: LaptopComponent }
20.     ]
21.   }
22. ];
23.
24. export const appRoutingProviders: any[] = [
25.
26. ];
27.
28. export const routing = RouterModule.forRoot(routes);
```

app.module.ts

```
1. import { NgModule, NO_ERRORS_SCHEMA } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { FormsModule } from "@angular/forms";
4.
5. import { routing, appRoutingProviders } from './src/app.routes';
6. import { HomeComponent } from './src/app.component.homepage';
7. import { HomeComponent } from './src/app.component.home';
8. import { MobileComponent } from './src/app.component.mobile';
9. import { TvComponent } from './src/app.component.tv';
10. import { ComputerComponent } from './src/app.component.computer';
11. import { DesktopComponent } from './src/app.component.desktop';
12. import { LaptopComponent } from './src/app.component.laptop';
13.
14. @NgModule({
15.   imports: [BrowserModule, FormsModule, routing],
16.   declarations: [HomeComponent, MobileComponent, TvComponent, ComputerComponent,
17.     DesktopComponent, LaptopComponent],
18.   bootstrap: [HomeComponent],
19.   schemas: [NO_ERRORS_SCHEMA],
20.   providers: [appRoutingProviders],
21. })
22. export class AppModule { }
```

main.ts

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
3. import { AppModule } from './app.module';
4.
5. const platform = platformBrowserDynamic();
6. platform.bootstrapModule(AppModule);
```

index.html

```
1. <!DOCTYPE html>
2. <html>
3. <head>
```

ANGULARJS2

```
4. <title>Angular2 - Advanced router example</title>
5. <meta charset="UTF-8">
6. <meta name="viewport" content="width=device-width, initial-scale=1">
7. <link href="../../../resources/style/bootstrap.css" rel="stylesheet" />
8. <link href="../../../resources/style/style1.css" rel="stylesheet" />
9. <!-- Polyfill(s) for older browsers -->
10. <script src="../../../resources/js/jquery-2.1.1.js"></script>
11. <script src="../../../resources/js/bootstrap.js"></script>
12.
13. <script src="../../node_modules/core-js/client/shim.min.js"></script>
14. <script src="../../node_modules/zone.js/dist/zone.js"></script>
15. <script src="../../node_modules/reflect-metadata/Reflect.js"></script>
16. <script src="../../node_modules/systemjs/dist/system.src.js"></script>
17. <script src="../../systemjs.config.js"></script>
18. <script>
19.     System.import('app').catch(function (err) { console.error(err); });
20. </script>
21. <!-- Set the base href, demo only! In your app: <base href="/"> -->
22. <script>document.write('<base href="' + document.location + '" />');</script>
23. </head>
24. <body>
25.     <home-page>Loading</home-page>
26. </body>
27. </html>
```

The output of the above code is shown below.

Demostration of Angular 2.0 Basic Route

HomeMobileTVComputers

Computer Details

Special Discount Rate : 10

DesktopLaptop

Nested Component Route Outlet

Name	Company	Quantity	Price
Lenovo Flex 2"	Lenovo	20	USD32,000.00
Lenovo Yoga 500"	Lenovo	20	USD70,000.00

I am here to continue the discussion around [AngularJS 2.0](#). So far, we have discussed about data binding, input properties, output properties, pipes, viewchild, Directives, Services including routes in Angular 2.0. Now, in this article, I will discuss how to create template based form, using Angular form module in Angular 2.0. In case, you did not have a look at the previous articles of this series, go through the links mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJS 2.0 From Beginning - Structural Directives \(Day 7\)](#)
- [AngularJS 2.0 From Beginning - Pipes \(Day 8\)](#)
- [AngularJS 2.0 From Beginning - Viewchild \(Day 9\)](#)
- [AngularJS 2.0 From Beginning - Dynamic Grid \(Day 10\)](#)
- [AngularJS 2.0 From Beginning - Service \(Day 11\)](#)
- [AngularJS 2.0 From Beginning - ngContent \(Day 12\)](#)
- [AngularJS 2.0 From Beginning - Route Part I \(Day 13\)](#)
- [AngularJS 2.0 From Beginning - Route Part 2 \(Day 14\)](#)

Angular 2.0 Forms – What is it?

In today's Web Application, a large category of front end Applications are very much form dependent, especially in the case of large enterprise type development. Most of these Applications contains simply a huge or large form, which contains multiple tabs, dialogs and with non – trivial business validation logic. Forms are very important of part of the Applications. In a component based Application, we always want to split the forms into a small and reusable piece of code, which is stored within the Smart and Dumb components. These components are normally spread over the entire Application, which provides several architectural benefits including flexibility and design changes.

In Angular 2.0 Framework, we have two different mechanisms, which are related to form-binding.

1. Template Driven Form.
2. Reactive or Model Driven Forms.

In this article, we will discuss about Template Driven Forms.

Angular 2 tackles forms via the famous ngModel. The instantaneous two-way data binding of ng-model in Angular 1 is really a life-saver as it allows to transparently keep in synchronization; a form with a view model. Forms are built with this Directive can only be tested in an end to end test because this requires the presence of a DOM, but still this mechanism is very useful and simple to understand. Unlike the case of AngularJs 1.0, ngModel and other form-related Directives are not available by default, we need to explicitly import them in our Application module. To include the form module in our Application, we need to inject the FormModule in our Application and bootstrapped it.

Template Driven Forms features

- Easy to use.
- Suitable for simple scenarios and fails for complex scenarios.
- Similar to Angular 1.0.
- Two way data binding (using [(NgModel)] syntax).
- Minimal component code.
- Automatic track of the form and its data.
- Unit testing is another challenge.

Advantages and Disadvantages of Template Driven Forms

In this simple example, we cannot really see it, but keeping the template as the source of all form validation truth is something that can become pretty hairy rather than quickly doing it.

As we add more and more validator tags to a field or when we start adding complex cross-field validations the readability of the form decreases to the point, where it will be hard to hand it off to a Web designer.

The up-side of this way of handling forms is its simplicity and its probably more than enough to build a very large range of forms.

On the downside, the form validation logic cannot be unit tested. The only way to test this logic is to run an end to end test with a Browser. For example, using a headless Browser like PhantomJs.

To demonstrate the above concept, we will develop the code given below. For the code, add the files given below with the code.

app.component.homepage.html

```
1. <h2>Template Driven Form</h2>
2. <div>
3.     <form #signupForm="ngForm" (ngSubmit)="registerUser(signupForm)">
4.         <table style="width: 60%; border-collapse: collapse;">
```

ANGULARJS2

```
5.         <tr>
6.             <td style="width : 40%; ">
7.                 <label for="username">User Name</label>
8.             </td>
9.             <td style="width : 60%; ">
10.                 <input type="text" name="username" id="username" [(ngModel)]="username" required>
11.             </td>
12.         </tr>
13.         <tr>
14.             <td style="width : 40%; ">
15.                 <label for="email">Email</label>
16.             </td>
17.             <td style="width : 60%; ">
18.                 <input type="text" name="email" id="email" [(ngModel)]="email" required>
19.             </td>
20.         </tr>
21.         <tr>
22.             <td style="width : 40%; ">
23.                 <label for="password">Password</label>
24.             </td>
25.             <td style="width : 60%; ">
26.                 <input type="password" name="password" id="password" [(ngModel)]="password" required>
27.             </td>
28.         </tr>
29.         <tr>
30.             <td style="width : 40%; "></td>
31.             <td style="width : 60%; ">
32.                 <button type="submit">Sign Up</button>
33.             </td>
34.         </tr>
35.     </table>
36. </form>
37. <div *ngIf="showMessage">
38.     <h3>Thanks You {{formData.username}} for registration</h3>
39. </div>
40. </div>
```

app.component.homepage.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2. import { NgForm } from '@angular/forms';
3.
4. @Component({
5.     moduleId: module.id,
6.     selector: 'home-page',
7.     templateUrl: 'app.component.homepage.html'
8. })
9.
10. export class HomePageComponent implements OnInit {
11.
12.     private formData: any = {};
13.     private showMessage: boolean = false;
14.
15.     constructor() {
16.     }
17.
18.     ngOnInit(): void {
19.     }
20.
21.     registerUser(formdata: NgForm) {
22.         this.formData = formdata.value;
23.         this.showMessage = true;
24.     }
25. }
```

ANGULARJS2

app.module.ts

```
1. import { NgModule, NO_ERRORS_SCHEMA } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { FormsModule } from "@angular/forms";
4.
5. import { HomeComponent } from '../src/app.component.homepage';
6.
7. @NgModule({
8.   imports: [BrowserModule, FormsModule],
9.   declarations: [HomeComponent],
10.  bootstrap: [HomeComponent]
11. })
12. export class AppModule { }
```

index.html

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <title>Angular2 - Template Driven Form </title>
5.   <meta charset="UTF-8">
6.   <meta name="viewport" content="width=device-width, initial-scale=1">
7.   <link href="../../resources/style/bootstrap.css" rel="stylesheet" />
8.   <link href="../../resources/style/style1.css" rel="stylesheet" />
9.   <!-- Polyfill(s) for older browsers -->
10.  <script src="../../resources/js/jquery-2.1.1.js"></script>
11.  <script src="../../resources/js/bootstrap.js"></script>
12.
13.  <script src="../../node_modules/core-js/client/shim.min.js"></script>
14.  <script src="../../node_modules/zone.js/dist/zone.js"></script>
15.  <script src="../../node_modules/reflect-metadata/Reflect.js"></script>
16.  <script src="../../node_modules/systemjs/dist/system.src.js"></script>
17.  <script src="../../systemjs.config.js"></script>
18.  <script>
19.    System.import('app').catch(function (err) { console.error(err); });
20.  </script>
21.  <!-- Set the base href, demo only! In your app: <base href="/" -->
22.  <script>document.write('<base href="' + document.location + '" />');</script>
23. </head>
24. <body>
25.   <home-page>Loading</home-page>
26. </body>
27. </html>
```

main.ts

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
3. import { AppModule } from './app.module';
4.
5. const platform = platformBrowserDynamic();
6. platform.bootstrapModule(AppModule);
```

ANGULARJS2

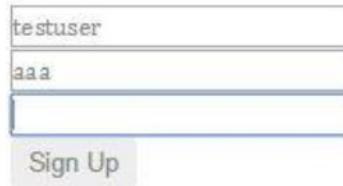
Now, run the code and the output is shown below.

Template Driven Form

User Name

Email

Password



testuser

aaa

Sign Up

I am here to continue the discussion around [AngularJS 2.0](#). So far, we have discussed about data binding, input properties, output properties, pipes, viewchild, Directives, Services including routes and template based form binding, using ngForms in Angular 2.0. Now, in this article, I will discuss how to create model based form, using Angular form module in Angular 2.0. In case you did not have a look at the previous articles of this series, go through the links mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJS 2.0 From Beginning - Structural Directives \(Day 7\)](#)
- [AngularJS 2.0 From Beginning - Pipes \(Day 8\)](#)
- [AngularJS 2.0 From Beginning - Viewchild \(Day 9\)](#)
- [AngularJS 2.0 From Beginning - Dynamic Grid \(Day 10\)](#)
- [AngularJS 2.0 From Beginning - Service \(Day 11\)](#)
- [AngularJS 2.0 From Beginning - ngContent \(Day 12\)](#)
- [AngularJS 2.0 From Beginning - Route Part I \(Day 13\)](#)
- [AngularJS 2.0 From Beginning - Route Part 2 \(Day 14\)](#)
- [AngularJS 2.0 From Beginning - ngForm Part 1 \(Day 15\)](#)

In my previous article, I already discussed about the template driven form binding in Angular 2.0. Now, in this article, we will discuss about the Model Driven or Reactive Form binding in the Angular 2.0.

While using Directives in our templates, it gives us the power of rapid prototyping without too much boilerplate. We are restricted in what we can do. Reactive Forms on the other hand lets us define our form through code and gives us much more flexibility and control over the data validation.

Advantages of Model Driven Forms

- **UNIT TESTABLE**
Since we have the form model defined in our code, we can unit test it.
- **LISTEN TO FORM AND CONTROLS CHANGES**
With Reactive Forms, we can listen to form or control the changes easily. Each form group or form control exposes few events, which we can subscribe to (e.g. statusChanges, valuesChanges, etc).

To begin, we must first ensure that we are working with the right Directives and the right classes in order to take advantage of procedural forms. For this, we need to ensure that `ReactiveFormsModule` was imported in Bootstrap phase of the Application module. This will give us an access to the components, Directives and providers like `FormBuilder`, `FormGroup` and `FormControl`.

ANGULARJS2

FormControl

Note that the FormControl class is assigned to similarly named fields, both on this and in the FormBuilder#group({ }) method. This is mostly for ease of access. By saving the references to the FormControl instances on this, you can access the inputs in the template without having to reference the form itself. The form fields can otherwise be reached in the template by using loginForm.controls.username and loginForm.controls.password. Likewise, any instance of FormControl in this situation can access its parent group by using its .root property (e.g. username.root.controls.password). A FormControl requires two properties: an initial value and a list of validators. Right now, we have no validation.

Validating Reactive Forms

Building from the previous login form, we can quickly and easily add the validation. Angular provides many validators out of the box. They can be imported along with the rest of dependencies for the procedural forms. We are using .valid and .untouched to determine, if we need to show the errors - while the field is required, there is no reason to tell the user that the value is wrong, if the field hasn't been visited yet. For built-in validation, we are calling .hasError() on the form element and we are passing a string, which represents the validator function , which we included. The error message only displays, if this test returns true.

Reactive Forms Custom Validation

As useful as the built-in validators are, it is very useful to be able to include your own. Angular allows you to do just that with minimal effort. A simple function takes the FormControl instance and returns null, if everything is fine. If the test fails, it returns an object with an arbitrarily named property. The property name is what will be used for the .hasError() test.

```
1. <div [hidden]="!password.hasError('needsExclamation')">
2.   Your password must have an exclamation mark!
3. </div>
```

Now, demonstrate this concept and write the code given below.

app.component.homepage.html

```
1. <h2>Model Driven Form</h2>
2. <div>
3.   <form [formGroup]="loginForm" (ngSubmit)="registerUser()">
4.     <table style="width: 60%; border-collapse: collapse;">
5.       <tr>
6.         <td style="width: 40%; text-align: right; padding-right: 10px;>
7.           <label for="username">User Name</label>
8.         </td>
9.         <td style="width: 60%; padding-left: 10px;>
10.          <input type="text" name="username" id="username" [formControl]="username">
11.          <div [hidden]="username.valid || username.untouched">
12.            <div>
13.              The following problems have been found with the username:
14.            </div>
15.            <div [hidden]="!username.hasError('minlength')">
16.              Username can not be shorter than 5 characters.
17.            </div>
18.            <div [hidden]="!username.hasError('required')">
19.              Username is required.
20.            </div>
21.          </div>
22.        </td>
23.      </tr>
24.      <tr>
25.        <td style="width: 40%; text-align: right; padding-right: 10px;>
26.          <label for="password">Password</label>
27.        </td>
```

ANGULARJS2

```
28.         <td style="width : 60%; ">
29.             <input type="password" name="password" id="password" [formControl]="password">
30.             <div [hidden]="password.valid || password.untouched">
31.                 <div>
32.                     The following problems have been found with the password:
33.                 </div>
34.
35.                 <div [hidden]="!password.hasError('required')">
36.                     The password is required.
37.                 </div>
38.                 <div [hidden]="!password.hasError('needsExclamation')">
39.                     Your password must have an exclamation mark!
40.                 </div>
41.             </div>
42.         </td>
43.     </tr>
44.     <tr>
45.         <td style="width : 40%; "></td>
46.         <td style="width : 60%; ">
47.             <button type="submit" [disabled]="!loginForm.valid">Log In</button>
48.         </td>
49.     </tr>
50. </table>
51. </form>
52. <div *ngIf="showMessage">
53.     <h3>Thanks You {{formData.username}} for registration</h3>
54. </div>
55. </div>
```

app.component.homepage.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2. import { Validators, FormBuilder, FormControl, FormGroup } from '@angular/forms';
3.
4. @Component({
5.     moduleId: module.id,
6.     selector: 'home-page',
7.     templateUrl: 'app.component.homepage.html'
8. })
9.
10. export class HomePageComponent implements OnInit {
11.
12.     private formData: any = {};
13.
14.     username = new FormControl('', [
15.         Validators.required,
16.         Validators.minLength(5)
17.     ]);
18.
19.     password = new FormControl('', [
20.         Validators.required,
21.         Validators.maxLength(10)
22.     ]);
23.
24.     loginForm: FormGroup = this.builder.group({
25.         username: this.username,
26.         password: this.password
27.     });
28.
29.
30.     private showMessage: boolean = false;
31.
32.     constructor(private builder: FormBuilder) {
33.     }
34.
35.     ngOnInit(): void {
```

ANGULARJS2

```
36.    }
37.
38.    registerUser() {
39.        this.formData = this.loginForm.value;
40.        this.showMessage = true;
41.    }
42. }
43.
44. function hasExclamationMark(input: FormControl) {
45.     const hasExclamation = input.value.indexOf('!') >= 0;
46.
47.     return hasExclamation ? null : { needsExclamation: true };
48. }
```

app.module.ts

```
1. import { NgModule, NO_ERRORS_SCHEMA } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { ReactiveFormsModule } from "@angular/forms";
4.
5. import { HomeComponent } from './src/app.component.homepage';
6.
7. @NgModule({
8.     imports: [BrowserModule, ReactiveFormsModule],
9.     declarations: [HomeComponent],
10.    bootstrap: [HomeComponent]
11. })
12. export class AppModule { }
```

index.html

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.     <title>Angular2 - Model Driven Form </title>
5.     <meta charset="UTF-8">
6.     <meta name="viewport" content="width=device-width, initial-scale=1">
7.     <link href="../../resources/style/bootstrap.css" rel="stylesheet" />
8.     <link href="../../resources/style/style1.css" rel="stylesheet" />
9.     <!-- Polyfill(s) for older browsers -->
10.    <script src="../../resources/js/jquery-2.1.1.js"></script>
11.    <script src="../../resources/js/bootstrap.js"></script>
12.
13.    <script src="../../node_modules/core-js/client/shim.min.js"></script>
14.    <script src="../../node_modules/zone.js/dist/zone.js"></script>
15.    <script src="../../node_modules/reflect-metadata/Reflect.js"></script>
16.    <script src="../../node_modules/systemjs/dist/system.src.js"></script>
17.    <script src="../../systemjs.config.js"></script>
18.    <script>
19.        System.import('app').catch(function (err) { console.error(err); });
20.    </script>
21.    <!-- Set the base href, demo only! In your app: <base href="/"> -->
22.    <script>document.write('<base href="' + document.location + '" />');</script>
23. </head>
24. <body>
25.     <home-page>Loading</home-page>
26. </body>
27. </html>
```

main.ts

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
3. import { AppModule } from './app.module';
```

ANGULARJS2

```
4.  
5. const platform = platformBrowserDynamic();  
6. platform.bootstrapModule(AppModule);
```

Now, run the code and the output is shown below.

Model Driven Form

User Name

Password



Thanks You debasis for registration

I am here to continue the discussion around [AngularJS 2.0](#). In my previous article, I already discussed about the model driven form binding in Angular 2.0. Now, in this article, we will discuss about http module or how to call external APIs in the Angular 2.0. In case, you did not have a look at the previous articles of this series, go through the links mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJS 2.0 From Beginning - Structural Directives \(Day 7\)](#)
- [AngularJS 2.0 From Beginning - Pipes \(Day 8\)](#)
- [AngularJS 2.0 From Beginning - Viewchild \(Day 9\)](#)
- [AngularJS 2.0 From Beginning - Dynamic Grid \(Day 10\)](#)
- [AngularJS 2.0 From Beginning - Service \(Day 11\)](#)
- [AngularJS 2.0 From Beginning - ngContent \(Day 12\)](#)
- [AngularJS 2.0 From Beginning - Route Part I \(Day 13\)](#)
- [AngularJS 2.0 From Beginning - Route Part 2 \(Day 14\)](#)
- [AngularJS 2.0 From Beginning - ngForm Part 1 \(Day 15\)](#)
- [AngularJS 2.0 From Beginning - ngForm Part 2 \(Day 16\)](#)

Angular 2 introduces many innovative concepts like performance improvements, Component Routing, sharpened Dependency Injection (DI), lazy loading, async templating, mobile development with Native Script; all linked with a solid tooling and excellent testing support. Making HTTP requests in Angular 2 apps looks somewhat different then what we're used to from Angular 1.x, a key difference being that Angular 2's Http returns observables.

It is very clear to us that Angular 2.0 always look and feels different compared to Angular 1.x. In case of Http API calling, the same scenario occurred. The \$http Service, which Angular 1.x provides us works very nicely in most of the cases. Angular 2.0 Http requires us to learn some new concept or mechanism, including how to work with observables.

Reactive Extensions for JavaScript (RxJS) is a reactive streams library, which allows you to work with Observables. RxJS combines Observables, Operators and Schedulers, so we can subscribe to streams and react to changes, using composable operations.

ANGULARJS2

Differences between Angular 1.x \$http and Angular 2 Http

Angular 2's Http API calling again provides a fairly straightforward way of handling the requests. For starters, HTTP calls in Angular 2 by default return observables through [RxJS](#), whereas \$http in Angular 1.x returns Promises. Using observable streams, gives us the benefit of greater flexibility, when it comes to handling the responses coming from HTTP requests. For example, we have the potential of tapping into useful RxJS operators like retry, so that a failed HTTP request is automatically re-sent, which is useful for the cases, where the users have poor or intermittent network communication.

In Angular 2, Http is accessed as an injectable class from angular2/http and, just like other classes, we import it when we want to use it in our components. Angular 2 also comes with a set of injectable providers for Http, which are imported via HTTP_PROVIDERS. With these , we get the providers such as RequestOptions and ResponseOptions, which allows us to modify the requests and the responses by extending the base class for each. In Angular 1.x, we would do this by providing a transformRequest or transformResponse function to our \$httpoptions.

Observables vs Promises

When used with Http, both implementations provide an easy API to handle the requests, but there are some key differences, which makes Observables; a superior alternative.

- Promises only accepts one value unless we compose multiple Promises (Eg: [\\$q.all](#)).
- Promises can't be cancelled.

Angular 2 http module @angular/http exposes a Http Service, which our Application can use to access the Web Services over HTTP. We'll use this utility in our PeopleService Service. We start by importing it together with all the types involved in doing http request:

```
1. import { Http, Response } from '@angular/http';  
2. import { Observable } from 'rxjs/Rx';
```

These are all the types and methods required to make and handle an HTTP request to a Web service:

- *Http*
The Angular 2 http service that provides the API to make HTTP requests with methods corresponding to HTTP verbs like get, post, put, etc
- *Response*
which represents a response from an HTTP service and follows the [fetch API specification](#)
- *Observable*
which is the async pattern used in Angular 2. The concept of observable comes from the [observer design pattern](#) as an object that notifies an interested party of observers when something interesting happens. In RxJs it has been generalized to manage sequences of data or events, to become composable with other observables and to provide a lot of utility functions known as operators that let you achieve amazing stuff.

Angular comes with its own HTTP library, which we can use to call out to external APIs.

When we make calls to an external Server, we want our user to continue to be able to interact with the page i.e. we don't want our page to freeze until the HTTP request returns from the external Server. To achieve this effect, our HTTP requests are asynchronous.

Dealing with an asynchronous code is, historically, more tricky than dealing with synchronous code. In Javascript, there are generally three approaches of dealing with asynchronous code, namely.

1. Callbacks
2. Promises

ANGULARJS2

3. Observables

Now in this article, we will demonstrate how to access or call GET method of a Web API controller to fetch the data.

For this, we first need to add another project of type ASP.NET Web Application and select Web API option from the new dialog box. After creating a project, add the files given below.

Model Class -- Employee.cs

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5.
6. namespace SampleAPI.Models.Sample
7. {
8.     public class Employee
9.     {
10.         public int Id { get; set; }
11.         public string Code { get; set; }
12.         public string Name { get; set; }
13.         public DateTime DOB { get; set; }
14.         public DateTime DOJ { get; set; }
15.         public string Department { get; set; }
16.         public string Designation { get; set; }
17.         public double Salary { get; set; }
18.     }
19. }
```

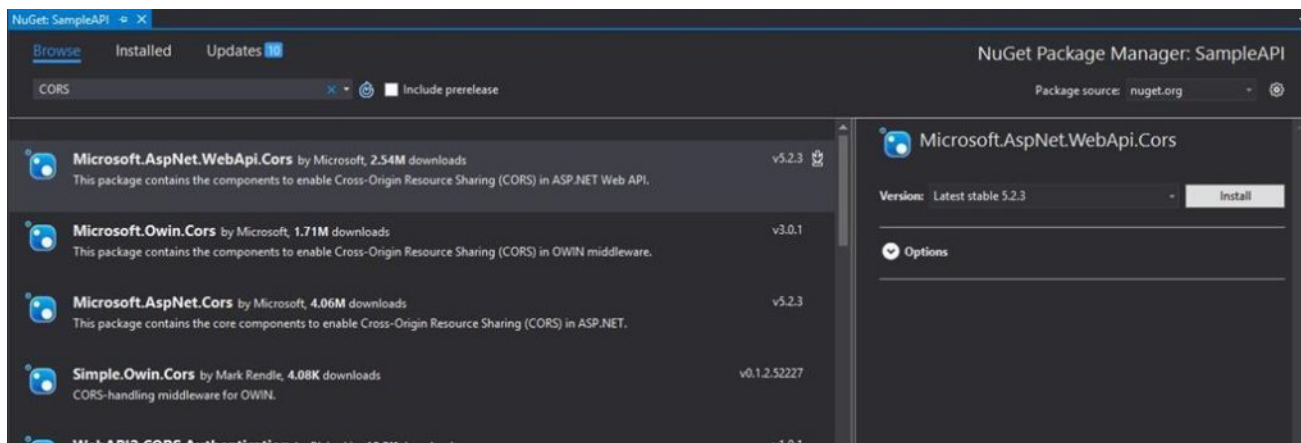
Web API Controller -- EmployeeController.cs

```
1. using SampleAPI.Models.Sample;
2. using System;
3. using System.Collections.Generic;
4. using System.Linq;
5. using System.Net;
6. using System.Net.Http;
7. using System.Web.Http;
8. using System.Web.Http.Description;
9.
10. namespace SampleAPI.Controllers.Sample
11. {
12.     public class EmployeeController : ApiController
13.     {
14.         public EmployeeController()
15.         {
16.
17.         }
18.
19.         [ResponseType(typeof(Employee))]
20.         [HttpGet]
21.         [Route("Employee/GetEmployee")]
22.         public IHttpActionResult GetEmployee()
23.         {
24.             return Ok(this.FetchEmployee());
25.         }
26.
27.         private List<Employee> FetchEmployee()
28.         {
29.             List<Employee> IstData = new List<Employee>();
30.             Employee objEmp = new Employee() { };
31.             objEmp.Id = 1;
32.             objEmp.Code = "A001";
```

ANGULARJS2

```
33.         obj Emp. Name = "RABIN";
34.         obj Emp. DOB = Convert.ToDateTime("10-06-1980");
35.         obj Emp. DOJ = Convert.ToDateTime("01-09-2006");
36.         obj Emp. Department = "ACCOUNTS";
37.         obj Emp. Designation = "CLERK";
38.         obj Emp. Salary = 15000.00;
39.         IstData.Add(obj Emp);
40.
41.         obj Emp = new Employee() { };
42.         obj Emp. Id = 2;
43.         obj Emp. Code = "A002";
44.         obj Emp. Name = "SUJIT";
45.         obj Emp. DOB = Convert.ToDateTime("12-22-1986");
46.         obj Emp. DOJ = Convert.ToDateTime("04-15-2010");
47.         obj Emp. Department = "SALES";
48.         obj Emp. Designation = "MANAGER";
49.         obj Emp. Salary = 35000.00;
50.         IstData.Add(obj Emp);
51.
52.         obj Emp = new Employee() { };
53.         obj Emp. Id = 3;
54.         obj Emp. Code = "A003";
55.         obj Emp. Name = "KAMALESH";
56.         obj Emp. DOB = Convert.ToDateTime("03-22-1982");
57.         obj Emp. DOJ = Convert.ToDateTime("07-15-2006");
58.         obj Emp. Department = "ACCOUNTS";
59.         obj Emp. Designation = "CLERK";
60.         obj Emp. Salary = 16000.00;
61.         IstData.Add(obj Emp);
62.
63.         return IstData;
64.     }
65.
66. }
67. }
```

Now, we can run the API project and access the GetEmployee method from the Browser to fetch the data but when we want to access the same Web API method from our Angular 2 project, then an error will occur called cross origin error. To solve this error, we need to install Microsoft ASP.NET Cors in Web API project from NuGet Manager, as shown below.



Now, change WebAPIConfig.cs file of Web AP project, as shown below.

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Net.Http;
```

ANGULARJS2

```
5. using System.Web.Http;
6. using Microsoft.Owin.Security.OAuth;
7. using Newtonsoft.Json.Serialization;
8. using System.Web.Http.Cors;
9.
10. namespace SampleAPI
11. {
12.     public static class WebApiConfig
13.     {
14.         public static void Register(HttpConfiguration config)
15.         {
16.             // Web API configuration and services
17.             // Configure Web API to use only bearer token authentication.
18.             config.SuppressDefaultHostAuthentication();
19.             config.Filters.Add(new HostAuthenticationFilter(OAuthDefaults.AuthenticationType));
20.
21.             // Web API routes
22.             config.MapHttpAttributeRoutes();
23.
24.             var cors = new EnableCorsAttribute("*", "*", "*");
25.             config.EnableCors(cors);
26.
27.             config.Routes.MapHttpRoute(
28.                 name: "DefaultApi",
29.                 routeTemplate: "api/{controller}/{id}",
30.                 defaults: new { id = RouteParameter.Optional }
31.             );
32.         }
33.     }
34. }
```

Now, go back to Angular 2 project and create the files given below with the code.

app.component.homepage.html

```
1. <div>
2.     <h3>HTTP Module Sample - Get Data</h3>
3.     <div class="panel panel-default">
4.         <div class="panel-body">
5.             <table class="table table-striped table-bordered">
6.                 <thead>
7.                     <tr>
8.                         <th>Sl No</th>
9.                         <th>Alias</th>
10.                        <th>Employee Name</th>
11.                        <th>Date of Birth</th>
12.                        <th>Join Date</th>
13.                        <th>Department</th>
14.                        <th>Designation</th>
15.                        <th>Salary</th>
16.                    </tr>
17.                </thead>
18.                <tbody>
19.                    <tr *ngFor="let item of data">
20.                        <td>{{item.Id}}</td>
21.                        <td>{{item.Code}}</td>
22.                        <td>{{item.Name}}</td>
23.                        <td>{{item.DOB | date:'shortDate'}}</td>
24.                        <td>{{item.DOJ | date:'mediumDate'}}</td>
25.                        <td>{{item.Department}}</td>
26.                        <td>{{item.Designation}}</td>
27.                        <td>{{item.Salary | currency:'INR':true}}</td>
28.                    </tr>
29.                </tbody>

```


ANGULARJS2

```
30.         </table>
31.         <p>
32.             <button class="btn btn-primary" (click)="loadData()">
33.                 Load Data
34.             </button>
35.         </p>
36.     </div>
37. </div>
38. </div>
```

app.component.homepage.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2. import { Http, Response } from '@angular/http';
3. import 'rxjs/Rx';
4.
5. @Component({
6.     moduleId: module.id,
7.     selector: 'home-page',
8.     templateUrl: 'app.component.homepage.html'
9. })
10.
11. export class HomePageComponent implements OnInit {
12.
13.     private data: Array<any> = [];
14.
15.     constructor(private http: Http) {
16.     }
17.
18.     ngOnInit(): void {
19.     }
20.
21.     private loadData(): void {
22.         debugger;
23.         let self = this;
24.         this.http.request('http://localhost:5201/employee/getemployee')
25.             .subscribe((res: Response) => {
26.                 self.data = res.json();
27.             });
28.     }
29.
30. }
```

app.module.ts

```
1. import { NgModule, NO_ERRORS_SCHEMA } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { ReactiveFormsModule } from '@angular/forms';
4. import { HttpClientModule } from '@angular/http';
5.
6. import { HomePageComponent } from './src/app.component.homepage';
7.
8. @NgModule({
9.     imports: [BrowserModule, ReactiveFormsModule, HttpClientModule],
10.    declarations: [HomePageComponent],
11.    bootstrap: [HomePageComponent]
12. })
13. export class AppModule { }
```

main.ts

ANGULARJS2

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
3. import { AppModule } from './app.module';
4.
5. const platform = platformBrowserDynamic();
6. platform.bootstrapModule(AppModule);
```

index.html

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <title>Angular2 - HTTP Module (GET) </title>
5.   <meta charset="UTF-8">
6.   <meta name="viewport" content="width=device-width, initial-scale=1">
7.   <link href="../../resources/style/bootstrap.css" rel="stylesheet" />
8.   <link href="../../resources/style/style1.css" rel="stylesheet" />
9.   <!-- Polyfill(s) for older browsers -->
10.  <script src="../../resources/js/jquery-2.1.1.js"></script>
11.  <script src="../../resources/js/bootstrap.js"></script>
12.
13.  <script src="../../node_modules/core-js/client/shim.min.js"></script>
14.  <script src="../../node_modules/zone.js/dist/zone.js"></script>
15.  <script src="../../node_modules/reflect-metadata/Reflect.js"></script>
16.  <script src="../../node_modules/systemjs/dist/system.src.js"></script>
17.  <script src="../../systemjs.config.js"></script>
18.  <script>
19.    System.import('app').catch(function (err) { console.error(err); });
20.  </script>
21.  <!-- Set the base href, demo only! In your app: <base href="/"> -->
22.  <script>document.write('<base href="' + document.location + '" />');</script>
23. </head>
24. <body>
25.   <home-page>Loading</home-page>
26. </body>
27. </html>
```

Now, run the code and the output is shown below.

HTTP Module Sample - Get Data

Srl No	Alias	Employee Name	Date of Birth	Join Date	Department	Designation	Salary
1	A001	RABIN	10/6/1980	Jan 9, 2006	ACCOUNTS	CLERK	₹15,000.00
2	A002	SUJIT	12/22/1986	Apr 15, 2010	SALES	MANAGER	₹35,000.00
3	A003	KAMALESH	3/22/1982	Jul 15, 2006	ACCOUNTS	CLERK	₹16,000.00

Load Data

I am here to continue the discussion around [AngularJS 2.0](#). In my previous article, I have already discussed about the get method of http module and how to fetch data from a Web API in Angular 2.0. Now, in this article, we will discuss about http module or how to call Post method of external APIs in the Angular 2.0. In case you did not have a look at the previous articles of this series, go through the links mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)

ANGULARJS2

- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJS 2.0 From Beginning - Structural Directives \(Day 7\)](#)
- [AngularJS 2.0 From Beginning - Pipes \(Day 8\)](#)
- [AngularJS 2.0 From Beginning - Viewchild \(Day 9\)](#)
- [AngularJS 2.0 From Beginning - Dynamic Grid \(Day 10\)](#)
- [AngularJS 2.0 From Beginning - Service \(Day 11\)](#)
- [AngularJS 2.0 From Beginning - ngContent \(Day 12\)](#)
- [AngularJS 2.0 From Beginning - Route Part I \(Day 13\)](#)
- [AngularJS 2.0 From Beginning - Route Part 2 \(Day 14\)](#)
- [AngularJS 2.0 From Beginning - ngForm Part 1 \(Day 15\)](#)
- [AngularJS 2.0 From Beginning - ngForm Part 2 \(Day 16\)](#)
- [AngularJS 2.0 From Beginning - Http Module - Part 1 \(Day 17\)](#)

Http Post

Let's imagine we received the *username* and *password* from a form the user submitted. We would call *authenticate* to log in the user. Once the user is logged in, we will proceed to store the token so we can include it in the following requests.

```
1. http.post(url: string, body: string, options?: RequestOptionsArgs) : Observable<Response>
```

The [http.post](#) signature above reads as follows. We need to provide a URL, a body, both strings, and then optionally an options object. In our example, we are passing the modified *headers* property. [http.post](#) returns an *Observable*, we use [map](#) to extract the JSON object from the response and [subscribe](#). This will setup our stream as soon as it emits the result.

Differences between \$http and angular/http

Angular 2 Http by default returns an *Observable* opposed to a *Promise* ([\\$q module](#)) in *\$http*. This allows us to use more flexible and powerful RxJS operators, like [switchMap](#) ([flatMapLatest](#) in version 4), [retry](#), [buffer](#), [debounce](#), [merge](#) or [zip](#). By using *Observables*, we improve the readability and maintenance of our application, as they can respond gracefully to more complex scenarios involving multiple emitted values opposed to only a one-off single value.

Now, demonstrate the below concept. We first need to develop a Web API with POST method which accepts a model data from UI and stores it in the repository.

Now, for this, first we need to create a model class and then create the Controller.

Employee.cs (Employee Model Class)

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5.
6. namespace SampleAPI.Models.Sample
7. {
8.     public class Employee
9.     {
10.         public int Id { get; set; }
11.         public string Code { get; set; }
12.         public string Name { get; set; }
13.         public DateTime DOB { get; set; }
14.         public DateTime DOJ { get; set; }
```

ANGULARJS2

```
15.         public string Department { get; set; }
16.         public string Designation { get; set; }
17.         public double Salary { get; set; }
18.     }
19. }
```

EmployeeController.cs

```
1. using SampleAPI.Models.Sample;
2. using System;
3. using System.Collections.Generic;
4. using System.Linq;
5. using System.Net;
6. using System.Net.Http;
7. using System.Web.Http;
8. using System.Web.Http.Description;
9.
10. namespace SampleAPI.Controllers.Sample
11. {
12.     public class EmployeeController : ApiController
13.     {
14.         static List<Employee> IstData = new List<Employee>();
15.
16.         public EmployeeController()
17.         {
18.         }
19.
20.         [ResponseType(typeof(Employee))]
21.         [HttpGet]
22.         [Route("Employee/GetEmployee")]
23.         public IHttpActionResult GetEmployee()
24.         {
25.             if (IstData.Count == 0)
26.                 this.FetchEmployee();
27.             return Ok(IstData);
28.         }
29.
30.         [ResponseType(typeof(Employee))]
31.         [HttpPost]
32.         [Route("Employee/AddEmployee")]
33.         public IHttpActionResult AddEmployee(Employee objEmp)
34.         {
35.             return Ok(this.SaveEmployee(objEmp));
36.         }
37.
38.         private List<Employee> FetchEmployee()
39.         {
40.             Employee objEmp = new Employee() { };
41.             objEmp.Id = 1;
42.             objEmp.Code = "A001";
43.             objEmp.Name = "RABIN";
44.             objEmp.DOB = Convert.ToDateTime("10-06-1980");
45.             objEmp.DOJ = Convert.ToDateTime("01-09-2006");
46.             objEmp.Department = "ACCOUNTS";
47.             objEmp.Designation = "CLERK";
48.             objEmp.Salary = 15000.00;
49.             IstData.Add(objEmp);
50.
51.             objEmp = new Employee() { };
52.             objEmp.Id = 2;
53.             objEmp.Code = "A002";
54.             objEmp.Name = "SUJIT";
55.             objEmp.DOB = Convert.ToDateTime("12-22-1986");
56.             objEmp.DOJ = Convert.ToDateTime("04-15-2010");
57.             objEmp.Department = "SALES";
58.             objEmp.Designation = "MANAGER";
```

ANGULARJS2

```
59.         obj Emp. Sal ary = 35000. 00;
60.         IstData. Add(obj Emp);
61.
62.         obj Emp = new Empl oyee() { };
63.         obj Emp. Id = 3;
64.         obj Emp. Code = "A003";
65.         obj Emp. Name = "KAMALESH";
66.         obj Emp. DOB = Convert. ToDateTime("03-22-1982");
67.         obj Emp. DOJ = Convert. ToDateTime("07-15-2006");
68.         obj Emp. Department = "ACCOUNTS";
69.         obj Emp. Designati on = "CLERK";
70.         obj Emp. Sal ary = 16000. 00;
71.         IstData. Add(obj Emp);
72.
73.         return IstData;
74.     }
75.
76.     private bool SaveEmpl oyee(Empl oyee obj Emp)
77.     {
78.         obj Emp. Id = (IstData. OrderByDescendi ng(s => s. Id). Fi rstOrDefaul t()). Id + 1;
79.         IstData. Add(obj Emp);
80.         return true;
81.     }
82. }
83. }
```

In the Web API Controller, we initially stored three records. And after it, when user fills up the employee form and clicks on Submit button, that particular record will be appened along with these three records.

app.component.employee.ts

```
1. import { Component, OnInit, EventEmitter, Output } from '@angul ar/core';
2. import { Http, Response, Headers } from '@angul ar/http';
3. import 'rxjs/Rx';
4.
5. @Component({
6.     moduleId: module. id,
7.     selector: 'empl oyee-add',
8.     templateUrl: 'app. component. empl oyeeadd. html'
9. })
10.
11. export class AddEmpl oyeeComponent implements OnInit {
12.
13.     private _model: any = {};
14.     @Output() private onHi de: EventEmi tter<boolean> = new EventEmi tter<boolean>();
15.
16.     constructor(private http: Http) {
17.     }
18.
19.     ngOnI niti(): void {
20.
21.     }
22.
23.     private onCancel(): void {
24.         thi s. _model = {};
25.         thi s. onHi de. emi t(false);
26.     }
27.
28.     private submi t(): void {
29.         if (thi s. vali date()) {
30.             let self = thi s;
31.             let headers = new Headers();
32.             headers. append('Content-Type', 'appl i cati on/j son');
```

ANGULARJS2

```
33.         this.http.post("http://localhost:5201/employee/AddEmployee", this._model, { headers: head
    ers })
34.             .subscribe((res: Response) => {
35.                 self.onCancel();
36.             });
37.
38.         }
39.     }
40.
41.     private reset(): void {
42.         this._model = {};
43.     }
44.
45.     private validate(): boolean {
46.         let status: boolean = true;
47.         if (typeof (this._model.code) === "undefined") {
48.             alert('Alias is Blank');
49.             status = false;
50.             return;
51.         }
52.         else if (typeof (this._model.name) === "undefined") {
53.             alert('Name is Blank');
54.             status = false;
55.             return;
56.         }
57.         else if (typeof (this._model.dob) === "undefined") {
58.             alert('dob is Blank');
59.             status = false;
60.             return;
61.         }
62.         else if (typeof (this._model.doj) === "undefined") {
63.             alert('DOJ is Blank');
64.             status = false;
65.             return;
66.         }
67.         else if (typeof (this._model.department) === "undefined") {
68.             alert('Department is Blank');
69.             status = false;
70.             return;
71.         }
72.         else if (typeof (this._model.designation) === "undefined") {
73.             alert('Designation is Blank');
74.             status = false;
75.             return;
76.         }
77.         else if (typeof (this._model.salary) === "undefined") {
78.             alert('Salary is Blank');
79.             status = false;
80.             return;
81.         }
82.         return status;
83.     }
84. }
```

app.component.employee.html

```
1. <div class="panel panel-default">
2.     <h4>Provide Employee Details</h4>
3.     <table class="table" style="width: 60%; ">
4.         <tr>
5.             <td>Employee Code</td>
6.             <td><input type="text" [(ngModel)]="_model.code" /></td>
7.         </tr>
8.         <tr>
9.             <td>Employee Name</td>
10.            <td><input type="text" [(ngModel)]="_model.name" /></td>
```

ANGULARJS2

```
11.         </tr>
12.         <tr>
13.             <td>Date of Birth</td>
14.             <td><input type="date" [(ngModel)]="_model.dob" /></td>
15.         </tr>
16.         <tr>
17.             <td>Date of Join</td>
18.             <td><input type="date" [(ngModel)]="_model.doj" /></td>
19.         </tr>
20.         <tr>
21.             <td>Department</td>
22.             <td><input type="text" [(ngModel)]="_model.department" /></td>
23.         </tr>
24.         <tr>
25.             <td>Designation</td>
26.             <td><input type="text" [(ngModel)]="_model.designation" /></td>
27.         </tr>
28.         <tr>
29.             <td>Salary</td>
30.             <td><input type="number" [(ngModel)]="_model.salary" /></td>
31.         </tr>
32.         <tr>
33.             <td></td>
34.             <td>
35.                 <input type="button" value="Submit" (click)="submit()" />
36.                 <input type="button" value="Cancel" (click)="onCancel()" />
37.             </td>
38.         </tr>
39.     </tr>
40. </table>
41. </div>
```

app.component.homepage.ts

```
1. import { Component, OnInit, ViewChild, AfterViewInit } from '@angular/core';
2. import { Http, Response } from '@angular/http';
3. import 'rxjs/Rx';
4.
5. @Component({
6.     moduleId: module.id,
7.     selector: 'home-page',
8.     templateUrl: 'app.component.homepage.html'
9. })
10.
11. export class HomePageComponent implements OnInit {
12.
13.     private data: Array<any> = [];
14.     private showDetails: boolean = true;
15.     private showEmployee: boolean = false;
16.
17.     constructor(private http: Http) {
18.     }
19.
20.     ngOnInit(): void {
21.     }
22.
23.
24.     ngAfterViewInit(): void {
25.         debugger;
26.         this.loadData();
27.     }
28.
29.     private loadData(): void {
30.         let self = this;
31.         this.http.request('http://localhost:5201/employee/getemployee')
32.             .subscribe((res: Response) => {
```

ANGULARJS2

```
33.         debugger;
34.         self.data = res.json();
35.     });
36. }
37.
38. private addEmployee(): void {
39.     this.showDetails = false;
40.     this.showEmployee = true;
41. }
42.
43. private onHide(args: boolean): void {
44.     this.showDetails = !args;
45.     this.showEmployee = args;
46.     this.loadData();
47. }
48. }
```

app.component.homepage.html

```
1. <div>
2.   <h3>HTTP Module Sample - Add and Fetch Data</h3>
3.   <div class="panel panel-default" *ngIf="showDetails">
4.     <div class="panel-body">
5.       <table class="table table-striped table-bordered">
6.         <thead>
7.           <tr>
8.             <th>Serial No</th>
9.             <th>Alias</th>
10.            <th>Employee Name</th>
11.            <th>Date of Birth</th>
12.            <th>Join Date</th>
13.            <th>Department</th>
14.            <th>Designation</th>
15.            <th>Salary</th>
16.          </tr>
17.        </thead>
18.        <tbody>
19.          <tr *ngFor="let item of data">
20.            <td>{{item.Id}}</td>
21.            <td>{{item.Code}}</td>
22.            <td>{{item.Name}}</td>
23.            <td>{{item.DOB | date : 'shortDate'}}</td>
24.            <td>{{item.DOJ | date : 'mediumDate'}}</td>
25.            <td>{{item.Department}}</td>
26.            <td>{{item.Designation}}</td>
27.            <td>{{item.Salary | currency: 'INR' : true}}</td>
28.          </tr>
29.        </tbody>
30.      </table>
31.      <p>
32.        <button class="btn btn-primary" (click)="addEmployee()">
33.          Add Employee
34.        </button>
35.      </p>
36.    </div>
37.  </div>
38.  <div class="panel panel-default" *ngIf="showEmployee">
39.    <employee-add (onHide)="onHide($event);"></employee-add>
40.  </div>
41. </div>
```

main.ts

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
```


ANGULARJS2

```
3. import { AppModule } from './app.module';
4.
5. const platform = platformBrowserDynamic();
6. platform.bootstrapModule(AppModule);
```

app.module.ts

```
1. import { NgModule, NO_ERRORS_SCHEMA } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { FormsModule } from "@angular/forms";
4. import { HttpClientModule } from '@angular/http';
5.
6. import { HomeComponent } from './src/app.component.homepage';
7. import { AddEmployeeComponent } from './src/app.component.employeeadd';
8.
9. @NgModule({
10.   imports: [BrowserModule, FormsModule, HttpClientModule],
11.   declarations: [HomeComponent, AddEmployeeComponent],
12.   bootstrap: [HomeComponent]
13. })
14. export class AppModule { }
```

index.html

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <title>Angular2 - HTTP Module (POST & GET) </title>
5.   <meta charset="UTF-8">
6.   <meta name="viewport" content="width=device-width, initial-scale=1">
7.   <link href="../../resources/style/bootstrap.css" rel="stylesheet" />
8.   <link href="../../resources/style/style1.css" rel="stylesheet" />
9.   <!-- Polyfill(s) for older browsers -->
10.  <script src="../../resources/js/jquery-2.1.1.js"></script>
11.  <script src="../../resources/js/bootstrap.js"></script>
12.
13.  <script src="../../node_modules/core-js/client/shim.min.js"></script>
14.  <script src="../../node_modules/zone.js/dist/zone.js"></script>
15.  <script src="../../node_modules/reflect-metadata/Reflect.js"></script>
16.  <script src="../../node_modules/systemjs/dist/system.src.js"></script>
17.  <script src="../../systemjs.config.js"></script>
18.  <script>
19.    System.import('app').catch(function (err) { console.error(err); });
20.  </script>
21.  <!-- Set the base href, demo only! In your app: <base href="/" -->
22.  <script>document.write('<base href="' + document.location + '" />');</script>
23. </head>
24. <body>
25.   <home-page>Loading</home-page>
26. </body>
27. </html>
```

ANGULARJS2

Now, run the code in browser and you will see the below result.

HTTP Module Sample - Add and Fetch Data

Srl No	Alias	Employee Name	Date of Birth	Join Date	Department	Designation	Salary
1	A001	RABIN	10/6/1980	Jan 9, 2006	ACCOUNTS	CLERK	₹15,000.00
2	A002	SUJIT	12/22/1986	Apr 15, 2010	SALES	MANAGER	₹35,000.00
3	A003	KAMALESH	3/22/1982	Jul 15, 2006	ACCOUNTS	CLERK	₹16,000.00

Add Employee

HTTP Module Sample - Add and Fetch Data

Provide Employee Details

Employee Code

B0098

Employee Name

SUMIT MONDAL

Date of Birth

04/19/1984

Date of Join

03/09/2017

Department

MARKETING

Designation

SALES MAN

Salary

10000

Submit

Cancel

HTTP Module Sample - Add and Fetch Data

Srl No	Alias	Employee Name	Date of Birth	Join Date	Department	Designation	Salary
1	A001	RABIN	10/6/1980	Jan 9, 2006	ACCOUNTS	CLERK	₹15,000.00
2	A002	SUJIT	12/22/1986	Apr 15, 2010	SALES	MANAGER	₹35,000.00
3	A003	KAMALESH	3/22/1982	Jul 15, 2006	ACCOUNTS	CLERK	₹16,000.00
4	B0098	SUMIT MONDAL	4/19/1984	Mar 9, 2017	MARKETING	SALES MAN	₹10,000.00

Add Employee

I am here to continue the discussion around [AngularJS 2.0](#). In my previous article, I have already discussed about the http module including get and post method calling in Angular 2.0. Now, in this article, we will discuss about the observables concept in Angular 2.0. In case, you did not have a look at the previous articles of this series, go through the links mentioned below.

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)

ANGULARJS2

- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJS 2.0 From Beginning - Structural Directives \(Day 7\)](#)
- [AngularJS 2.0 From Beginning - Pipes \(Day 8\)](#)
- [AngularJS 2.0 From Beginning - Viewchild \(Day 9\)](#)
- [AngularJS 2.0 From Beginning - Dynamic Grid \(Day 10\)](#)
- [AngularJS 2.0 From Beginning - Service \(Day 11\)](#)
- [AngularJS 2.0 From Beginning - ngContent \(Day 12\)](#)
- [AngularJS 2.0 From Beginning - Route Part I \(Day 13\)](#)
- [AngularJS 2.0 From Beginning - Route Part 2 \(Day 14\)](#)
- [AngularJS 2.0 From Beginning - ngForm Part 1 \(Day 15\)](#)
- [AngularJS 2.0 From Beginning - ngForm Part 2 \(Day 16\)](#)
- [AngularJS 2.0 From Beginning - Http Module - Part 1 \(Day 17\)](#)
- [AngularJS 2.0 From Beginning - Http Module - Part 2 \(Day 18\)](#)

We already discussed that in Angular 2.0, there are many new features introduced in Angular 2.0. An exciting new feature used with Angular is *Observable*. This isn't an Angular specific feature but rather a proposed standard for managing async data that will be included in the release of ES7. Observables open up a continuous channel of communication in which multiple values of data can be emitted over time. From this, we get a pattern of dealing with data by using array-like operations to parse, modify and maintain the data. Angular uses Observables extensively - you'll see them in the HTTP service and the event system.

In version 2.0, Angular mainly introduces the reactive programming concept based on the observables for dealing the asynchronous processing of data. In Angular 1.x, we basically used promises to handle the asynchronous processing. But still in Angular 2.0, we can still use the promises for the same purpose. The main concept of reactive programming is the observable element which is related to the entity that can be observed. Basically, at normal look, promises and observables seem very similar to each other. Both of them allow us to execute asynchronous processing, register callbacks for both successful and error responses, and also notify or inform us when the result is there.

How to define Observables

Before going into detailed example of observables, first we need to understand how to define an observable object. To create an observable, we can use the create method of the Observable object. A function must be provided as parameter with the code to initialize the observable processing. A function can be returned by this function to cancel the observable.

```
1. var observable = Observable.create((observer) =>
2. {
3.     setTimeout(() => {
4.         observer.next('some event');
5.     }, 500);
6. });
```

Similar to promises, observables can produce several notifications using different methods from the observer.

1. *next*
Emit an event. This can be called several times.
2. *error*
Throw an error. This can be called once and will break the stream. This means that the error callback will be immediately called and no more events or completion can be received.
3. *complete*
Mark the observable as completed. After this, no more events or errors will be handled and provided to corresponding callbacks.

ANGULARJS2

Observables allow us to register callbacks for previously described notifications. The subscribe method tackles this issue. It accepts three callbacks as parameters,

- The onNext callback that will be called when an event is triggered.
- The onError callback that will be called when an error is thrown.
- The onCompleted callback that will be called when the observable completes.

Observable specificities

Observables and promises have many similarities, and in spite of that, observables provide us some new specifications like below -

- *Lazy*
An observable is only enabled when a first observer subscribes. This is a significant difference compared to promises. Due to this, processing to initialize a promise is always executed even if no listener is registered. This means that promises don't wait for subscribers to be ready to receive and handle the response. When creating the promise, the initialization processing is always immediately called. Observables are lazy so we have to subscribe a callback to let them execute their initialization callback.
- *Execute Several Times*
Another particularity of observables is that they can be triggered several times unlike promises which can't be used after they were resolved or rejected.
- *Canceling Observables*
Another characteristic of observables is that they can be cancelled. For this, we can simply return a function within the initialization call of the Observable.create function. We can refactor our initial code to make it possible to cancel the timeout function.

Error Handling

If something unexpected arises, we can raise an error on the Observable stream and use the function reserved for handling errors in our subscribe routine to see what happened.

app.component.homepage.ts

```
1. import { Component, OnInit, ViewChild, AfterViewInit } from '@angular/core';
2. import { Observable } from 'rxjs/Observable';
3.
4. @Component({
5.   moduleId: module.id,
6.   selector: 'home-page',
7.   templateUrl: 'app.component.homepage.html'
8. })
9.
10. export class HomeComponent implements OnInit {
11.
12.   private data: Observable<Array<number>>;
13.   private values: Array<number> = [];
14.   private anyErrors: boolean;
15.   private finished: boolean;
16.   private status: string;
17.
18.   constructor() {
19.   }
20.
21.   ngOnInit(): void {
22.
23.   }
24.
25.   private onClick(): void {
```

ANGULARJS2

```
26.     let self = this;
27.     this.data = Observable.create(observer => {
28.         setTimeout(() => {
29.             observer.next(10);
30.         }, 500);
31.
32.         setTimeout(() => {
33.             observer.next(100);
34.         }, 1000);
35.
36.         setTimeout(() => {
37.             observer.complete();
38.         }, 2000);
39.         this.status = "Started";
40.     });
41.
42.     let subscription = this.data.subscribe(
43.         (value) => {
44.             let data: any = { val: value };
45.             self.values.push(data);
46.         },
47.         error => self.anyErrors = true,
48.         () => self.finished = true
49.     );
50. }
51.
52. private onClick1(): void {
53.     let self = this;
54.     this.data = Observable.create(observer => {
55.         setTimeout(() => {
56.             observer.next(10);
57.         }, 500);
58.
59.         setTimeout(() => {
60.             observer.next(100);
61.         }, 1000);
62.
63.         setTimeout(() => {
64.             observer.complete();
65.         }, 2000);
66.         this.status = "Started";
67.     });
68.
69.     let subscription = this.data.forEach(
70.         (value) => {
71.             let data: any = { val: value };
72.             self.values.push(data);
73.         })
74.     .then(() => self.status = "Ended");
75. }
76. }
```

app.component.homepage.html

```
1. <div>
2.     <h3>Observables Sample</h3>
3.     <b>Angular 2 Sample Component Using Observables!</b>
4.
5.     <h6 style="margin-bottom: 0">Count: </h6>
6.     <div *ngFor="let item of values">{{ item.val }}</div>
7.
8.     <h6 style="margin-bottom: 0">ERRORs: </h6>
9.     <div>Errors: {{anyErrors}}</div>
10.
11.     <h6 style="margin-bottom: 0">FINISHED: </h6>
12.     <div>Finished: {{ finished }}</div>
```

ANGULARJS2

```
13.
14.   <h6 style="margin-bottom: 0">STATUS:</h6>
15.   <div>{{status}}</div>
16.
17.   <button style="margin-top: 2rem;" (click)="onClick()">Init Page</button>
18.   <error-page></error-page>
19. </div>
```

app.component.error.ts

```
1. import { Component, OnInit, ViewChild, AfterViewInit } from '@angular/core';
2. import { Observable } from 'rxjs/Observable';
3.
4. @Component({
5.   moduleId: module.id,
6.   selector: 'error-page',
7.   templateUrl: 'app.component.error.html'
8. })
9.
10. export class ErrorComponent implements OnInit {
11.
12.   private data: Observable<Array<number>>;
13.   private values: Array<number> = [];
14.   private anyErrors: any;
15.
16.   constructor() {
17.   }
18.
19.   ngOnInit(): void {
20.
21.   }
22.
23.   private onClick(): void {
24.     let self = this;
25.     this.data = Observable.create(observer => {
26.       setTimeout(() => {
27.         observer.next(10)
28.       }, 1500);
29.       setTimeout(() => {
30.         observer.error('Hey something bad happened I guess');
31.       }, 2000);
32.       setTimeout(() => {
33.         observer.next(50)
34.       }, 2500);
35.     });
36.
37.     let subscription = this.data.subscribe(
38.       (value) => {
39.         let data: any = { val: value };
40.         self.values.push(data);
41.       },
42.       error => self.anyErrors = "Ended"
43.     );
44.   }
45. }
```

app.component.error.html

```
1. <div>
2.   <h3>Observables Sample - Error handling</h3>
3.   <b>Angular 2 Component using Observables!</b>
4.
5.   <h5 style="margin-bottom: 0">VALUES</h5>
6.   <div *ngFor="let item of values">{{ item.val.toString() }}</div>
7. </div>
```

ANGULARJS2

```
8.      <h5 style="margin-bottom: 0">ERRORS</h5>
9.      <pre><code>{{anyErrors}}</code></pre>
10.
11.      <button style="margin-top: 2rem;" (click)="onClick()">Init Page</button>
12. </div>
```

app.module.ts

```
1. import { NgModule, NO_ERRORS_SCHEMA } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { FormsModule } from '@angular/forms';
4. import { HttpClientModule } from '@angular/http';
5.
6. import { HomeComponent } from './src/app.component.homepage';
7. import { ErrorComponent } from './src/app.component.error';
8.
9. @NgModule({
10.   imports: [BrowserModule, FormsModule, HttpClientModule],
11.   declarations: [HomeComponent, ErrorComponent],
12.   bootstrap: [HomeComponent]
13. })
14. export class AppModule { }
```

index.html

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <title>Angular2 - Observables </title>
5.   <meta charset="UTF-8">
6.   <meta name="viewport" content="width=device-width, initial-scale=1">
7.   <link href="../../resources/style/bootstrap.css" rel="stylesheet" />
8.   <link href="../../resources/style/style1.css" rel="stylesheet" />
9.   <!-- Polyfill(s) for older browsers -->
10.  <script src="../../resources/js/jquery-2.1.1.js"></script>
11.  <script src="../../resources/js/bootstrap.js"></script>
12.
13.  <script src="../../node_modules/core-js/client/shim.min.js"></script>
14.  <script src="../../node_modules/zone.js/dist/zone.js"></script>
15.  <script src="../../node_modules/reflect-metadata/Reflect.js"></script>
16.  <script src="../../node_modules/systemjs/dist/system.src.js"></script>
17.  <script src="../../systemjs.config.js"></script>
18.  <script>
19.    System.import('app').catch(function (err) { console.error(err); });
20.  </script>
21.  <!-- Set the base href, demo only! In your app: <base href="/"> -->
22.  <script>document.write('<base href="' + document.location + '"' />');</script>
23. </head>
24. <body>
25.   <home-page>Loading</home-page>
26. </body>
27. </html>
```

main.t

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
3. import { AppModule } from './app.module';
4.
5. const platform = platformBrowserDynamic();
6. platform.bootstrapModule(AppModule);
```

Now, run the code and get the output as below.

Observables Sample

Angular 2 Sample Component Using Observables!

Count:

10

100

ERRORs:

Errors:

FINISHED:

Finished: true

STATUS:

Started

Init Page

Observables Sample - Error handling

Angular 2 Component using Observables!

VALUES

10

ERRORS

Ended

Init Page

Observables Sample

Angular 2 Sample Component Using Observables!

Count:

10

100

ERRORs:

Errors:

FINISHED:

Finished: true

STATUS:

Started

Init Page

I am here to continue the discussion around [AngularJS 2.0](#). I already discussed about the basic concept of Reactive programming concept or observables in Angular 2.0. Now, in this article, we discuss about the observables concept with http module or form modules in Angular 2.0. In case, you did not have a look at the previous articles of this series, go through the links mentioned below.

ANGULARJS2

- [AngularJS 2.0 From Beginning Introduction of AngularJS 2.0 \(Day 1\)](#)
- [AngularJS 2.0 From Beginning Component \(Day 2\)](#)
- [AngularJS 2.0 From Beginning Data Binding \(Day 3\)](#)
- [AngularJS 2.0 From Beginning Input Data Binding \(Day 4\)](#)
- [AngularJS 2.0 From Beginning - Output Property Binding \(Day 5\)](#)
- [AngularJS 2.0 From Beginning - Attribute Directive \(Day 6\)](#)
- [AngularJS 2.0 From Beginning - Structural Directives \(Day 7\)](#)
- [AngularJS 2.0 From Beginning - Pipes \(Day 8\)](#)
- [AngularJS 2.0 From Beginning - Viewchild \(Day 9\)](#)
- [AngularJS 2.0 From Beginning - Dynamic Grid \(Day 10\)](#)
- [AngularJS 2.0 From Beginning - Service \(Day 11\)](#)
- [AngularJS 2.0 From Beginning - ngContent \(Day 12\)](#)
- [AngularJS 2.0 From Beginning - Route Part I \(Day 13\)](#)
- [AngularJS 2.0 From Beginning - Route Part 2 \(Day 14\)](#)
- [AngularJS 2.0 From Beginning - ngForm Part 1 \(Day 15\)](#)
- [AngularJS 2.0 From Beginning - ngForm Part 2 \(Day 16\)](#)
- [AngularJS 2.0 From Beginning - Http Module - Part 1 \(Day 17\)](#)
- [AngularJS 2.0 From Beginning - Http Module - Part 2 \(Day 18\)](#)
- [AngularJS 2.0 From Beginning - Observables \(Day 19\)](#)

Observables vs Promises

Both Promises and Observables provide us with abstraction that helps us to deal with the asynchronous nature of our Applications. However, there are important differences between the two:

As seen in the example above, Observables can define both the setup and teardown aspects of asynchronous behavior.

Observables are cancellable.

Moreover, Observables can be retried, using one of the retry operators provided by the API, such as `retry` and `retryWhen`. On the other hand, Promises require the caller to have an access to the original function, which returned the Promise in order to have a retry capability.

Observable HTTP Events

A common operation in any Web Application is getting or posting the data to a Server. Angular Applications do this with `Http` library, which previously used Promises to operate in an asynchronous manner. The updated `Http` library now incorporates Observables to trigger the events and getting new data.

Observables Array Operations

In addition to simply iterating over an asynchronous collection, we can perform other operations such as `filter` or `map` and many more as defined in the RxJS API. This is what bridges an Observable with the iterable pattern and lets us conceptualize them as collections.

Here are two really useful array operations - `map` and `filter`. What exactly do these do?

map will create a new array with the results of calling a provided function on every element in this array. In the example given below, we used it to create a new result set by iterating through each item and appending the word 'Mr' abbreviation in front of every employee's name.

filter will create a new array with all the elements that pass the test implemented by a provided function. Here, we have used it to create a new result set by excluding any user whose salary is below 20000. Now, when our subscribe callback gets invoked, the data it receives will be a list of JSON objects, whose id properties are greater than or equal to 20000 salary.

ANGULARJS2

Note the chaining function style and the optional static typing, which comes with TypeScript, which we used in this example. Most importantly functions like filter return an Observable, as in Observables we get other Observables, which are similar to Promises. In order to use map and filter in a chaining sequence, we have flattened the results of our Observable, using flatMap. Since filter accepts an Observable and not an array, we have to convert our array of JSON objects from data.json() to an Observablestream. This is done with flatMap

To demonstrate the above concept, create the files given below with the code.

app.component.homepage.ts

```
1. import { Component, OnInit, ViewChild } from '@angular/core';
2. import { Http, Response } from '@angular/http';
3. import 'rxjs/Rx';
4.
5. @Component({
6.   moduleId: module.id,
7.   selector: 'home-page',
8.   templateUrl: 'app.component.homepage.html'
9. })
10.
11. export class HomeComponent implements OnInit {
12.
13.   private data: Array<any> = [];
14.
15.   constructor(private http: Http) {
16.   }
17.
18.   ngOnInit(): void {
19.   }
20.
21.   private loadData(): void {
22.     this.data = [];
23.     let self = this;
24.     this.http.request('http://localhost:5201/employee/getemployee')
25.       .flatMap((result) => result.json())
26.       .subscribe((result) => {
27.         self.data.push(result);
28.       });
29.   }
30.
31.   private loadDataMap(): void {
32.     this.data = [];
33.     let self = this;
34.     this.http.request('http://localhost:5201/employee/getemployee')
35.       .flatMap((result) => result.json())
36.       .map((emp) => {
37.         emp["Name"] = "Mr. " + emp["Name"];
38.         return emp;
39.       })
40.       .subscribe((result) => {
41.         self.data.push(result);
42.       });
43.   }
44.
45.   private loadDataFilter(): void {
46.     this.data = [];
47.     let self = this;
48.     this.http.request('http://localhost:5201/employee/getemployee')
49.       .flatMap((result) => result.json())
50.       .filter((emp) => emp["Salary"] <= 20000)
51.       .subscribe((result) => {
```

ANGULARJS2

```
52.         self.data.push(result);
53.     });
54. }
55. }
```

app.component.homepage.html

```
1. <div>
2.     <h3>HTTP Module Sample - Get Data</h3>
3.     <div class="panel panel-default">
4.         <div class="panel-body">
5.             <table class="table table-striped table-bordered">
6.                 <thead>
7.                     <tr>
8.                         <th>Serial No</th>
9.                         <th>Alias</th>
10.                        <th>Employee Name</th>
11.                        <th>Date of Birth</th>
12.                        <th>Join Date</th>
13.                        <th>Department</th>
14.                        <th>Designation</th>
15.                        <th>Salary</th>
16.                    </tr>
17.                </thead>
18.                <tbody>
19.                    <tr *ngFor="let item of data">
20.                        <td>{{item.Id}}</td>
21.                        <td>{{item.Code}}</td>
22.                        <td>{{item.Name}}</td>
23.                        <td>{{item.DOB | date : 'shortDate'}}</td>
24.                        <td>{{item.DOJ | date : 'mediumDate'}}</td>
25.                        <td>{{item.Department}}</td>
26.                        <td>{{item.Designation}}</td>
27.                        <td>{{item.Salary | currency: 'INR': true}}</td>
28.                    </tr>
29.                </tbody>
30.            </table>
31.            <p>
32.                <button class="btn btn-primary" (click)="loadData()">
33.                    Load All Data
34.                </button>
35.                <button class="btn btn-primary" (click)="loadDataMap()">
36.                    Load Data With Map
37.                </button>
38.                <button class="btn btn-primary" (click)="loadDataFilter()">
39.                    Load Data With Filter
40.                </button>
41.            </p>
42.        </div>
43.    </div>
44. </div>
```

app.module.ts

```
1. import { NgModule, NO_ERRORS_SCHEMA } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { ReactiveFormsModule } from '@angular/forms';
4. import { HttpClientModule } from '@angular/http';
5.
6. import { HomeComponent } from './src/app.component.homepage';
7.
8. @NgModule({
9.     imports: [BrowserModule, ReactiveFormsModule, HttpClientModule],
10.    declarations: [HomeComponent],
11.    bootstrap: [HomeComponent]
```

ANGULARJS2

```
12. })
13. export class AppModule { }
```

main.ts

```
1. import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2.
3. import { AppModule } from './app.module';
4.
5. const platform = platformBrowserDynamic();
6. platform.bootstrapModule(AppModule);
```

index.html

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <title>Angular2 - HTTP Module (GET) </title>
5.   <meta charset="UTF-8">
6.   <meta name="viewport" content="width=device-width, initial-scale=1">
7.   <link href="../../../resources/style/bootstrap.css" rel="stylesheet" />
8.   <link href="../../../resources/style/style1.css" rel="stylesheet" />
9.   <!-- Polyfills for older browsers -->
10.  <script src="../../../resources/js/jquery-2.1.1.js"></script>
11.  <script src="../../../resources/js/bootstrap.js"></script>
12.
13.  <script src="../../../node_modules/core-js/client/shim.min.js"></script>
14.  <script src="../../../node_modules/zone.js/dist/zone.js"></script>
15.  <script src="../../../node_modules/reflect-metadata/Reflect.js"></script>
16.  <script src="../../../node_modules/systemjs/dist/system.src.js"></script>
17.  <script src="../../../systemjs.config.js"></script>
18.  <script>
19.    System.import('app').catch(function (err) { console.error(err); });
20.  </script>
21.  <!-- Set the base href, demo only! In your app: <base href="/"> -->
22.  <script>document.write('<base href="' + document.location + '"' />');</script>
23. </head>
24. <body>
25.   <home-page>Loading</home-page>
26. </body>
27. </html>
```

Employee.cs (model class)

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5.
6. namespace SampleAPI.Models.Sample
7. {
8.     public class Employee
9.     {
10.         public int Id { get; set; }
11.         public string Code { get; set; }
12.         public string Name { get; set; }
13.         public DateTime DOB { get; set; }
14.         public DateTime DOJ { get; set; }
15.         public string Department { get; set; }
16.         public string Designation { get; set; }
17.         public double Salary { get; set; }
18.     }
19. }
```

```
1. using SampleAPI.Models.Sample;  
2. using System;  
3. using System.Collections.Generic;  
4. using System.Linq;  
5. using System.Net;  
6. using System.Net.Http;  
7. using System.Web.Http;  
8. using System.Web.Http.Description;  
9.  
10. namespace SampleAPI.Controllers.Sample  
11. {  
12.     public class EmployeeController : ApiController  
13.     {  
14.         static List<Employee> IstData = new List<Employee>();  
15.  
16.         public EmployeeController()  
17.         {  
18.         }  
19.  
20.         [ResponseType(typeof(Employee))]  
21.         [HttpGet]  
22.         [Route("Employee/GetEmployee")]  
23.         public IHttpActionResult GetEmployee()  
24.         {  
25.             if (IstData.Count == 0)  
26.                 this.FetchEmployee();  
27.             return Ok(IstData);  
28.         }  
29.  
30.         [ResponseType(typeof(Employee))]  
31.         [HttpPost]  
32.         [Route("Employee/AddEmployee")]  
33.         public IHttpActionResult AddEmployee(Employee objEmp)  
34.         {  
35.             return Ok(this.SaveEmployee(objEmp));  
36.         }  
37.  
38.         private List<Employee> FetchEmployee()  
39.         {  
40.             Employee objEmp = new Employee() { };  
41.             objEmp.Id = 1;  
42.             objEmp.Code = "A001";  
43.             objEmp.Name = "RABIN";  
44.             objEmp.DOB = Convert.ToDateTime("10-06-1980");  
45.             objEmp.DOJ = Convert.ToDateTime("01-09-2006");  
46.             objEmp.Department = "ACCOUNTS";  
47.             objEmp.Designation = "CLERK";  
48.             objEmp.Salary = 15000.00;  
49.             IstData.Add(objEmp);  
50.  
51.             objEmp = new Employee() { };  
52.             objEmp.Id = 2;  
53.             objEmp.Code = "A002";  
54.             objEmp.Name = "SUJIT";  
55.             objEmp.DOB = Convert.ToDateTime("12-22-1986");  
56.             objEmp.DOJ = Convert.ToDateTime("04-15-2010");  
57.             objEmp.Department = "SALES";  
58.             objEmp.Designation = "MANAGER";  
59.             objEmp.Salary = 35000.00;  
60.             IstData.Add(objEmp);  
61.  
62.             objEmp = new Employee() { };  
63.             objEmp.Id = 3;  
64.             objEmp.Code = "A003";
```

ANGULARJS2

```
65.         obj Emp. Name = "KAMALESH";
66.         obj Emp. DOB = Convert.ToDateTime("03-22-1982");
67.         obj Emp. DOJ = Convert.ToDateTime("07-15-2006");
68.         obj Emp. Department = "ACCOUNTS";
69.         obj Emp. Designation = "CLERK";
70.         obj Emp. Salary = 16000.00;
71.         IstData.Add(obj Emp);
72.
73.         return IstData;
74.     }
75.
76.     private bool SaveEmployee(Employee obj Emp)
77.     {
78.         obj Emp. Id = (IstData.OrderByDescending(s => s.Id).FirstOrDefault()).Id + 1;
79.         IstData.Add(obj Emp);
80.         return true;
81.     }
82. }
83. }
```