Angular JS is an open source framework built over JavaScript. It was built by the developers at Google. This framework was used to overcome obstacles encountered while working with Single Page applications. Also, testing was considered as a key aspect while building the framework. It was ensured that the framework could be easily tested. The initial release of the framework was in October 2010.

Features of Angular 2

Following are the key features of Angular 2 −

* **Components** − The earlier version of Angular had a focus of Controllers but now has changed the focus to having components over controllers. Components help to build the applications into many modules. This helps in better maintaining the application over a period of time.
* **TypeScript** − The newer version of Angular is based on TypeScript. This is a superset of JavaScript and is maintained by Microsoft.
* **Services** − Services are a set of code that can be shared by different components of an application. So for example if you had a data component that picked data from a database, you could have it as a shared service that could be used across multiple applications.­­­­­­­­

In addition, Angular 2 has better event-handling capabilities, powerful templates, and better support for mobile devices.

Components of Angular 2

Angular 2 has the following components −

* **Modules** − This is used to break up the application into logical pieces of code. Each piece of code or module is designed to perform a single task.
* **Component** − This can be used to bring the modules together.
* **Templates** − This is used to define the views of an Angular JS application.
* **Metadata** − This can be used to add more data to an Angular JS class.
* **Service** − This is used to create components which can be shared across the entire application.

Modules are used in Angular JS to put logical boundaries in your application. Hence, instead of coding everything into one application, you can instead build everything into separate modules to separate the functionality of your application. Let’s inspect the code which gets added to the demo application.

In Visual Studio code, go to the app.module.ts folder in your app folder. This is known as the root module class.

The following code will be present in the **app.module.ts** file.

import { NgModule } from '@angular/core';

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

import { AppComponent } from './app.component';

@NgModule ({

imports: [ BrowserModule ],

declarations: [ AppComponent ],

bootstrap: [ AppComponent ]

})

export class AppModule { }

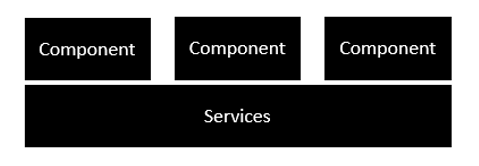
Let’s go through each line of the code in detail.

* The import statement is used to import functionality from the existing modules. Thus, the first 3 statements are used to import the NgModule, BrowserModule and AppComponent modules into this module.
* The NgModule decorator is used to later on define the imports, declarations, and bootstrapping options.
* The BrowserModule is required by default for any web based angular application.
* The bootstrap option tells Angular which Component to bootstrap in the application.

A module is made up of the following parts −

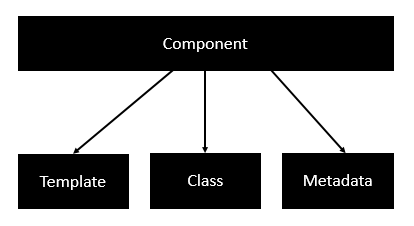
* **Bootstrap array** − This is used to tell Angular JS which components need to be loaded so that its functionality can be accessed in the application. Once you include the component in the bootstrap array, you need to declare them so that they can be used across other components in the Angular JS application.
* **Export array** − This is used to export components, directives, and pipes which can then be used in other modules.
* **Import array** − Just like the export array, the import array can be used to import the functionality from other Angular JS modules.

The following screenshot shows the anatomy of an Angular 2 application. Each application consists of Components. Each component is a logical boundary of functionality for the application. You need to have layered services, which are used to share the functionality across components.



Following is the anatomy of a Component. A component consists of −

* **Class** − This is like a C++ or Java class which consists of properties and methods.
* **Metadata** − This is used to decorate the class and extend the functionality of the class.
* **Template** − This is used to define the HTML view which is displayed in the application.



Following is an example of a component.

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

@Component ({

selector: 'my-app',

templateUrl: 'app/app.component.html'

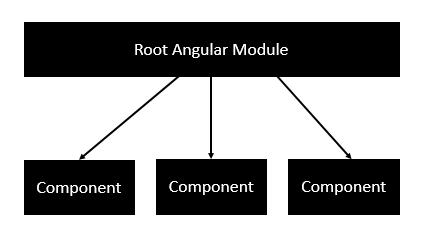
})

export class AppComponent {

appTitle: string = 'Welcome';

}

Each application is made up of modules. Each Angular 2 application needs to have one Angular Root Module. Each Angular Root module can then have multiple components to separate the functionality.



Following is an example of a root module.

import { NgModule } from '@angular/core';

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

import { AppComponent } from './app.component';

@NgModule ({

imports: [ BrowserModule ],

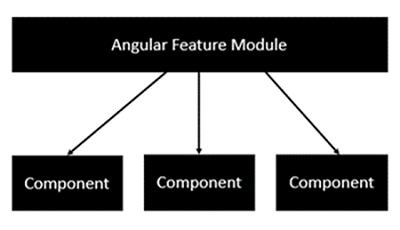
declarations: [ AppComponent ],

bootstrap: [ AppComponent ]

})

export class AppModule { }

Each application is made up of feature modules where each module has a separate feature of the application. Each Angular feature module can then have multiple components to separate the functionality.



Components are a logical piece of code for Angular JS application. A Component consists of the following −

* **Template** − This is used to render the view for the application. This contains the HTML that needs to be rendered in the application. This part also includes the binding and directives.
* **Class** − This is like a class defined in any language such as C. This contains properties and methods. This has the code which is used to support the view. It is defined in TypeScript.
* **Metadata** − This has the extra data defined for the Angular class. It is defined with a decorator.

| **Observables** | **Promise** |
| --- | --- |
| Observables handle multiple values over time | Promises are only called once and will return a single value |
| Observables are cancellable | Promises are not cancellable |

Some of these operators are:

* Map
* Filter
* Take
* Skip
* Debounce

With the map operator, we call the .json method on the response because the actual response is not a collection of data but a JSON string.

It is always advisable to handle errors so we can use the catch operator to return another subscribable observable but this time a failed one.

**// subscribe to output from this observable and bind**

**// the output to the component when received**

**.subscribe( res => this.tasks = res);**

1- You assigned the observable itself to this.result. When you actually wanted to assign the list of friends to this.result. The correct way to do it is:

* you subscribe to the observable. .subscribe is the function that actually executes the observable. It takes three callback parameters as follow:

.subscribe(success, failure, complete);

for example:

.subscribe(

function(response) { console.log("Success Response" + response)},

function(error) { console.log("Error happened" + error)},

function() { console.log("the subscription is completed")}

);

|  |
| --- |
| //emit (1,2,3,4,5) |
|  | const source = Rx.Observable.from([1,2,3,4,5]); |
|  | //add 10 to each value |
|  | const example = source.map(val => val + 10); |
|  | //output: 11,12,13,14,15 |
|  | const subscribe = example.subscribe(val => console.log(val)); |
|  |  |
|  | //emit ({name: 'Joe', age: 30}, {name: 'Frank', age: 20},{name: 'Ryan', age: 50}) |
|  | const sourceTwo = Rx.Observable.from([{name: 'Joe', age: 30}, {name: 'Frank', age: 20},{name: 'Ryan', age: 50}]); |
|  | //grab each persons name |
|  | const exampleTwo = sourceTwo.map(person => person.name); |
|  | //output: "Joe","Frank","Ryan" |
|  | const subscribe = exampleTwo.subscribe(val => console.log(val)); |

#### *Map*

Apply certain logic to each element passed.

A very good example would be to convert data coming from backend to Json format.

* Filter

|  |
| --- |
| //emit (1,2,3,4,5) |
|  | const source = Rx.Observable.from([1,2,3,4,5]); |
|  | //add 10 to each value |
|  | const example = source.map(val => val + 10); |
|  | //output: 11,12,13,14,15 |
|  | const subscribe = example.subscribe(val => console.log(val)); |
|  |  |
|  | //emit ({name: 'Joe', age: 30}, {name: 'Frank', age: 20},{name: 'Ryan', age: 50}) |
|  | const sourceTwo = Rx.Observable.from([{name: 'Joe', age: 30}, {name: 'Frank', age: 20},{name: 'Ryan', age: 50}]); |
|  | //grab each persons name |
|  | const exampleTwo = sourceTwo.map(person => person.name); |
|  | //output: "Joe","Frank","Ryan" |
|  | const subscribe = exampleTwo.subscribe(val => console.log(val)); |