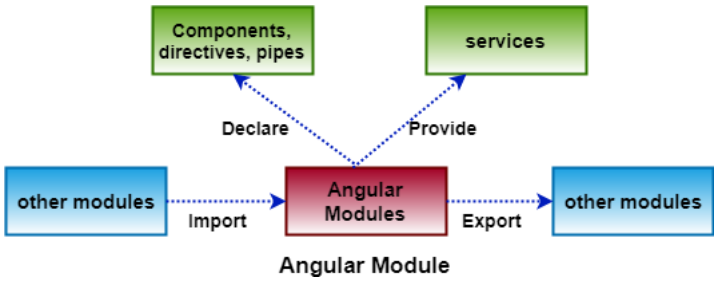
Angular 8 Module

August 22, 2019

Angular 8 Module

It is a collection of **services**, **directives**, **controllers, filters**, and **configuration information**. **angular.module** is used to configure the $injector. The module is a container of the different parts of an application. Controllers always belong to a module. In case of developing a website, the header, footer, left, center, and the right section become part of a module.

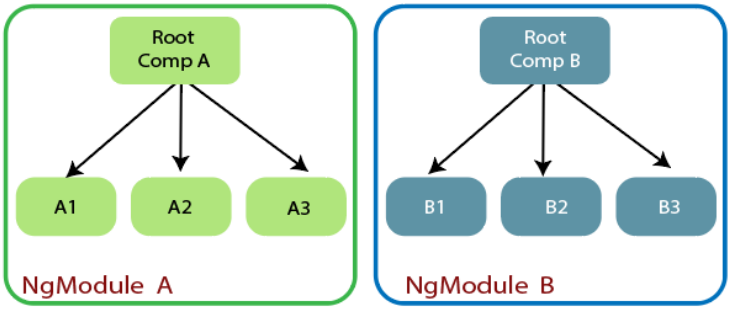


Angular Module

In Angular, a module is a technique to group the components, directives, pipes, and services which are related, in such a way that is combined with other modules to create an application.

Another way to understand Angular modules is classes. In class, we can define public or private methods. The general purposes are the API that other parts of our code can use to interact with it while the individual techniques are implemented details which are hidden.

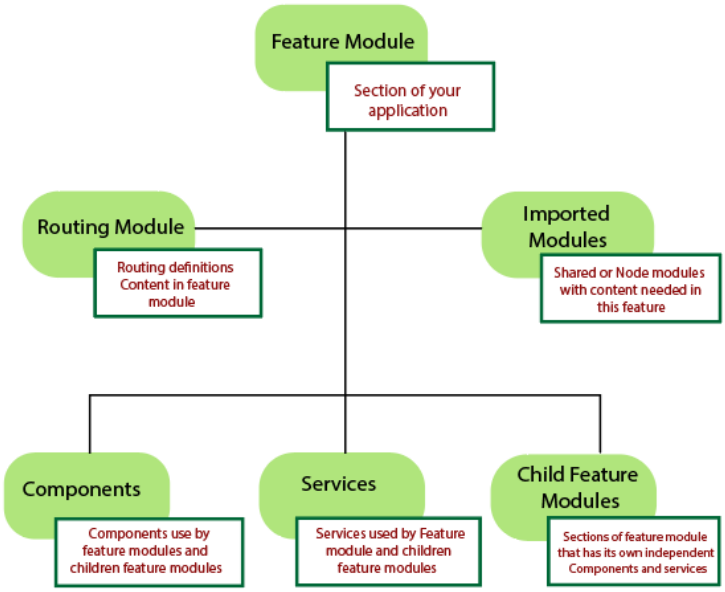
In the same way, a module export or hide **components, directives, pipes**, and **services**. The exported elements are used by other modules, while that are not export in the module and cannot be accessed by other modules of our application.



Creating a Module

A module is created by using the Angular function **angular.module**

```
<div ng-app=""myApp"">…</div>
 <script>
 var app= angular.module(“myApp”,[]);
 </script>
```



Use of Module

We have to use the decorator **NgModule** to define modules.

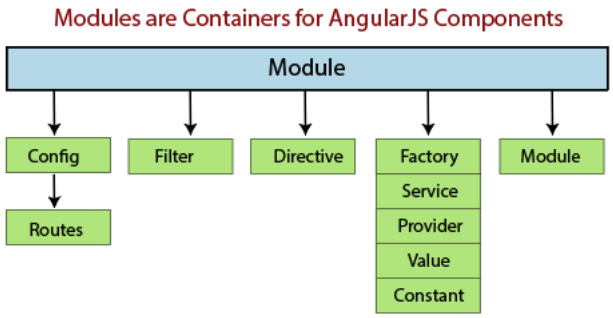
```
import {NgModule} from '@angular/core';
 @NgModule({
 Import:[…],
 declarations:[…],
 bootstrap:[…]
 }]
 export class AppModule{}
```

In the above example, we turned the class **AppModule** into an Angular module by using the NgModule decorator. The **NgModule** decorator needs at least three properties:

- **Imports**
- **declarations**
- **bootstrap**

The property **imports** expect an array of modules. Where we define the piece of the puzzle.

The property **declaration** expects an array of components, directives, and pipes that are the part of the module.



The **bootstrap** property is when we define the root component of the module. However, this property is also an array, 99% of the time, and we are going to explain only one element.

- **Note:** ***There are particular circumstances where more than one component is required to bootstrap a module, but we are not covering those edge cases.***

Example:

app/app.component.ts file

```
import { Component }from ‘@angular/core';
 @Component ({
 selector:'app-root',
 template:
 '<h1>my angular app</h1>'
 })
```

app/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 {}
```

app.component.ts is a “**hello world**” component, nothing is interesting there. In the other hand, the structure that we’ve seen before for defining a module but in the case, we are defining the modules and components that we are about to using.

The first thing is that we notice our module is importing the **BrowserModule** like an explicit dependency. The **BrowserModule** is a built-in module which exports basic pipes, directives, and services. Apart from previous versions of Angular, we have to precisely import those dependencies to use directives like ***ngFor** or ***ngIf** in our templates.

The root component of our module is the **AppComponent** where we have to list it into the **bootstrap** array. Because in the declaration property we have supposed to define all elements or pipes that make our application reactive, we have to set the **AppComponent** again.

Before moving on, there’s a necessary clarification to make.

There are two types of modules:

- **Root modules**
- **Features modules**

We have one root component and many possible secondary components in any application, and only have one root module and zero or more feature modules. To understand bootstrap, we need to know the root module.

Bootstrapping an application in Angular 8

Bootstrapping refers to the starting of a self-dependent process that is supposed to process without external input. To bootstrap our module based application, we need to inform Angular, which is our root module to perform the compilation in the browser. This compilation in the browser is known as “**Just in Time**” (**JIT**) compilation process.

main.ts file

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';
platformBrowserDynamic().bootstrapModule(AppModule);
```

It is possible to perform the compilation like a build step to our workflow. This method is called “**Ahead of Time**” (AOT) compilation and will require a slightly different bootstrap process.

NgModule metadata

NgModule is defined through a class decorated with **the help of @NgModule()**. The @NgModule() decorator is a function which extracts a single metadata object, whose properties describe the module. The most important features are given below.

- `<strong>Declarations</strong>` : The **components**, *directives*, and *pipes* belong to the **NgModule**.
- `<strong>Exports</strong>` : The subset of declarations should be visible and usable into the *component templates* of other **NgModules**.
- `<strong>Imports</strong>` : Component templates need the other modules whose exported classes are declared in *the* NgModule.

- `<strong>Providers</strong>` : Creators of services that NgModule contributes to the global collection of services; they become accessible in all parts of the app. (we can also specify providers at the component, which is selected).
- `<strong>Bootstrap</strong>` : The main application view, also called the *root component*, which hosts all the app views. Only the *root* ***NgModule***will set the `bootstrap` property.

`<strong>src /app/app</strong>` **.module.ts file**

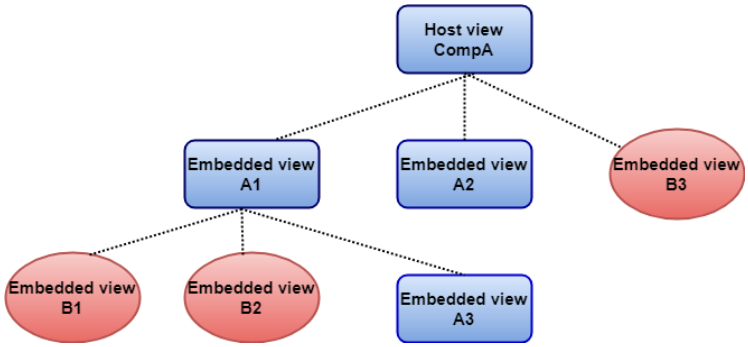
```
import { NgModule } from '@angular/core';
import { BrowserModule }from '@angular/platform-browser';
@NgModule ({
imports:[BrowserModule ],
providers:[Logger ],
declarations:[ AppComponent ],
exports:[ AppComponent ],
 bootstrap:[ AppComponent ]
})
 export class AppModule{ }
```

App Component is included in the exports list for illustration; it is not necessary in the example. A root NgModule has no any reason to export anything from file because the other modules does'nt import the root NgModule.

NgModules and Components

NgModules provide a *compilation context* for its components. A root NgModule has a root component which is created during the bootstrap working, but NgModule can include only the additional components. Which load through the router or created by the template.

A component and its template simultaneously define a *view*. A part contains a *view hierarchy*, which allows us to represent the complex aimless area of the screen that can be created, modified, and destroyed as a unit. A view hierarchy can define in components which belong to different NgModules.



When we create a component, it is connected directly with a single view, called the *host view*. The host view is the root of a view hierarchy, which contains *embedded views*, which are, in turn, the host view of other components. Those components can in the same **NgModule** and be imported from other **NgModules**. Views in the tree can nested to any depth.

- **Note: The hierarchical structure of views is a key factor in the way Angular detects and responds to change in the DOM and app data.**

NgModules and JavaScript modules

The **Ngmodule** system is different from the JavaScript module system for managing collections of JavaScript objects. These are complementary module systems that we can use together to write our app.

In JavaScript, every file is a module and object defined in the file belong to that module. The module. The module declares some objects by marking them with the export keyword. Other JavaScript module use import statement to access public objects from other modules.

```
import { NgModule }   from '@angular/core';
import { AppComponent } from './app.component';
export class AppModule { }
```

Difference between Module and Component

Module	Component
A module is a collection of components, services, pipes, directives, and so on.	A component in Angular is a building block of the application with an associated template view.