**Why use multiple NgModules?**

Multiple [NgModules](https://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html" \t "_blank) provides some potential benefits.

Actually, the modules help you to organize an application into associative blocks of functionality.

First one is organizing an application code. If you are putting around 99 resource files in the default app module and see the happing.

And the second one is - It opens the possibility of lazy loading via the router.

What Types of NgModules?

There are four types of NgModules –

1.          Features Module

2.          Routing Module

3.          Service Module

4.          Widget Module

5.          Shared Module

Features Module – The feature modules are [NgModules](https://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html" \t "_blank) for the purpose of organizing an application code.

Routing Module – The Routing is used to manage routes and also enables navigation from one view to another view as users perform application tasks.

Service Module – The modules that only contain services and providers. It provides utility services such as data access and messaging. The root AppModule is the only module that should import service modules. The [HttpClientModule](https://www.code-sample.com/2018/04/ngmodule-metadata-properties.html" \t "_blank) is a good example of a service.

Widget Module - The third party UI component libraries are widget modules.

Shared Module – The shared module allows you to organize your application code. You can put your commonly used components, directives, and pipes into the one module and use whenever required to this module.

**What are the @NgModule Metadata Properties?**

The @NgModule decorator identifies AppModule as a NgModule class.

The [@NgModule](http://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html) takes a metadata object that tells Angular how to compile and launch the application.

The NgModule’s important metadata properties are as follows –

1.          providers

2.          declarations

3.          imports

4.          exports

5.          entryComponents

6.          bootstrap

7.          schemas

8.          id

The @NgModule class with the decorator and metadata properties -

@NgModule({

  providers?: Provider[]

  declarations?: Array<Type<any> | any[]>

  imports?: Array<Type<any> | ModuleWithProviders | any[]>

  exports?: Array<Type<any> | any[]>

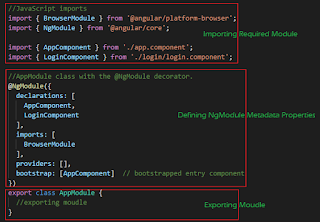
  entryComponents?: Array<Type<any> | any[]>

  bootstrap?: Array<Type<any> | any[]>

  schemas?: Array<SchemaMetadata | any[]>

  id?: string

})

[](https://3.bp.blogspot.com/-Tt28J-c22Ns/WuL6KDvmYMI/AAAAAAAASZk/vInbrXPB3k4a2gwKkIt0hQOuplunipOVQCLcBGAs/s1600/Angular-5-6-Modules-NgModule-decorator.png)

Let understand in detail about NgModule metadata is as follows-

Providers – A list of dependency injection (DI) providers and it defines the set of injectable objects that are available in the injector of this module.

Declarations - A list of declarable classes, components, directives, and pipes that belong to this module. The compiler throws an error if you try to declare the same class in multiple modules.

Imports - A list of modules and it used to import the supporting modules like FormsModule, RouterModule, CommonModule, or any other custom made feature module.

Exports - A list of declarable components, directives, pipes, and modules that an importing module can be used within a template of any component.

EntryComponents - A list of components that should be compiled when this module is defined. By default, an Angular app always has at least one entry component, the root component, AppComponent.

A bootstrapped component is an entry component that Angular loads into DOM during the application launch and other root components loaded dynamically into entry components.

Bootstrap – A list of components that are automatically bootstrapped and the listed components will be added automatically to entryComponents.

Schemas - Defines a schema that will allow any non-Angular elements and properties.

Id – The Id used to identify the modules in getModuleFactory. If left undefined, the NgModule will not be registered with getModuleFactory.

What Are the Purpose of @NgModule?

The NgModule is used to simplify the ways you define and manage the dependencies in your applications and also you can consolidate different components and services into cohesive blocks of functionality.

The @NgModule metadata divided into three categories as follows.

1.          Static

2.          Runtime

3.          Composability/Grouping

Static – It is compiler configuration and configured via the declarations array.

Runtime - It is injector configuration and configured via the provider’s array.

Composability/Grouping – Introducing [NgModules](http://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html" \t "_blank) together and configured via the imports and exports arrays.

The following is an example of specifying a NgModule metadata -

@NgModule({

  // Static, This is the compiler configuration

  declarations: [], //declarations is used for configure the selectors.

  entryComponents: [], //entryComponents is used to generate the host factory.

  //Runtime or injector configuration

  providers: [], // providers is used for runtime injector configuration.

  //Composability and Grouping

  imports: [], // imports used for composing NgModules together.

  exports: [] //A list of declarations components, directives, and pipes classes that an importing module can use.

})