# Fundamentals

## NgModules: Singleton Services

A singleton service is a service for which only one instance exists in an app.

For a sample app using the app-wide singleton service that this page describes, see the [live example](https://angular.io/generated/live-examples/ngmodules/stackblitz.html) / [download example](https://angular.io/generated/zips/ngmodules/ngmodules.zip) showcasing all the documented features of NgModules.

## Providing a singleton service

There are two ways to make a service a singleton in Angular:

* Set the providedIn property of the @[Injectable](https://angular.io/api/core/Injectable)() to "root".
* Include the service in the AppModule or in a module that is only imported by the AppModule

### Using providedIn

Beginning with Angular 6.0, the preferred way to create a singleton service is to set providedIn to root on the service's @[Injectable](https://angular.io/api/core/Injectable)() decorator. This tells Angular to provide the service in the application root.

src/app/user.service.ts

content\_copyimport { [Injectable](https://angular.io/api/core/Injectable) } from '@angular/core';

@[Injectable](https://angular.io/api/core/Injectable)({

providedIn: 'root',

})

export class UserService {

}

For more detailed information on services, see the [Services](https://angular.io/tutorial/toh-pt4) chapter of the [Tour of Heroes tutorial](https://angular.io/tutorial).

### NgModule providers array

In apps built with Angular versions prior to 6.0, services are registered NgModule providers arrays as follows:

content\_copy@[NgModule](https://angular.io/api/core/NgModule)({

...

providers: [UserService],

...

})

If this NgModule were the root AppModule, the UserService would be a singleton and available throughout the app. Though you may see it coded this way, using the providedIn property of the @[Injectable](https://angular.io/api/core/Injectable)() decorator on the service itself is preferable as of Angular 6.0 as it makes your services tree-shakable.

## The forRoot() pattern

Generally, you'll only need providedIn for providing services and forRoot()/forChild() for routing. However, understanding how forRoot() works to make sure a service is a singleton will inform your development at a deeper level.

If a module defines both providers and declarations (components, directives, pipes), then loading the module in multiple feature modules would duplicate the registration of the service. This could result in multiple service instances and the service would no longer behave as a singleton.

There are multiple ways to prevent this:

* Use the [providedIn syntax](https://angular.io/guide/singleton-services#providedIn) instead of registering the service in the module.
* Separate your services into their own module.
* Define forRoot() and forChild() methods in the module.

**Note:** There are two example apps where you can see this scenario; the more advanced [NgModules live example](https://angular.io/generated/live-examples/ngmodules/stackblitz.html), which contains forRoot() and forChild() in the routing modules and the GreetingModule, and the simpler [Lazy Loading live example](https://angular.io/generated/live-examples/lazy-loading-ngmodules/stackblitz.html). For an introductory explanation see the [Lazy Loading Feature Modules](https://angular.io/guide/lazy-loading-ngmodules) guide.

Use forRoot() to separate providers from a module so you can import that module into the root module with providers and child modules without providers.

1. Create a static method forRoot() on the module.
2. Place the providers into the forRoot() method.

src/app/greeting/greeting.module.ts

content\_copy[static](https://angular.io/api/upgrade/static) forRoot(config: UserServiceConfig): [ModuleWithProviders](https://angular.io/api/core/ModuleWithProviders)<GreetingModule> {

return {

ngModule: GreetingModule,

providers: [

{provide: UserServiceConfig, useValue: config }

]

};

}

### forRoot() and the [Router](https://angular.io/api/router/Router)

[RouterModule](https://angular.io/api/router/RouterModule) provides the [Router](https://angular.io/api/router/Router) service, as well as router directives, such as [RouterOutlet](https://angular.io/api/router/RouterOutlet) and [routerLink](https://angular.io/api/router/RouterLink). The root application module imports [RouterModule](https://angular.io/api/router/RouterModule) so that the application has a [Router](https://angular.io/api/router/Router) and the root application components can access the router directives. Any feature modules must also import [RouterModule](https://angular.io/api/router/RouterModule) so that their components can place router directives into their templates.

If the [RouterModule](https://angular.io/api/router/RouterModule) didn’t have forRoot() then each feature module would instantiate a new [Router](https://angular.io/api/router/Router) instance, which would break the application as there can only be one [Router](https://angular.io/api/router/Router). By using the forRoot() method, the root application module imports RouterModule.forRoot(...) and gets a [Router](https://angular.io/api/router/Router), and all feature modules import RouterModule.forChild(...) which does not instantiate another [Router](https://angular.io/api/router/Router).

**Note:** If you have a module which has both providers and declarations, you can use this technique to separate them out and you may see this pattern in legacy apps. However, since Angular 6.0, the best practice for providing services is with the @[Injectable](https://angular.io/api/core/Injectable)() providedIn property.

### How forRoot() works

forRoot() takes a service configuration object and returns a [ModuleWithProviders](https://angular.io/api/core/ModuleWithProviders), which is a simple object with the following properties:

* ngModule: in this example, the GreetingModule class
* providers: the configured providers

In the [live example](https://angular.io/generated/live-examples/ngmodules/stackblitz.html) / [download example](https://angular.io/generated/zips/ngmodules/ngmodules.zip) the root AppModule imports the GreetingModule and adds the providers to the AppModule providers. Specifically, Angular accumulates all imported providers before appending the items listed in @[NgModule.providers](https://angular.io/api/core/NgModule#providers). This sequence ensures that whatever you add explicitly to the AppModule providers takes precedence over the providers of imported modules.

The sample app imports GreetingModule and uses its forRoot() method one time, in AppModule. Registering it once like this prevents multiple instances.

You can also add a forRoot() method in the GreetingModule that configures the greeting UserService.

In the following example, the optional, injected UserServiceConfig extends the greeting UserService. If a UserServiceConfig exists, the UserService sets the user name from that config.

src/app/greeting/user.service.ts (constructor)

content\_copyconstructor(@[Optional](https://angular.io/api/core/Optional)() config?: UserServiceConfig) {

if (config) { this.\_userName = config.userName; }

}

Here's forRoot() that takes a UserServiceConfig object:

src/app/greeting/greeting.module.ts (forRoot)

content\_copy[static](https://angular.io/api/upgrade/static) forRoot(config: UserServiceConfig): [ModuleWithProviders](https://angular.io/api/core/ModuleWithProviders)<GreetingModule> {

return {

ngModule: GreetingModule,

providers: [

{provide: UserServiceConfig, useValue: config }

]

};

}

Lastly, call it within the imports list of the AppModule. In the following snippet, other parts of the file are left out. For the complete file, see the [live example](https://angular.io/generated/live-examples/ngmodules/stackblitz.html) / [download example](https://angular.io/generated/zips/ngmodules/ngmodules.zip), or continue to the next section of this document.

src/app/app.module.ts (imports)

content\_copyimport { GreetingModule } from './greeting/greeting.module';

@[NgModule](https://angular.io/api/core/NgModule)({

imports: [

GreetingModule.forRoot({userName: 'Miss Marple'}),

],

})

The app displays "Miss Marple" as the user instead of the default "Sherlock Holmes".

Remember to import GreetingModule as a Javascript import at the top of the file and don't add it to more than one @[NgModule](https://angular.io/api/core/NgModule) imports list.

## Prevent reimport of the GreetingModule

Only the root AppModule should import the GreetingModule. If a lazy-loaded module imports it too, the app can generate [multiple instances](https://angular.io/guide/ngmodule-faq#q-why-bad) of a service.

To guard against a lazy loaded module re-importing GreetingModule, add the following GreetingModule constructor.

src/app/greeting/greeting.module.ts

content\_copyconstructor(@[Optional](https://angular.io/api/core/Optional)() @[SkipSelf](https://angular.io/api/core/SkipSelf)() parentModule?: GreetingModule) {

if (parentModule) {

throw new Error(

'GreetingModule is already loaded. Import it in the AppModule only');

}

}

The constructor tells Angular to inject the GreetingModule into itself. The injection would be circular if Angular looked for GreetingModule in the current injector, but the @[SkipSelf](https://angular.io/api/core/SkipSelf)() decorator means "look for GreetingModule in an ancestor injector, above me in the injector hierarchy."

By default, the injector throws an error when it can't find a requested provider. The @[Optional](https://angular.io/api/core/Optional)() decorator means not finding the service is OK. The injector returns null, the parentModule parameter is null, and the constructor concludes uneventfully.

It's a different story if you improperly import GreetingModule into a lazy loaded module such as CustomersModule.

Angular creates a lazy loaded module with its own injector, a child of the root injector. @[SkipSelf](https://angular.io/api/core/SkipSelf)() causes Angular to look for a GreetingModule in the parent injector, which this time is the root injector. Of course it finds the instance imported by the root AppModule. Now parentModule exists and the constructor throws the error.

Here are the two files in their entirety for reference:

app.module.ts

greeting.module.ts

content\_copyimport { [BrowserModule](https://angular.io/api/platform-browser/BrowserModule) } from '@angular/platform-browser';

import { [NgModule](https://angular.io/api/core/NgModule) } from '@angular/core';

/\* App Root \*/

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

/\* Feature Modules \*/

import { ContactModule } from './contact/contact.module';

import { GreetingModule } from './greeting/greeting.module';

/\* Routing Module \*/

import { AppRoutingModule } from './app-routing.module';

@[NgModule](https://angular.io/api/core/NgModule)({

imports: [

[BrowserModule](https://angular.io/api/platform-browser/BrowserModule),

ContactModule,

GreetingModule.forRoot({userName: 'Miss Marple'}),

AppRoutingModule

],

declarations: [

AppComponent

],

bootstrap: [AppComponent]

})

export class AppModule { }

## More on NgModules

You may also be interested in:

* [Sharing Modules](https://angular.io/guide/sharing-ngmodules), which elaborates on the concepts covered on this page.
* [Lazy Loading Modules](https://angular.io/guide/lazy-loading-ngmodules).
* [NgModule FAQ](https://angular.io/guide/ngmodule-faq).