NgModule API

At a high level, NgModules are a way to organize Angular apps and they accomplish this through the metadata in the `@NgModule` decorator. The metadata falls into three categories:

- **Static:** Compiler configuration which tells the compiler about directive selectors and where in templates the directives should be applied through selector matching. This is configured via the `declarations` array.
- **Runtime:** Injector configuration via the `providers` array.
- **Composability/Grouping:** Bringing NgModules together and making them available via the `imports` and `exports` arrays.

```
@NgModule({
  // Static, that is compiler configuration
  declarations: [], // Configure the selectors
  entryComponents: [], // Generate the host factory

  // Runtime, or injector configuration
  providers: [], // Runtime injector configuration

  // Composability / Grouping
  imports: [], // composing NgModules together
  exports: [] // making NgModules available to other parts of the app
})
```

`@NgModule` metadata

The following table summarizes the `@NgModule` metadata properties.

Property	Description
`declarations`	A list of declarable classes, (*components*, *directives*, and *pip* *belong to this module*. 1. When compiling a templat need to determine a set of selectors which should be triggering their correspond directives. 2. The template is compiled the context of an NgModul NgModule within which th template's component is d —which determines the se selectors using the followi ◦ All selectors of direc listed in `declaration ◦ All selectors of direc exported from impo NgModules.

	Components, directives, and pipe belong to *exactly* one module. Th compiler emits an error if you try declare the same class in more t module. Be careful not to re-decl class that is imported directly or indirectly from another module.
`providers`	A list of dependency-injection pro Angular registers these providers the NgModule's injector. If it is th NgModule used for bootstrappin is the root injector. These services become available injection into any component, dir pipe or service which is a child o injector. A lazy-loaded module has its own injector which is typically a child application root injector. Lazy-loaded services are scoped lazy module's injector. If a lazy-lo

	module also provides the `UserSe` any component created within th module's context (such as by rou navigation) gets the local instanc service, not the instance in the rc application injector. Components in external modules continue to receive the instance by their injectors. For more information on injector hierarchy and scoping, see Provi the DI Guide.
`imports`	A list of modules which should b into this module. Folded means i all the imported NgModule's expo properties were declared here. Specifically, it is as if the list of n whose exported components, dir or pipes are referenced by the component templates were decl this module.

	A component template can refere another component, directive, or when the reference is declared in module or if the imported modul exported it. For example, a comp can use the `NgIf` and `NgFor` dire only if the module has imported t Angular `CommonModule` (perhaps indirectly by importing `BrowserM` You can import many standard d from the `CommonModule` but som familiar directives belong to othe modules. For example, you can u `[(ngModel)]` only after importin Angular `FormsModule`.
`exports`	A list of declarations—*componen* *directive*, and *pipe* classes—that importing module can use. Exported declarations are the mo *public API*. A component in anotl

module can use *this* module's `UserComponent` if it imports this and this module exports `UserCon

Declarations are private by defau module does *not* export `UserCom then only the components within module can use `UserComponent`.

Importing a module does *not* automatically re-export the impo module's imports. Module 'B' can `ngIf` just because it imported m which imported `CommonModule`. M 'B' must import `CommonModule` its

A module can list another modul its `exports`, in which case all of module's public components, dire and pipes are exported.

Re-export makes module transiti explicit. If Module 'A' re-exports

	`CommonModule` and Module 'B' im Module 'A', Module 'B' componen use `ngIf` even though 'B' itself di import `CommonModule`.
`bootstrap`	A list of components that are automatically bootstrapped. Usually there's only one compone this list, the *root component* of tl application. Angular can launch with multiple bootstrap components, each witl location in the host web page. A bootstrap component is autom added to `entryComponents`.
`entryComponents`	A list of components that can be dynamically loaded into the view. By default, an Angular app alway least one entry component, the r

component, `AppComponent`. Its p
is to serve as a point of entry intc
app, that is, you bootstrap it to la
app.

Routed components are also *ent*
components because they need t
loaded dynamically. The router cı
them and drops them into the DC
a `<router-outlet>`.

While the bootstrapped and route
components are *entry componer*
don't have to add them to a modu
`entryComponents` list, as they ar
implicitly.

Angular automatically adds comp
in the module's `bootstrap` and rı
definitions into the `entryCompon`
list.

That leaves only components
bootstrapped using one of the im

techniques, such as `ViewComponentRef.createComp` as undiscoverable.

Dynamic component loading is n common in most apps beyond th If you need to dynamically load components, you must add these components to the `entryCompon` yourself.

For more information, see Entry Components.

More on NgModules

You may also be interested in the following:

- Feature Modules.
- Entry Components.
- Providers.
- Types of Feature Modules.