NgModules

NgModules configure the injector and the compiler and help organize related things together.

An NgModule is a class marked by the `@NgModule` decorator. `@NgModule` takes a metadata object that describes how to compile a component's template and how to create an injector at runtime. It identifies the module's own components, directives, and pipes, making some of them public, through the `exports` property, so that external components can use them. `@NgModule` can also add service providers to the application dependency injectors.

For an example app showcasing all the techniques that NgModules related pages cover, see the live example / download example. For explanations on the individual techniques, visit the relevant NgModule pages under the NgModules section.

Angular modularity

Modules are a great way to organize an application and extend it with capabilities from external libraries.

Angular libraries are NgModules, such as `FormsModule`, `HttpClientModule`, and `RouterModule`. Many third-party libraries are available as NgModules such as Material Design, Ionic, and AngularFire2.

NgModules consolidate components, directives, and pipes into cohesive blocks of functionality, each focused on a feature area, application business domain, workflow, or common collection of utilities.

Modules can also add services to the application. Such services might be internally developed, like something you'd develop yourself or come from outside sources, such as the Angular router and HTTP client.

Modules can be loaded eagerly when the application starts or lazy loaded asynchronously by the router.

NgModule metadata does the following:

- Declares which components, directives, and pipes belong to the module.
- Makes some of those components, directives, and pipes public so that other module's component templates can use them.
- Imports other modules with the components, directives, and pipes that components in the current module need.
- Provides services that the other application components can use.

Every Angular app has at least one module, the root module. You bootstrap that module to launch the application.

The root module is all you need in a simple application with a few components. As the app grows, you refactor the root module into feature modules that represent collections of related functionality. You then import these modules into the root module.

The basic NgModule

The Angular CLI generates the following basic `AppModule` when creating a new app.

src/app/app.module.ts (default AppModule)

```
// imports
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

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

// @NgModule decorator with its metadata
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

At the top are the import statements. The next section is where you configure the `@NgModule` by stating what components and directives belong to it (`declarations`) as well as which other modules it uses (`imports`). For more information on the structure of an `@NgModule`, be sure to read Bootstrapping.

More on NgModules

You may also be interested in the following:

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