# Fundamentals

## NgModules: Feature modules

Feature modules are NgModules for the purpose of organizing code.

For the final sample app with a feature module that this page describes, see the [live example](https://angular.io/generated/live-examples/feature-modules/stackblitz.html) / [download example](https://angular.io/generated/zips/feature-modules/feature-modules.zip).

As your app grows, you can organize code relevant for a specific feature. This helps apply clear boundaries for features. With feature modules, you can keep code related to a specific functionality or feature separate from other code. Delineating areas of your app helps with collaboration between developers and teams, separating directives, and managing the size of the root module.

## Feature modules vs. root modules

A feature module is an organizational best practice, as opposed to a concept of the core Angular API. A feature module delivers a cohesive set of functionality focused on a specific application need such as a user workflow, routing, or forms. While you can do everything within the root module, feature modules help you partition the app into focused areas. A feature module collaborates with the root module and with other modules through the services it provides and the components, directives, and pipes that it shares.

## How to make a feature module

Assuming you already have an app that you created with the [Angular CLI](https://angular.io/cli), create a feature module using the CLI by entering the following command in the root project directory. Replace CustomerDashboard with the name of your module. You can omit the "Module" suffix from the name because the CLI appends it:

content\_copyng generate module CustomerDashboard

This causes the CLI to create a folder called customer-dashboard with a file inside called customer-dashboard.module.ts with the following contents:

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

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

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

imports: [

[CommonModule](https://angular.io/api/common/CommonModule)

],

declarations: []

})

export class CustomerDashboardModule { }

The structure of an NgModule is the same whether it is a root module or a feature module. In the CLI generated feature module, there are two JavaScript import statements at the top of the file: the first imports [NgModule](https://angular.io/api/core/NgModule), which, like the root module, lets you use the @[NgModule](https://angular.io/api/core/NgModule) decorator; the second imports [CommonModule](https://angular.io/api/common/CommonModule), which contributes many common directives such as [ngIf](https://angular.io/api/common/NgIf) and [ngFor](https://angular.io/api/common/NgForOf). Feature modules import [CommonModule](https://angular.io/api/common/CommonModule) instead of [BrowserModule](https://angular.io/api/platform-browser/BrowserModule), which is only imported once in the root module. [CommonModule](https://angular.io/api/common/CommonModule) only contains information for common directives such as [ngIf](https://angular.io/api/common/NgIf) and [ngFor](https://angular.io/api/common/NgForOf) which are needed in most templates, whereas [BrowserModule](https://angular.io/api/platform-browser/BrowserModule) configures the Angular app for the browser which needs to be done only once.

The declarations array is available for you to add declarables, which are components, directives, and pipes that belong exclusively to this particular module. To add a component, enter the following command at the command line where customer-dashboard is the directory where the CLI generated the feature module and CustomerDashboard is the name of the component:

content\_copyng generate component customer-dashboard/CustomerDashboard

This generates a folder for the new component within the customer-dashboard folder and updates the feature module with the CustomerDashboardComponent info:

src/app/customer-dashboard/customer-dashboard.module.ts

content\_copy// import the new component

import { CustomerDashboardComponent } from './customer-dashboard/customer-dashboard.component';

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

imports: [

[CommonModule](https://angular.io/api/common/CommonModule)

],

declarations: [

CustomerDashboardComponent

],

})

The CustomerDashboardComponent is now in the JavaScript import list at the top and added to the declarations array, which lets Angular know to associate this new component with this feature module.

## Importing a feature module

To incorporate the feature module into your app, you have to let the root module, app.module.ts, know about it. Notice the CustomerDashboardModule export at the bottom of customer-dashboard.module.ts. This exposes it so that other modules can get to it. To import it into the AppModule, add it to the imports in app.module.ts and to the imports array:

src/app/app.module.ts

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

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

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

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

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

// import the feature module here so you can add it to the imports array below

import { CustomerDashboardModule } from './customer-dashboard/customer-dashboard.module';

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

declarations: [

AppComponent

],

imports: [

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

[FormsModule](https://angular.io/api/forms/FormsModule),

[HttpClientModule](https://angular.io/api/common/http/HttpClientModule),

CustomerDashboardModule // add the feature module here

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

Now the AppModule knows about the feature module. If you were to add any service providers to the feature module, AppModule would know about those too, as would any other feature modules. However, NgModules don’t expose their components.

## Rendering a feature module’s component template

When the CLI generated the CustomerDashboardComponent for the feature module, it included a template, customer-dashboard.component.html, with the following markup:

src/app/customer-dashboard/customer-dashboard/customer-dashboard.component.html

content\_copy<p>

customer-dashboard works!

</p>

To see this HTML in the AppComponent, you first have to export the CustomerDashboardComponent in the CustomerDashboardModule. In customer-dashboard.module.ts, just beneath the declarations array, add an exports array containing CustomerDashboardComponent:

src/app/customer-dashboard/customer-dashboard.module.ts

content\_copyexports: [

CustomerDashboardComponent

]

Next, in the AppComponent, app.component.html, add the tag <app-customer-dashboard>:

src/app/app.component.html

content\_copy<h1>

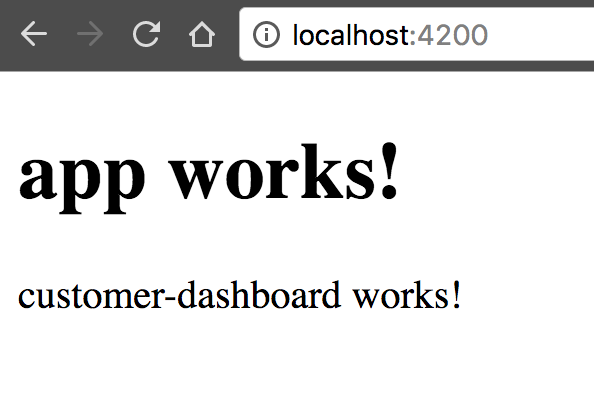
{{title}}

</h1>

<!-- add the selector from the CustomerDashboardComponent -->

<app-customer-dashboard></app-customer-dashboard>

Now, in addition to the title that renders by default, the CustomerDashboardComponent template renders too:



## More on NgModules

You may also be interested in the following:

* [Lazy Loading Modules with the Angular Router](https://angular.io/guide/lazy-loading-ngmodules).
* [Providers](https://angular.io/guide/providers).
* [Types of Feature Modules](https://angular.io/guide/module-types).