

Feature modules

Contents >

Feature modules vs. root modules

How to make a feature module

Importing a feature module

...

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](#) / [download example](#).

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](#), 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:


```
ng 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:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
  imports: [
```



```
    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, which, like the root module, lets you use the @NgModule decorator; the second imports CommonModule, which contributes many common directives such as ngIf and ngFor. Feature modules import CommonModule instead of BrowserModule, which is only imported once in the root module. CommonModule only contains information for common directives such as ngIf and ngFor which are needed in most templates, whereas 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:

```
ng 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

```
// import the new component
import { CustomerDashboardComponent } from './customer-dashboard/customer-
dashboard.component';
@NgModule({
  imports: [
    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

```
import { HttpClientModule } from '@angular/common/http';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { 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({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    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

```
<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

```
exports: [  
  CustomerDashboardComponent  
]
```



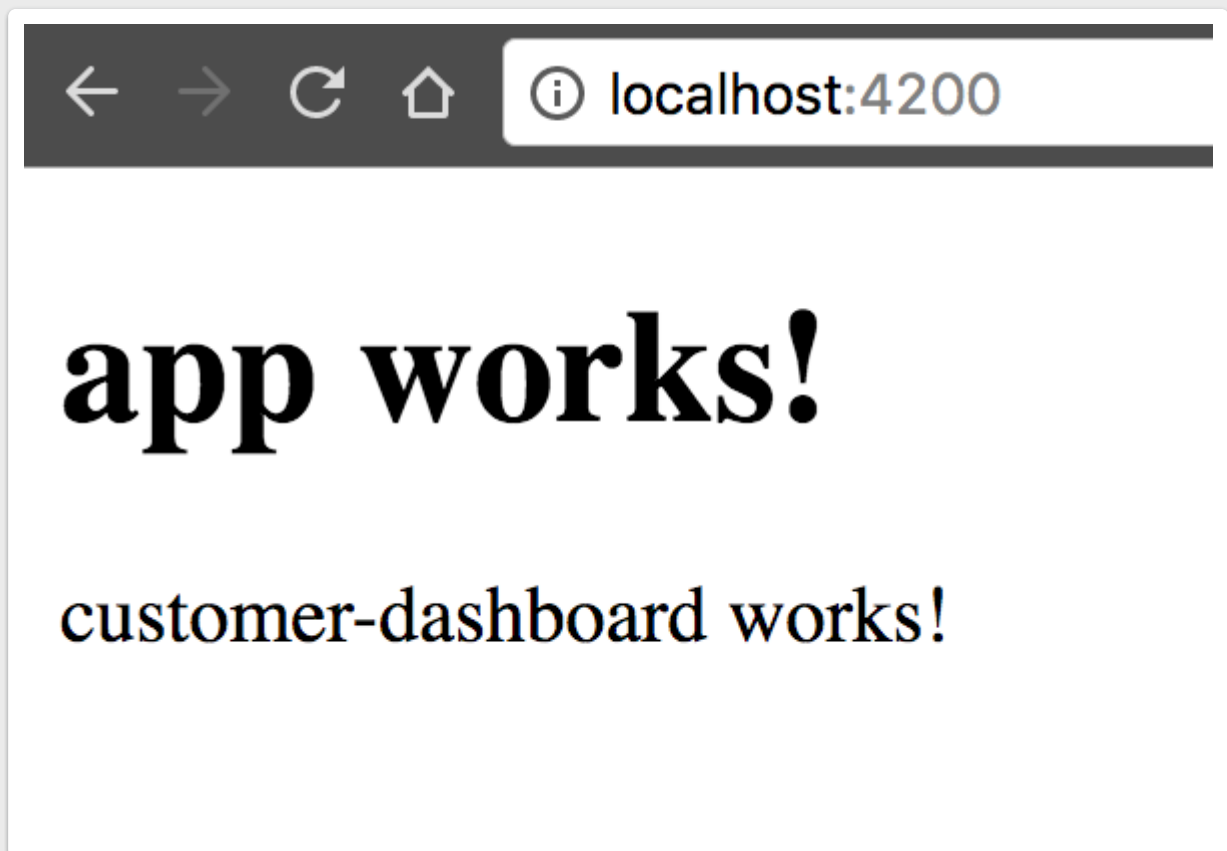
Next, in the AppComponent, app.component.html, add the tag <app-customer-dashboard>:

src/app/app.component.html

```
<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](#).

- [Providers.](#)
- [Types of Feature Modules.](#)

RESOURCES

[About](#)

[Resource Listing](#)

[Press Kit](#)

[Blog](#)

[Usage Analytics](#)

HELP

[Stack Overflow](#)

[Gitter](#)

[Report Issues](#)

[Code of Conduct](#)

COMMUNITY

[Events](#)

[Meetups](#)

[Twitter](#)

[GitHub](#)

[Contribute](#)

LANGUAGES

[简体中文版](#)

[正體中文版](#)

[日本語版](#)

[한국어](#)

Super-powered by Google ©2010-2020. Code licensed under an MIT-style License. Documentation licensed under CC BY 4.0.

Version 10.0.10-local+sha.b32126c335.