**Creating a New Project**

In this lesson, we'll create a new project for learning modules.

**We'll cover the following**

* [New project with routing](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/4958983896432640#New-project-with-routing)
* [Installing Bootstrap](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/4958983896432640#Installing-Bootstrap)

It’s time to learn about modules. Modules are a way to group a set of components that share a common purpose. They’re a great way to organize an application by feature. We’ll dive more into it throughout this section. First things first, let’s create a new project.

**New project with routing**

We’re going to be focusing on multiple topics for this project. It’s going to be larger than what we’ve done so far. We’re going to include routing.

If you’re running code locally, then you’ll need to run the following command:

ng new website --routing

During the setup process, the CLI will ask you about the default settings for the project. Typically, it will ask if you’d like to add routing. By adding the --routing option, it will skip this question and automatically install routing for you. As for the rest of the settings, you can go with the default.

Alternatively, you can just run the command without the --routing option. You can select to have routing installed during the setup. They both result in the same thing.

**Installing Bootstrap**

After installing the project, navigate to the newly created directory, and install Bootstrap. We’ll be using it to help us with styles. You can run the following command to install Bootstrap:

npm i bootstrap

We’ll need to update the styles.css file to import Bootstrap.

@import "bootstrap/dist/css/bootstrap.css"

**Exploring Modules**

In this lesson, we'll learn what modules are and what they contain.

**We'll cover the following**

* [The app module](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5068927442354176#The-app-module)
* [declarations](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5068927442354176#declarations)
* [imports](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5068927442354176#imports)
* [exports](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5068927442354176#exports)
* [providers](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5068927442354176#providers)
* [bootstrap](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5068927442354176#bootstrap)
* [Final thoughts](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5068927442354176#Final-thoughts)

In Angular, there’s a concept called modules. The idea of modules in Angular is similar to ES6 modules in JavaScript/TypeScript. For example, let’s look at a module in vanilla JavaScript.

const foo = 5;  
const chocolate = 'dark';  
  
export { chocolate };

In the example above, we have two variables: foo and chocolate. The chocolate variable is exported, which makes it **public** for other modules to use. The foo variable is **private**, which makes it inaccessible outside of the module.

The idea of modules in JavaScript is to split, share, and organize code within your project. Angular expands on this idea by building its own module system for sharing and organizing code. Angular’s module system is much more comprehensive because it allows you to isolate, export, and group code by feature.

**The app module**

By default, every project you work on will have a root module. The root module is what will be used to start the application and tie everything together. The CLI will create the root module for you. It’s called the app module.

Let’s open the app.module.ts module file in the src directory.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

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

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

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

There’s quite a bit going on here. The most important part of this code is the @NgModule() decorator. This will help Angular understand that the AppModule class is a module. We can pass in an object to configure the module. Let’s run through the options one by one.

**declarations**

Anytime we create a directive, component, or pipe, we need to register it with our project. It isn’t automatically injected into our application. We can register these classes by adding them to the declarations option.

We never had to register the classes because the CLI performed this task for us. It updated the declarations option in the app.module.ts file with whatever we generated with the CLI.

With this update, the module will expose the feature we’ve created. For example, if we created a component, that component will become available to all other components registered with the module. The same goes for pipes and directives.

This is why we never had to worry about registering the components, directives, or pipes we created. The CLI took care of registering them for us. For example, let’s look at the app.module.ts file from the pipes project.

import { BrowserModule } from '@angular/platform-browser';  
import { NgModule } from '@angular/core';  
  
import { AppComponent } from './app.component';  
import { DoublePipe } from './double.pipe';  
  
@NgModule({  
  declarations: [  
    AppComponent,  
    DoublePipe  
  ],  
  imports: [  
    BrowserModule  
  ],  
  providers: [],  
  bootstrap: [AppComponent]  
})  
export class AppModule { }

The DoublePipe class is registered in the declarations array. One important thing to understand is that, just like ES6 modules, anything defined and registered in a module is only available inside that module. The DoublePipe is not available to other modules.

**imports**

Modules can import other modules. This is how we can share things, like components, between modules. In the app.module.ts file, a module, called BrowserModule, is being imported into the module. This is a module defined by Angular to help us interact with the browser. We won’t have to worry about it.

**exports**

Even though this option is not in the app.module.ts file, it goes hand-in-hand with the imports option. The exports option allows us to specify what pieces of a module can be exported. Not everything becomes exposed the second we import a module. We need to specifically tell Angular what can be exposed.

We’ll be looking at an example later in this section.

**providers**

The providers option is deprecated. We won’t have to worry about it since it won’t be available in future versions of Angular.

**bootstrap**

The bootstrap option is an exclusive option to the root module of an application. In this case, the app module. It’s a list of components that will be inserted into the browser when the application starts.

In this example, the AppComponent is what’s inserted into the browser. We have the option of inserting multiple components, but that’s uncommon. Usually, the AppComponent will suffice for most cases.

**Final thoughts**

For the most part, you’ll never have to worry about the declarations, providers, or bootstrap options. Most of your time will be spent on the imports and exports options. We’ll be getting some practice with them in the upcoming lessons.

Lastly, it’s completely optional to use modules. We’ve built a few apps without having to create additional modules. They’re great for larger-scale applications because they allow you to separate code by feature.

**Creating a Module**

In this lesson, we'll learn how to create a module with routing capabilities.

**We'll cover the following**

* [Project overview](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6578234314981376#Project-overview)
* [Generating modules](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6578234314981376#Generating-modules)
* [The routing module](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6578234314981376#The-routing-module)

Like anything else, modules can be created with the Angular CLI. It’s the preferred and recommended way to create a module. We can run the ng generate module <name> command. The <name> portion can be replaced with the name of the module you’d like to give.

**Project overview**

The project we will be working on is a simple website. We’ll have a homepage, an about page, and a contact page. This is why we installed the router. We want to be able to switch between pages like a standard website.

If we want to, we can write everything in the AppComponent. However, that defeats the purpose of using Angular. Angular’s job is to help us split our codebase into separate files. This way, things are manageable.

One solution is to write one component per page. This works and is an entirely acceptable solution. One problem with this approach, however, is that it doesn’t scale as well. Some pages may be more complicated than others. When you have complex pages, you’ll want to break things down into smaller components.

If we just used components, the app module would have a long list of components registered under it. This would mean that other pages would have access to those modules. This can cause issues, such as longer loading times and namespace conflicts.

Therefore, it’s much better to write one module per page. Then, we can register a component to that module. This is the approach we’ll be taking.

**Generating modules**

Let’s create three modules for each page. In the command line, run the following commands:

For the homepage:

ng generate module home --routing

For the about page:

ng generate module about --routing

For the contact page:

ng generate module contact --routing

**Importing and Exporting Modules**

We'll learn how to import and export a module so that we can share components.

**We'll cover the following**

* [Creating components](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/4862619560706048#Creating-components)
* [Importing and exporting](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/4862619560706048#Importing-and-exporting)

In the previous lesson, if you run the code, nothing changes in the application. It’s still rendering the default page.

There are two reasons for this. First, modules don’t render content. That’s the job of components. Second, Angular will not register the other modules for you. Unlike components, directives, or pipes, modules need to be registered manually.

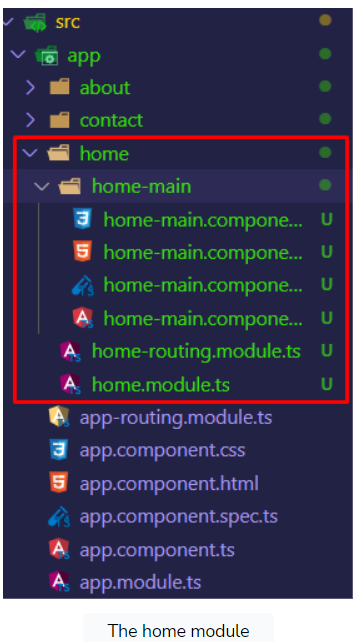
Let’s create a component that will render the content for the homepage.

## Creating components

In the command line, run the following command:

ng generate component home/home-main

The command is slightly different. We’re providing the path to the module under which we’d like to register the component. By doing so, Angular will create the component in the homes module.



Let’s look inside the home.module.ts file.

import·{·CommonModule·}·from·'@angular/common';

import·{·HomeRoutingModule·}·from·'./home-routing.module';

import·{·HomeMainComponent·}·from·'./home-main/home-main.component';

@NgModule({

··declarations:·[HomeMainComponent],

··imports:·[

····CommonModule,

····HomeRoutingModule

··]

})

export·class·HomeModule·{·}

The CLI is smart enough to register the component inside the home module instead of the app module. Let’s try using the component in the app.component.html file. Update it to the following:

<app-home-main></app-home-main>

This will not work because the component is registered with the home module. Registering components in one module doesn’t make them available in other modules. The AppComponent is registered under the app module. Therefore, it only has access to components registered under the app module.

**Importing and exporting**

In vanilla JavaScript, if we want to make variables or functions available to other files, then we need to export them. Similarly, components will need to be exported from their respective modules if we want to use them in other modules.

We can export components from modules using the exports option. Let’s update the home.module.ts file to the following:

import { CommonModule } from '@angular/common';

import { HomeRoutingModule } from './home-routing.module';

import { HomeMainComponent } from './home-main/home-main.component';

@NgModule({

  declarations: [HomeMainComponent],

  imports: [

    CommonModule,

    HomeRoutingModule

  ],

  exports: [HomeMainComponent]

})

export class HomeModule { }

In the example above, the HomeMainComponent class is being exported. It’s also registered in the declarations option. The declarations option will make the component available to other components registered under the home module. The exports option will make the component available outside of the module.

Next, we need to import the module. In the app.module.ts file, we’ll update it to the following:

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

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

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

import { HomeModule } from './home/home.module';

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    HomeModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

In the example above, we’re importing the module. We don’t need to import the component because it’s already being exported by the module. It’s better to import the module instead of the component because a single module can contain multiple components. By importing a module, we can import every component it exports.

Inside the @NgModule() directive, we’re adding the HomeModule to the imports array option. We can now use the <app-home-main> component in the app component.