The Modular design helps to keep the Separation of concerns. Keep the features together. Makes it easy to maintain the code. Makes it easier to reuse the code.



### Why are Angular modules needed?

An Angular module allows Angular to define a context for compiling templates. For example, when Angular is parsing HTML templates, it's looking for a certain list of components, directives and pipes.

Each HTML tag is compared to that list to see if a component should be applied on top of it, and the same goes for each attribute. The problem is: how does Angular know which components, directives and pipes should it be looking for while parsing the HTML?

That is when Angular modules come in, they provide that exact information in a single place.

So in summary, we can say the following about Angular modules:

* they are essential for template parsing, both in the Just In Time or Ahead Of Time Compilation scenarios as we will see
* they are also very useful simply as documentation for grouping related functionality
* They can be used to clarify which components and directives are meant to be used publicly vs internal implementation details, as we will soon see

### Angular Modules vs ES6 modules

An Angular Module is something very different than an ES6 module: An ES6 is a formalization of the typical Javascript module that the community has been using for many years: wrap private details in a closure and expose only the public API we want.

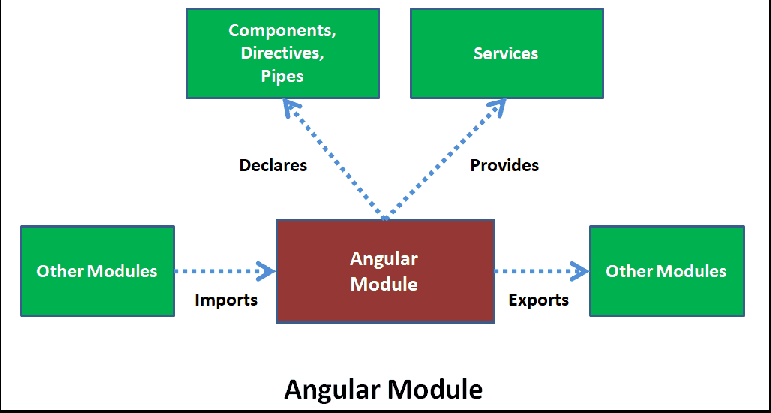
An Angular Module is mainly a template compilation context but it also helps to define a public API for a subset of functionality as well as help with the dependency injection configuration of our application.

Angular Modules are actually one of the main enablers for fast and mobile-friendly applications, more on this further. Let's now go over the different types of modules and when they should be used.

The Angular itself is built on the concept of Modules.The @angular/core is the main Angular module, which implements the Angular’s core functionality, low-level services, and utilities.

The Features like Forms, HTTP and Routing are implemented as separate Feature modules such as [FormsModule](https://www.tektutorialshub.com/angular-forms-fundamentals/), [HttpClientModule](https://www.tektutorialshub.com/introduction-to-angular-httpclient-api/), and [RouterModule](https://www.tektutorialshub.com/angular/angular-routing-navigation/). There are Many third-party libraries built around Angular such as such as [Material Design](https://material.angular.io/), [Ionic](https://ionicframework.com/), etc.

Angular Applications are assembled from the modules. The module exports [Component](https://www.tektutorialshub.com/angular/angular-component/), [directive](https://www.tektutorialshub.com/angular/angular-directives/), [service](https://www.tektutorialshub.com/angular/angular-services/), [pipe](https://www.tektutorialshub.com/angular/angular-pipes/), etc, which can be then imported in another module



An **Angular Module**is a mechanism to group components, directives, pipes and services that are related to a feature area of an angular application.

**ntroduction to Angular Modules**

* Declarations array.
* Providers array.
* Imports array.
* Exports array.
* Bootstrap.
* EntryComponents

For example, if you are building an application to manage employees, you might have the following features in your application.

|  |  |
| --- | --- |
| **Application Feature** | **Description** |
| Employee Feature | Deals with creating, reading, updating and deleting employees |
| Login Feature | Deals with login, logout, authenticate and authorize users |
| Report Feature | Deals with generating employee reports like total number of employees by department, top 10 best employees etc |

To group the components, directives, pipes and services related to a specific feature area, we create a module for each feature area.

These modules are called **feature modules**.

What are the types of feature modules?

There are five types of feature modules: **Domain, Routed, Routing, Service and Widget**. Each of them concentrates and provides a particular type of utilities.

In addition to feature modules, an Angular application also contains the following modules.

|  |  |
| --- | --- |
| **Module Type** | **Description** |
| Root Module | Every Angular application has at least one module, the root module. By default, this root application module is called AppModule. We bootstrap this root module to launch the application. If the application that you are building is a simple application with a few components, then all you need is the root module. As the application starts to grow and become complex, in addition to the root module, we may add several feature modules. We then import these feature modules into the root module. We can see here several things going on:   * the @NgModule annotation is what actually defines the module * we can list the components, directives, and pipes that are part of the module in the declarations array * we can import other modules by listing them in the imports array * we can list the services that are part of the module in the providers array but read further on why this should only be used in some cases   This declarative grouping is useful if for nothing else for organizing our view of the application and documenting which functionality is closely related. |
| Core Module | The most important use of this module is to include the providers of http services. Services in Angular are usually singletons. So to ensure that, only one instance of a given service is created across the entire application, we include all our singleton service providers in the core module. In most cases, a CoreModule is a pure services module with no declarations. The core module is then imported into the root module (AppModule) only. CoreModule should never be imported in any other module. We will discuss creating a core module in our upcoming videos |
| Shared Module | This module contains reusable components, directives, and pipes that we want to use across our application. The Shared module is then imported into specific Feature Modules as needed. The Shared module might also export the commonly used Angular modules like CommonModule, FormsModule etc. so they can be easily used across your application, without importing them in every Feature Module. We will discuss creating a shared module in our upcoming videos |
| Routing Modules | An angular application may also have one or more routing modules for application level routes and feature module routes |

**What are the advantages of splitting an angular application into multiple Angular Modules**  
  
Well, there are several benefits of Angular Modules. 

|  |  |
| --- | --- |
| **Benefit** | **Description** |
| Organizing Angular Application | First of all, Modules are a great way to organise an angular application. Every feature area is present in it's own feature module. All Shared pieces (like components, directives & pipes) are present in a Shared module. All Singleton services are present in a core module. As we clearly know what is present in each module, it's easier to understand, find and change code if required |
| Code Reuse | Modules are great way to reuse code. For example, if you have components, directives or pipes that you want to reuse, you include them in a Shared module and import it into the module where you need them rather than duplicating code. Code duplication is just plain wrong, and results in unmaintainable and error prone code. We will discuss creating a Shared module and how it can help us reuse code in our upcoming videos |
| Code Maintenance | Since Angular Modules promote code reuse and separation of concerns, they are essential for writing maintainable code in angular projects |
| Performance | Another great reason to refactor your application into modules is performance. Angular modules can be loaded either eagerly when the application starts or lazily on demand when they are actually needed or in the background. Lazy loading angular modules can significantly boost the application start up time. We will discuss lazy loading modules in our upcoming videos |

**@NgModule Decorator**  
  
As we have already discussed an Angular module is a class that is decorated with @NgModule decorator. The @NgModule decorator has the following properties.

* declarations
* bootstrap
* providers
* imports
* exports
* The following is the auto-generated code of the AppModule. The class is decorated with the @NgModule decorator

|  |
| --- |
| 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 { } |

**Creating a Module**

Let us add another Module to our above application.

Create a folder home under src/app folder

**Components**

Let us add three components. HomeComponent, AboutUsComponent, ContactUsComponent.

Create a folder called pages under the home folder. Create a three folder under pages aboutus, contactus & home.

**home/pages/aboutus/about-us.component.ts**

|  |  |
| --- | --- |
|  | import { Component } from '@angular/core';    @Component({      templateUrl: './about-us.component.html',  })  export class AboutUsComponent  {  } |

**home/pages/aboutus/about-us.component.html**

|  |  |
| --- | --- |
| 1  2  3 | <h1> About Us Page</h1> |

**home/pages/contactus/contact-us.component.ts**

|  |  |
| --- | --- |
|  | import { Component } from '@angular/core';    @Component({      templateUrl: './contact-us.component.html',  })  export class ContactUsComponent  {  } |

**home/pages/contactus/contact-us.component.html**

|  |  |
| --- | --- |
|  | <h1> Contact Us</h1> |

**home/pages/home/home.component.ts**

|  |
| --- |
| import { Component } from '@angular/core';    @Component({      templateUrl: './home.component.html',  })  export class HomeComponent  {  } |

**home/pages/home/home.component.html**

|  |  |
| --- | --- |
|  | <h1> Welcome To Module Demo</h1> |

**home/pages/index.ts**

|  |  |
| --- | --- |
|  | export \* from './aboutus/about-us.component';  export \* from './contactus/contact-us.component';  export \* from './home/home.component'; |

**Home Module**

Now, our components are ready, we can now create the HomeModule

Create the home.module.ts under the home folder

|  |
| --- |
| import { NgModule } from '@angular/core';  import { RouterModule, Routes } from '@angular/router';  import { CommonModule } from '@angular/common';    import { AboutUsComponent,ContactUsComponent,HomeComponent } from './pages';    const routes: Routes = [    {   path: '',   component: HomeComponent   },    {   path: 'home',   component: HomeComponent   },    {   path: 'contactus',   component: ContactUsComponent   },    {   path: 'aboutus',   component: AboutUsComponent   },  ];    @NgModule({    declarations: [AboutUsComponent,ContactUsComponent,HomeComponent],    imports: [      CommonModule,      RouterModule.forChild(routes),    ],    providers: [],  })  export class HomeModule { } |

We decorate the HomeModule class with @NgModule to let Angular know that it is a Angular Module

Next, use the declarations array to declare the three components that we have created.

The HomeModule do not expose any services, hence we keep the providers array empty

We are keeping the Routes in the module itself. You can create the separate routing module similar to the AppRoutingModule. That will help to keep the HomeModule code clean.

HomeModule requires CommonModule hence it is added to the imports array. The RouterModule module is imported and routes are registered with the call to forChild(routes). forChild registered the routes but does not registers any services.

**home/index.ts**

|  |  |
| --- | --- |
| 1  2  3  4 | export \* from './pages';  export \* from './home.module'; |

**Using HomeModule in AppModule**

Next step is to import the HomeModule in the AppModule. First import the HomeModule

|  |  |
| --- | --- |
|  | import { HomeModule} from './home'; |

Next, add HomeModule to the imports metadata of the @ngModule

|  |
| --- |
| imports: [      BrowserModule,      AppRoutingModule,      HomeModule    ], |

Next, we need to use the Components of the HomeModule. Open the app.component.ts and add the following code

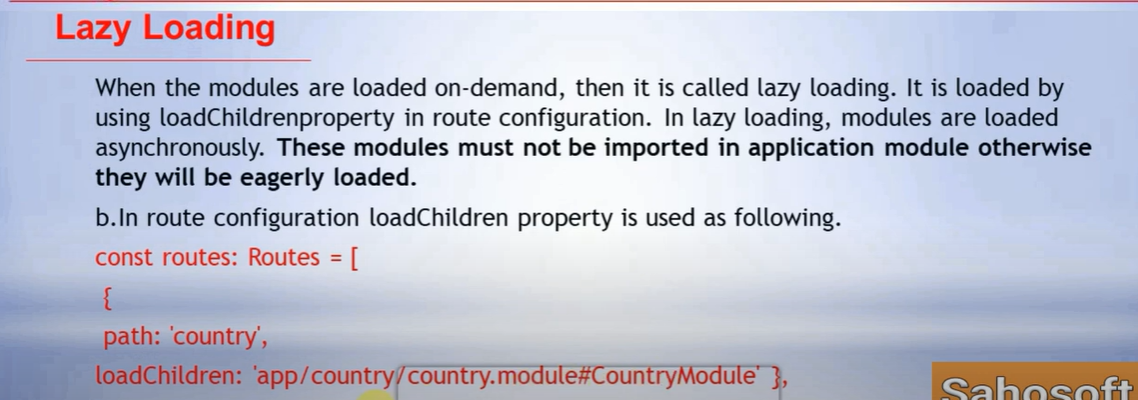
|  |  |
| --- | --- |
|  | <ul>    <li>      <a class="navbar-brand" routerLink="/">Home</a>    </li>    <li>        <a class="navbar-brand" routerLink="/aboutus">About</a>    </li>    <li>      <a class="navbar-brand" routerLink="/contactus">Contact</a>    </li>  </ul>    <router-outlet></router-outlet> |

All we have done is to use them routerLink to create a menu item.

Copy the following CSS to app.component.css

|  |  |
| --- | --- |
| 1 | ul {      list-style-type: none;      margin: 0;      padding: 0;      overflow: hidden;      background-color: #333333;  }    li {      float: left;  }    li a {      display: block;      color: white;      text-align: center;      padding: 16px;      text-decoration: none;  }    li a:hover {      background-color: #111111;  } |

Finally, run the app



In preparation for refactoring our application into multiple modules, let's create the following 2 components

* HomeComponent
* PageNotFoundComponent

Use the following Angular CLI command to create the HomeComponent

ng g c home --flat

Use the following Angular CLI command to create the PageNotFoundComponent

ng g c page-not-found --flat

If you are new to Angular CLI, please check out our Angular CLI course at the following link.  
<https://www.youtube.com/playlist?list=PL6n9fhu94yhWUcq5Pc16uf8YKXoZ87Vh_>

**Copy and paste the following code in home.component.html**

<div class="panel panel-primary">

  <div class="panel-heading">

    <h3 class="panel-title">Employee Management System</h3>

  </div>

  <div class="panel-body">

    <img src="../assets/images/Employees.jpg" class="img-responsive"/>

  </div>

</div>

**Please note :**Create **images**folder in the **assets**folder. Download the following image. Name it **Employees.jpg** and place it in the **images**folder.  
  
[](https://3.bp.blogspot.com/-ZHkznRxGNrU/XAP3RcV8O4I/AAAAAAAArcU/AlrfBPZcmKAioa7xX4a926LVRlY-ob56gCLcBGAs/s1600/Employees.jpg)  
  
**Copy and paste the following code in page-not-found.component.html**

<h1>

  The page you are looking for cannot be found.

</h1>

Include home and wild card routes in **app-routing.module.ts**

import { HomeComponent } from './home.component';

import { PageNotFoundComponent } from './page-not-found.component';

const appRoutes: Routes = [

  // home route

  { path: 'home', component: HomeComponent },

  { path: 'list', component: ListEmployeesComponent },

  { path: 'create', component: CreateEmployeeComponent },

  { path: 'edit/:id', component: CreateEmployeeComponent },

  // redirect to the home route if the client side route path is empty

  { path: '', redirectTo: '/home', pathMatch: 'full' },

  // wild card route

  { path: '\*\*', component: PageNotFoundComponent }

];

In the root component (**app.component.html**) include a menu item for the home route

<div class="container">

    <nav class="navbar navbar-default">

        <ul class="nav navbar-nav">

            <!-- Include this Home menu item for the home route-->

            <li>

                <a routerLinkActive="active" routerLink="home">Home</a>

            </li>

            <li>

                <a routerLinkActive="active" routerLink="list">List</a>

            </li>

            <li>

                <a routerLinkActive="active" routerLink="create">Create</a>

            </li>

        </ul>

    </nav>

    <router-outlet></router-outlet>

</div>