Using Shared Modules in Angular

Introduction

When working on complex applications, you often end up with a lot of feature modules. To minimize the number of these modules, Angular lets you put all the common modules and directives together in a shared module. Shared modules can help you write more organized code in less time, helping you be more productive. Shared modules are an ideal spot to declare components in order to make them reusable. You won’t have to re-import the same components in every module—you’ll just import the shared module.

In this guide, you will learn how to use shared modules to organize your code more effectively. This guide will build on another Pluralsight guide that covered [how to implement lazy loading with preloading in an application](https://app.pluralsight.com/guides/lazy-loading-angular-modules-and-preloading-strategies), using the same example to explain how to use shared modules.

Introduction to Shared Modules

In real-life projects, you need to use a lot of components in multiple feature modules. But importing the standard modules and components in all feature modules is a nightmare. You have common modules, like ReactiveFormsModule or FormsModule, and common components numbering in the hundreds or sometimes even more need to be imported. Shared modules can help you to achieve this with less code, improving your productivity.

Benefits of Shared Modules

Instead of importing these common modules and components in every feature module, you can create a shared module that has all these modules and components. Import them into a shared module and import this shared module into all feature modules. This will save imports and a lot of coding lines.

Now it's time to dive into the code and have a look into the implementation of shared modules.

Implementation of a Shared Module

Create a project to implement the shared module. The previously mentioned [guide on lazy loading](https://app.pluralsight.com/guides/lazy-loading-angular-modules-and-preloading-strategies) described how to create a project with a feature module. We will use those same modules and components in this guide.

A shared module is a type of feature module. Create a shared module using the following code:

1

$ ng g m shared --routing

shell

You have created SharedModule. The feature module FirstModule was built as described in the [previous guide](https://app.pluralsight.com/guides/lazy-loading-angular-modules-and-preloading-strategies).

Now create a shared component in the SharedModule and export it from the shared module so you can directly use it in other modules or components without importing the shared component into other modules.

Create a shared component using the snippet below:

1

$ ng g c shared/components/shared --module shared

shell

It's time to test the functionality. Use @Input() to get a name dynamically and display it in the HTML of the shared component.

shared.component.ts

1

2

3

4

5

6

7

8

9

10

11

12

13

import { Component, OnInit, Input } from "@angular/core";

@Component({

selector: "app-shared",

templateUrl: "./shared.component.html",

styleUrls: ["./shared.component.scss"]

})

export class SharedComponent implements OnInit {

@Input() name: string;

constructor() {}

ngOnInit() {}

}

ts

In the above code, an @Input() decorative is used to get a name from the parent component.

shared.component.html

1

<span>{{name}}</span>

html

As you know, you can use the component if it is registered in the module. But to use SharedComponent, you will not import it in AppModule or FirstModule. You need to export SharedComponent from ShareModule.

*shared.module.ts*

1

2

3

4

5

6

7

8

9

10

11

12

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

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

import { SharedRoutingModule } from "./shared-routing.module";

import { SharedComponent } from "./components/shared/shared.component";

@NgModule({

declarations: [SharedComponent],

imports: [CommonModule, SharedRoutingModule],

exports: [SharedComponent]

})

export class SharedModule {}

ts

The above snippet shows that SharedComponent has been exported from ShareModule.

Instead of importing the components into the other modules, import the SharedModule into other modules. Every component registered in the SharedModule can be used by other modules.

Make changes in app.component.html from the previous guide.

app.component.html

1

2

3

4

5

6

7

8

<div>

<app-shared name="App"></app-shared>

</div>

<div>

<a [routerLink]="['/first']">Go To First</a>

</div>

<router-outlet></router-outlet>

html

Add the SharedModule in the imports section of the AppModule .

app.module.ts

1

2

3

4

5

6

7

8

9

10

11

12

13

14

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

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

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

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

import { SharedModule } from "./shared/shared.module";

@NgModule({

declarations: [AppComponent],

imports: [BrowserModule, AppRoutingModule, SharedModule],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule {}

ts

Use the same configuration in FirstModule.

Make changes in first.component.html .

first.component.html

1

<app-shared name="First"></app-shared>

html

Now import the SharedModule into the FirstModule.

first.module.ts

1

2

3

4

5

6

7

8

9

10

11

12

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

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

import { FirstRoutingModule } from "./first-routing.module";

import { FirstComponent } from "./components/first/first.component";

import { SharedModule } from "../shared/shared.module";

@NgModule({

declarations: [FirstComponent],

imports: [CommonModule, FirstRoutingModule, SharedModule]

})

export class FirstModule {}

ts

You have made all the required changes to implement the shared modules. You have used a shared component without declaring it in the module. Import the SharedModule into any module where you need to use shared modules or components.

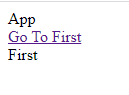
Run the application, and you can see this output: :



As you can see in the image, App is the name you've passed in the app.component.html. It means the application is working now.

Now click on *Go To First*. This will take you to FirstComponent, where you have used SharedComponent.

After clicking on it, you'll see this output:



Conclusion

Shared modules play an essential role in lazy loading, where you have a lot of modules and need to be productive. It comes down to smart work instead of hard work and impacts bundle size and performance as well. If you need another resource to learn more about this topic, [check this out](https://angular.io/guide/sharing-ngmodules).