270)Using Modules and optimizing app

Modules play a important role in optimizing our angular Project. Modules help us to increase performance of app, decrease file size and also to restructure your code a better and easier to maintain way.

271)Idea behind Modules

What’s the idea behind modules? Typically our application has couple of components, directives, services, pipes things like that. right now we only have app.module. in general ,In angular application, all these elements which make up your app have to registered in a module or in multiple modules as we will see in this course.

The idea behind this is you clearly have to tell angular,what your app consist of, what are elements of your app, which components you use,which directives you use. We have to explicit about it because angular doesn’t scan your directory or files.. It does not include all files automatically. By using modules and being clear about what you need , you are also very clear about what you don’t need. And this of course is a big thing because this allows you, in the end when it comes to building your app for production, to strip out all the things that you do not need.

This is idea behind the modules. Right now we are using only one module, this is not wrong or bad but we can use improve our app by using multiple modules. We can make it more maintainable from developer perspective but we can also use some other advantages when it comes to building the app and you will it in this section.

272)Understanding the AppModule

Here we open our app and we can see that app.module is very large.it has lot of imports like-

**Import {} from “./sumit”;**

Now these imports are typescript feature, i.e typescript needs to know where a specific thing ,class for ex, where it lives ,before we can use it. They are not related to angular modules in any way. Typescript needs these imports and in the end webpack which is used by angular cli, will go through these and bundle our app in one final bundle in the end, taking our imports and dependencies into account .but angular modules are totally different. Angular modules define how our app looks like to angular. these imports here(in app.module ) are language feature and are simply required from technical perspective .back to app.modules-

Here we got lot of import statements(discussed above) and then we got lot of declaration, imports and providers here. Now this is not a bad approach but we can improve it by using multiple modules.

In app.modules(@NgModules) we got-

1)Declarations – we tell which components, directives, pipes are used by this module.

2)Imports- here we tell that which other modules this module use. here we generally import some built in modules angular ships with .here we are also importing AppRoutingModule which defines routes for our application.

3)Providers,

4)Bootstrap

Later we will learn that there is one more thing

When we import another module ,we basically imports everything that other module exports. we can see this by example of routing module. Open app.routing module. Here we can see a additional property called exports in @NgModule. Now here we are exporting configured RouterModule. exports was this additional property in @NgModule that we said we will see later.

so when we import AppRoutingModule in app.module which exports RoutingModule, we get access to everything defined in RouterModule in AppRouting Module. Because RouterModule is exported from AppRoutingModule.

This is also true for built in modules like, FormsModule . it provides us various directives related to forms like ngMode,ngModelGroup etc. Now we dnt need to import all of them here one by one. We can simply import the whole module. HttpModule provides services like http service. So that’s idea behid modules .they bundle certain functionality and we can import them. This becomes even more important when we have our own modules.

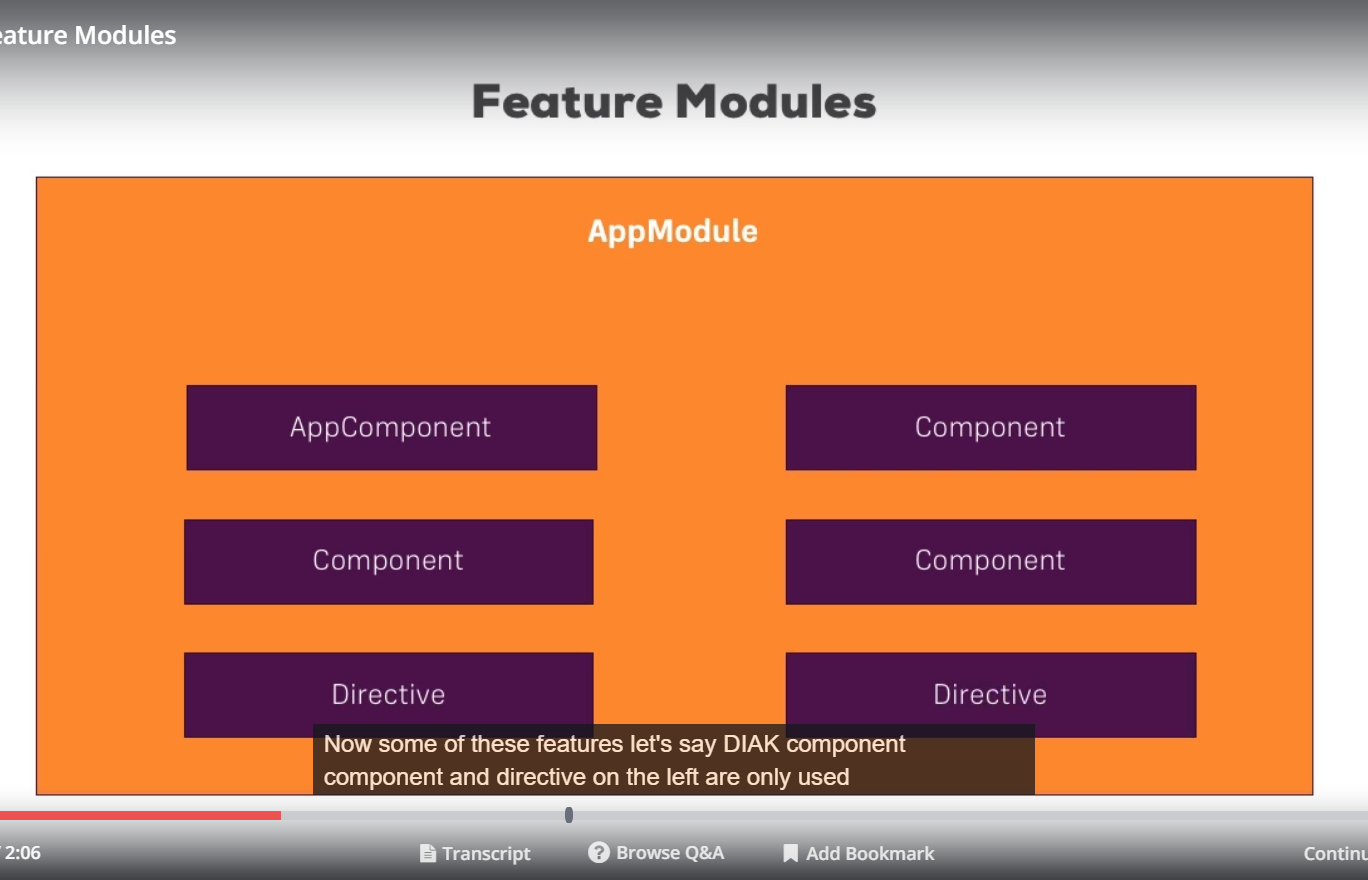
Then we have providers array.it means means which services can we use in this module and here is something important and we will come back to it. Mostly that means in our app. Here in appmodule it definately means that. services that we provide here can be used application wide. It means whole application will use same instance.

Then comes the bootstrap array. we have only when component. This component is our root component. root component is different from root module.

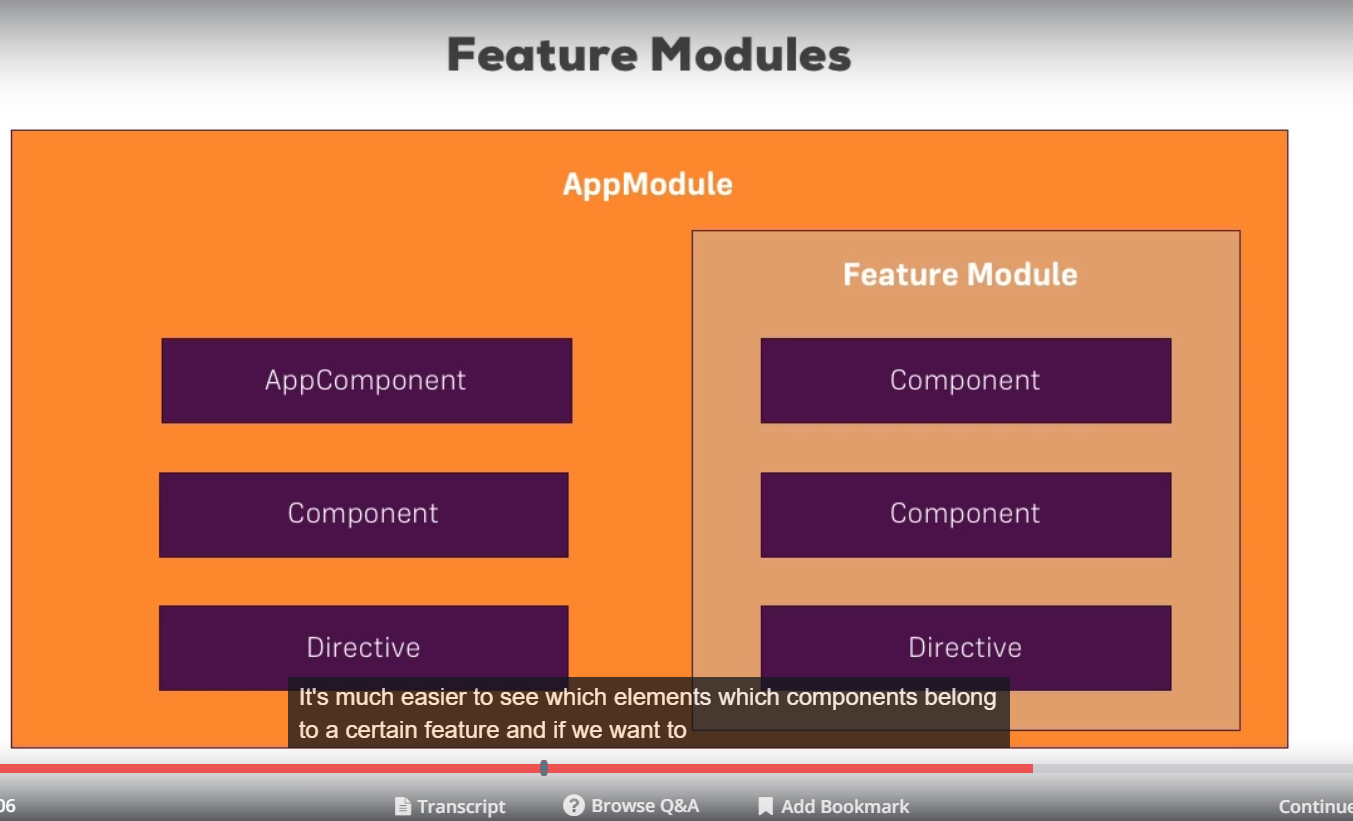
273)Understanding Feature Module

Here we will understand which modules should we build and why. Custom module that we build and can add to our application is called feature module.

Here is our application-



now it has only app.module. Our appmodule had components ,directives. lets say some of these feature (on left side of figure above) are only used to build AppComponet or really make up the root of our app. Then we may have lets say component and directives on right hand side which only really only make up feature of our app. In RecipeBook all receipe related functions will be example of it. that typically is good case to be outsourced into feature module.so we will put everything which belongs to it in its own feature module. then we will import this module in our app.component.



Here we won’t gain any performance boost. but our app.module will be more cleaner and we can manage our app easily. Lets later we want to change something in receipes functionality when we would know where to go and make changes. So right now it’s great enhancement from code structuring perspective. later we will also see how we can increase performance and decrease file size.

274)Creating a receipe feature Module

Here we will move everything that belongs to recipe feature to recipe.modulets.ts file in receipes folder. We wnt move receipe service into this module and we wil back on this later.

In receipe.module.ts first we export a class. Now we want to convert this class into module so we use @NgModule. now we will move components that are used only by recipe module. we will remove these components from app.modules and we will add them in declaration and exports section of app.module.

We won’t move RecipesService because this service is used by other parts of app too. Like in headerComponent ,when we store data we reach out to dataStorage service, now this service uses our recipeService. So we need to provide this service on application level. So it remains in app.module.

Now as a side note,technically even if you were to move recipeService into recipe module, it will still work, because **all modules getting loaded at application launch will be merged into one root injector**. So we would still basically move the provided service back to the root level automatically. That’s the default behavior of angular, we will come back to it later. Still in most cases you should not move the service to feature nodule and especially not here where you definitely want to use it in your whole application.

Now lets see apart from components and services there is anything else that we want to move from app.module to our feature module? I see one thing that we should move and 1 thing that we should copy.

We can also remove ReactiveFormsModule , because we do not clearly use it in app.module, the only place where I need access to reactive forms directive is receipe.edit component and that clearly belongs to receipes module. So we remove import of this module from app.module.

we also need to copy some things like Dropdown directive. We use this directive in recipe component as well as in other parts of apps.so we copy this directive in our root module as well as in feature module.

Then we have import property in ngModule(In feature module), here we need ReactiveFormsModule and Common Module. Common module is almost needed in every module. It gives us access to common directives like ngclass,ngModule ,ngfor,ngif etc.chances are there that almost all modules use this directive so we need to include this common directive.

now if we go to app.module then this module does not have import for commonModule. we have import of browser module. reason for it is browser module has all the features of common module + some additional feature. these additonal feature are only needed when application starts, so they are only required in app.module. **so use CommonModule in all other modules and BrowserModule in app.module.**

now lets say we import recipes.modules in app.module(we do this by adding recipe module to imports array in app.module) and we run our app. we get error that dropdown directive is part of declaration of 2 modules.

we are not allowed to declare components,pipes , directives in more than one module. we can import same module into modules, we can provide same service to different modules but we don’t duplicate our declarations.so we remove dropdown directive from feature module. and it still works, we will see why it works, shortly.

now we will get another error, this error is of routing. now we define routing module and imported it in app.module. but our recipes.component is also using

<router-outlet></router-outlet>

our recipes module got no information about routing modules. Because routing information does’nt travel down to our feature modules. These are all child routes of recipes routes.

keep in mind a module is able to use only what we define in that module. services are kind of exception. but regarding other modules and components and so on, only what’s declared and imported here(in module) can be used.

so this is not working with routing. so will have look at how we can put the routing regarding the routes related to recipes in our recipemodule.

Code-

Recipe.module-

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

import { ReactiveFormsModule } from '@angular/forms';

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

import { RecipesComponent } from './recipes.component';

import { RecipeStartComponent } from './recipe-start/recipe-start.component';

import { RecipeListComponent } from './recipe-list/recipe-list.component';

import { RecipeEditComponent } from './recipe-edit/recipe-edit.component';

import { RecipeDetailComponent } from './recipe-detail/recipe-detail.component';

import { RecipeItemComponent } from './recipe-list/recipe-item/recipe-item.component';

import { RecipesRoutingModule } from './recipes-routing.module';

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

@NgModule({

declarations: [

RecipesComponent,

RecipeStartComponent,

RecipeListComponent,

RecipeEditComponent,

RecipeDetailComponent,

RecipeItemComponent

],

imports: [

CommonModule,

ReactiveFormsModule

]

})

export class RecipesModule {}

questions asked-

<https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/questions/5313230>

<https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/questions/5934218>

275)module Import Order

n the next module, we'll add routes to a feature module. For this to work, you need to ensure that you get the import order (i.e. the order in which you add all modules to the imports[]  array in the AppModule ) right.

To be precise - unlike as shown in the video - you need to position your RecipesModule  prior to the AppRoutingModule .

So imports[]  should look like this (in AppModule ):

1. imports: [
2. ..., // Other modules
3. RecipesModule,
4. AppRoutingModule
5. ]

This is required to ensure that the Catch-all/ wildcard routes work correctly.

Why does it work in the video (even though I DON'T use that setup there)? Because I recorded this without the wildcard route - a mistake from my side.

276)Registering Routes in Feature Module

As explained in last lecture, we got child routes in component which are part of our feature module.. so we will define routes related to recipes in receipes module.so when we create a feature module, we also need to move routes related to that feature module into that module.

so we create a new module **recipes-routing.module** in recipes folder. we cut paste all our routes related to recipes from app-routing.module to this new routing module.

we cut -paste these routes-

{ path: 'recipes', component: RecipesComponent, children: [

{ path: '', component: RecipeStartComponent },

{ path: 'new', component: RecipeEditComponent, canActivate: [AuthGuard] },

{ path: ':id', component: RecipeDetailComponent },

{ path: ':id/edit', component: RecipeEditComponent, canActivate: [AuthGuard] },

] },

we also cut paste main recipes route along with its childRoutes. here we also have authGuards we simply cut paste them.

now in our new routing module(recipes-routing.module), in imports array , import **RouterModule** and (this is super important )we use **forChild**(not .forRoot) method to give it our registered roots. In app-routing.module we used **forRoot** method of Routing module. here is simple rule-

In your application you must only call .forRoot in your app.module, in your root module. If you register routes anywhere else in your app using RouterModule like we are about to do in recpies-routing.module you must use .forChild, because you are not on root router anymore, or you are not on route module anymore, i should say. But you are on child module. Every other module is a child modules because in the end everything will be imported into app module directly or because you imported a module into a module which is imported into app.module.

after this we export RoterModule(configured RouterModule) from recipe-router.module,which has our recipe routes register(as we used forChild method in imports array and passed our routes as a parameter) . In recipes.module we import this recipes-router.module. with this we have routes registered in this feature module(recipe module) which you will import in app.module. now therefore in the end these routes will also be available in our whole application, but technically we changed the way we set them up. now receip.module also has them. now if we run our application it is working fine.

I had one doubt, why we are not exporting Recipes-routing.module.ts. this is because We're only using the RecipesRoutingModule in the Recipes module, not in any other module - hence there's no need to export it from that module.

but our dropdown directive is not working in receipe-detail.component. this component is part of recipe.module. we dnt get error but it is not working anymore. reason for it is recepie.module dnt know about this directive (), as it is not in declaration array.so we cnt use this directive in receipe detail component. because it is not recognized it will be simply ignored.

Code-

Recipes-routing.module-

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

import { Routes, RouterModule } from '@angular/router';

import { AuthGuard } from '../auth/auth-guard.service';

import { RecipeEditComponent } from './recipe-edit/recipe-edit.component';

import { RecipeDetailComponent } from './recipe-detail/recipe-detail.component';

import { RecipeStartComponent } from './recipe-start/recipe-start.component';

import { RecipesComponent } from './recipes.component';

const recipesRoutes: Routes = [

{ path: 'recipes', component: RecipesComponent, children: [

{ path: '', component: RecipeStartComponent },

{ path: 'new', component: RecipeEditComponent, canActivate: [AuthGuard] },

{ path: ':id', component: RecipeDetailComponent },

{ path: ':id/edit', component: RecipeEditComponent, canActivate: [AuthGuard] },

] },

];

@NgModule({

imports: [

RouterModule.forChild(recipesRoutes)

],

exports: [RouterModule]

})

export class RecipesRoutingModule {}

**recipe.module.ts-**

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

import { ReactiveFormsModule } from '@angular/forms';

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

import { RecipesComponent } from './recipes.component';

import { RecipeStartComponent } from './recipe-start/recipe-start.component';

import { RecipeListComponent } from './recipe-list/recipe-list.component';

import { RecipeEditComponent } from './recipe-edit/recipe-edit.component';

import { RecipeDetailComponent } from './recipe-detail/recipe-detail.component';

import { RecipeItemComponent } from './recipe-list/recipe-item/recipe-item.component';

import { RecipesRoutingModule } from './recipes-routing.module';

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

@NgModule({

declarations: [

RecipesComponent,

RecipeStartComponent,

RecipeListComponent,

RecipeEditComponent,

RecipeDetailComponent,

RecipeItemComponent

],

imports: [

CommonModule,

ReactiveFormsModule,

RecipesRoutingModule

]

})

export class RecipesModule {}

**app.module.ts-**

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

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

import { HttpModule } from '@angular/http';

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

import { HeaderComponent } from './header/header.component';

import { ShoppingListService } from './shopping-list/shopping-list.service';

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

import { RecipeService } from './recipes/recipe.service';

import { DataStorageService } from './shared/data-storage.service';

import { AuthService } from './auth/auth.service';

import { AuthGuard } from './auth/auth-guard.service';

import { RecipesModule } from './recipes/recipes.modules';

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

import { ShoppingListModule } from './shopping-list/shopping-list.module';

import { AuthModule } from './auth/auth.module';

@NgModule({

declarations: [

AppComponent,

HeaderComponent

],

imports: [

BrowserModule,

HttpModule,

AppRoutingModule,

RecipesModule

],

providers: [ShoppingListService, RecipeService, DataStorageService, AuthService, AuthGuard],

bootstrap: [AppComponent]

})

export class AppModule { }

app-routing.module.ts-

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

import { Routes, RouterModule } from '@angular/router';

import { ShoppingListComponent } from './shopping-list/shopping-list.component';

const appRoutes: Routes = [

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

{ path: 'shopping-list', component: ShoppingListComponent }

];

@NgModule({

imports: [RouterModule.forRoot(appRoutes)],

exports: [RouterModule]

})

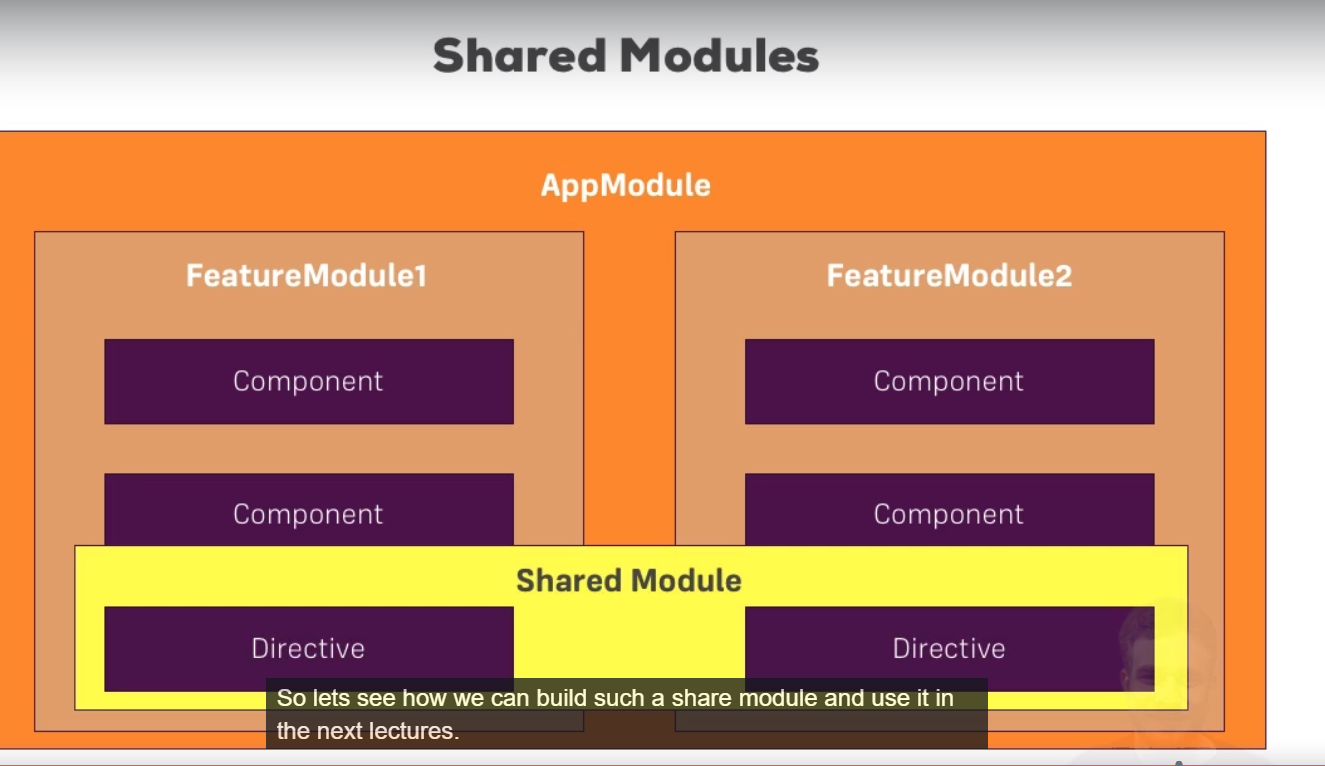
export class AppRoutingModule {

}

277)Understanding shared Module

here we will put that directive which is both in app.module and receipe.module into a shared module. this shared module will export this directive. then we will import this module in app.module and recipe.module. now both of them can use it.

we use shared module if a component is shared between 2 or more feature modules or between app.module and feature modules.



278)Creating Shared Module

we can have multiple shared modules then by name we can indicate which feature a particular shared module will provide. typically we will have only one shared module. this common module is actually a normal module, it generally differs how we use it.

so in shared folder we created shared.module.ts.

in @NgModule we also add declaration array. here we add declarations for all components ,directives or pipes we want to share basically and here we want to share Dropdown directive. so we declare it in declaration array. now we want to make it available to all modules that import this module, so we add DropDown Directive to exports array also. This far we used exports array to export configured RoutingModule. now we use it to export a dropdown directive.why?

well declaring here is important because, every directive,component,pipe that we are going to use has to be declared somewhere in your app in some module. it has to declared exactly once and only once not multiple times. however idea behind shared module is that we will now be able to import the shared module into other module. therefore to be able to use dropdown directive in other module(that import this shared module), we have to export dropdown directive from shared module. keep in mind by default everything you set in module is by default available only inside that module and is not accessible from outside. to make some components(components, directives and pipes) accessible outside, you have to export them.

We can also add something else to exports array, CommonModule. we also export common module from shared module without importing it. so while in case of components(components,directives and pipes) we have to add them in declaration array in order to export them, we can export modules directly without adding them to import array. we can add it in import array too, but it wnt give error if we even dnt.

**this is what shared module does, it provides a common module which is something that we will probably use in other modules too and most importantly in our case dropdown directive that is used in multiple modules.** And you could add more components or modules which are shared across other modules.

now we add shared Module in import array of app.module and remove the dropdown directive declaration from app.module. now directive feature is avaliable in App.Module also. we also add the import (import {} from " ") of shared module in app.module. Important -we export common module in shared module, that does’nt interfere with our browser module here (in app.module) though. We will simple use the browser module, it will overwrite the other , so that’s the no problem.

The only reason why you would export modules in shared module in the fist place is that if you have more shared modules you don’t have to list them all into components where you simply import a shared module then(03:43).

now we add import of this shared module in recipes.module also. because we also want to use this dropdwon directive in receipes module.

Questions asked-

<https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/questions/5316256>

<https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/questions/5316298>

<https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/questions/5316238>

279)Creating a shopping list feature module

Here we are create the shopping list module. we wnt provide services here bcoz i want to use them application wide.

note services are exception- you need to provide them once in module and you can use them in multple modules. like in our example we provide services in app.module but we are using them in multiple modules.

also we dnt have child routes in shoppingList , so we won’t add different module for shopping related routes. we will have route for shoppinglist component in app.routes. but our app.module do not have shoppinglist component in declaration section(we moved it to shopping-list module). so how do app.module know about it. answer is, we import shoppinglistmodule into app.module. and this shopping list module know about this shoppinglist component.

as we do refer shopping list componet in app-routing.module so we need to have import for this shopping-list compoent(import {} form "".). it is because import here is typescript feature, so typescript need to know where it can find this type we are refereing here.so it has nothing to do with angular or how angular modules works.

280)Loading Components via Selectors Vs Routing

In last lecture we left the shopping-list route in app.routing. but we do reference a shopping list component here(in app.routing.ts) and therefore we have to add a import(import {} form " ";) in this file. this is typescript thing, typescript needs to know where it can find this type. this has nothing to do with angular modules.

now we can say that shopping-list component is declared in shopping-list.module . how we can use this component in app.routing module. and both these modules(shopping list and app-routing) are not related. we declared the shopping-list compoent somewhere else than where we use it. now that's the special thing about routing and it is true for all other components also. technically app.module is also different from app.routes.module . For routing its not important that you declare components in same file as to where routes lives .its' just important that you declare them anywhere in application before you get a chance of visiting that route, which includes ,before link to that route gets rendered somewehere, which also presents the chance of going there i guess.

its different if you are loading a component via selector.

if you are loading shopping list component via selctor any where in your components define in app.module ,then you will get error, this is beacuse app.module dnt know about this component. this component is declared in shopping-list.module. you will get this type of error-

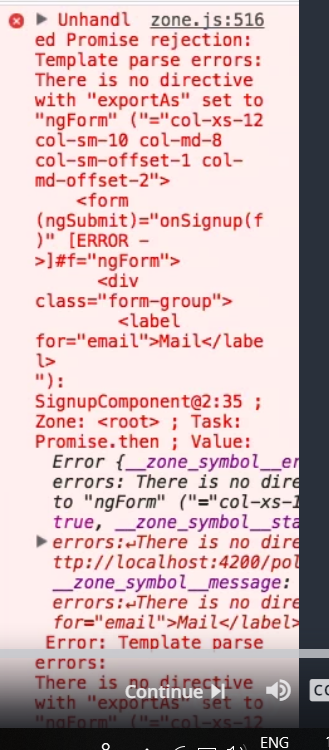
**shopping-list is not known element.**

so thats the difference between using selector and routing. for selector you have to declare it in module where you plan to use it or you have to import a another module that exports that componet,directive or pipe,like dropdowndirective. you have to get the direct connection to declaration of this thing.

for routing we dnt need to have this connection but here you need to make sure that the component you try to load here is declared somewhere and definately is declared before you try to access this route.

281)A Common Gotcha

If we run our code we get this error-



In last lecture we finished shopping-list.module. now if we run our code we get above error.

Since we are using ngModel directive in thatsignin and signup components, we got an error.so you have to Provide Forms Module(which simply unlocks the couple of directives) in every and not just one module, where we want to use that directives.

since we removed FormModule from app.module and moved to shopping-list module, we cnt access it anywhere else. One thing we could do this we can export FormsModule from shoppingList Module.

Shopping-listModule.ts-

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

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

import { FormsModule } from '@angular/forms';

import { ShoppingListComponent } from './shopping-list.component';

import { ShoppingEditComponent } from './shopping-edit/shopping-edit.component';

@NgModule({

declarations: [

ShoppingListComponent,

ShoppingEditComponent

],

imports: [

CommonModule,

FormsModule

],

exports: [FormsModule]

})

export class ShoppingListModule {}

if we do this our code will work fine, but we should’nt do this. You should’nt export a module in a feature module, so that another feature module is able to use it. You should do it in each module, so that you can see which modules a module is using. Well let’s fix it in different way though , by simply creating a different feature module for auth section, where then we can explicitly import our FormsModule.

Here is goo question about export array of module-

<https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/questions/5118968>

this question was asked by me-

<https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/questions/5317470>

282)Creating the auth feature Module

Here we created a auth module. In that module we have only 2 components signin and signp components. Then we have imports array. Here we dnt import the shared module because we do not use anything from that module, we do not use drop-down directive . but we do need FormsModule. Import this authModule in app.module and verything will work fine.

However for this auth module we can think about outsourcing our auth related routes from app.routing.module. so we create auth-routing.module file. Here we cut(or copy) and paste the routes related to login and signup component from app.routing.module to auth-routing.module.

Auth.module-

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

import { FormsModule } from '@angular/forms';

import { SigninComponent } from './signin/signin.component';

import { SignupComponent } from './signup/signup.component';

import { AuthRoutingModule } from './auth-routing.module';

@NgModule({

declarations: [

SigninComponent,

SignupComponent

],

imports: [

FormsModule,

AuthRoutingModule

]

})

export class AuthModule {}

auth-routing.module.ts-

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

import { Routes, RouterModule } from '@angular/router';

import { SignupComponent } from './signup/signup.component';

import { SigninComponent } from './signin/signin.component';

const authRoutes: Routes = [

{ path: 'signup', component: SignupComponent },

{ path: 'signin', component: SigninComponent },

];

@NgModule({

imports: [

RouterModule.forChild(authRoutes)

],

exports: [RouterModule]

})

export class AuthRoutingModule {}

we import this auth-routing.module in auth.mdoule. now since we import auth.module in app.module., these routes which are registered with forChild(which simply means for a child module and feature module is a child module, any module except for app.module is child module) since we have this setup everything should work as it was working before.

Now in next lectures, we will how we can use this modules to improve performance of app and what is role of services in modules.