**Step1**:

Lazy loaded routes need to be outside of the root app module. You will want to have your lazy loaded features in feature modules.

Create a new project with Angular Router:

1. ng new angular-lazy-loading-example --routing --style=css --skip-tests

Then navigate to the new project directory:

cd angular-lazy-loading-example

Let’s create a new feature module:

1. ng generate module shop --route shop --module app.module

Now let’s also create 3 components inside our shop feature module:

The first will be a cart component:

1. ng generate component shop/cart

The second will be a checkout component:

ng generate component shop/checkout

The third will be a confirm component:

ng generate component shop/confirm

All three components will be located in the shop directory.

**Note:** Do not import feature modules that should be lazy-loaded in your app module, otherwise they will be eager loaded.

At this point, you should have a new Angular project with a shop module and 3 components.

Step2:

[**loadChildren**](https://www.digitalocean.com/community/tutorials/angular-lazy-loading#step-2-using-loadchildren)

In your main routing configuration, you will want to do something like the following:

src/app/app-routing.module.ts

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

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

const routes: Routes = [

{ path: 'shop', loadChildren: () => import('./shop/shop.module').then(m => m.ShopModule) },

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

loadChildren expects a function that uses the dynamic import  syntax to import your lazy-loaded module only when it’s needed. The dynamic import is promise-based and gives you access to the module, where the module’s class can be called.

Step3 : Setting Route Configuration in the Feature Module.

Now all that’s left to do is to configure routes specific to the feature module.

Here’s an example:

src/app/shop/shop-routing.module.ts

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

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

import { CartComponent } from './cart/cart.component';

import { CheckoutComponent } from './checkout/checkout.component';

import { ConfirmComponent } from './confirm/confirm.component';

const routes: Routes = [

{ path: '', component: CartComponent },

{ path: 'checkout', component: CheckoutComponent },

{ path: 'confirm', component: ConfirmComponent },

];

@NgModule({

imports: [RouterModule.forChild(routes)],

exports: [RouterModule]

})

export class ShopRoutingModule { }

And finally, in the feature module itself, you’ll include your routes with RouterModule’s forChild method instead of forRoot:

shop/shop.module.ts

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

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

import { ShopRoutingModule } from './shop-routing.module';

import { ShopComponent } from './shop.component';

import { CartComponent } from './cart/cart.component';

import { CheckoutComponent } from './checkout/checkout.component';

import { ConfirmComponent } from './confirm/confirm.component';

@NgModule({

declarations: [

ShopComponent,

CartComponent,

CheckoutComponent,

ConfirmComponent,

],

imports: [

CommonModule,

ShopRoutingModule

]

})

export class ShopModule { }

App-routing-module.ts

  { path: '', component: ShopComponent },

Now you can use the routerLink directive to navigate to /shop, /shop/checkout, or /shop/confirm and the module will be loaded the first time one of these paths are navigated to.

In your terminal, start the server:

1. ng serve

This will generate a main.js file and a src\_app\_shop\_shop\_module\_ts.js file:

Output

Initial Chunk Files | Names | Size

vendor.js | vendor | 2.38 MB

polyfills.js | polyfills | 128.58 kB

main.js | main | 57.18 kB

runtime.js | runtime | 12.55 kB

styles.css | styles | 119 bytes

| Initial Total | 2.58 MB

Lazy Chunk Files | Names | Size

src\_app\_shop\_shop\_module\_ts.js | - | 10.62 kB

Next, use your browser to visit localhost:4200.

Verify that lazy loading works by opening the browser’s DevTools and looking at the Network tab. When the application initially loads at the application root, you should not observe lazy chunk files. When you navigate to a route like /shop, you should observe src\_app\_shop\_shop\_module\_ts.js.

**Note:** If it is not working right away, try restarting your server.

Your application now supports lazy loading.