Lazy Loading in Angular

https://www.youtube.com/watch?v=ATC_PY0igfg&list=PL3Fbzkmjfeadme1HOL2ROojrTP9RoHP6y&index=1&t=64s

```
Lazy loading Untitled-1
1  Lazy loading
2      Make a module
3      Make two Components
4      Use module in lazy loading way
5      Make Routing
6      Make Routing Link
```

In Angular, Lazy Loading is applied on the Routing. When we load the Angular application without Lazy Loading, all the Routes are loaded on the first load. So, if we have 1000 pages (that means 1000 routes), then all the 1000 routes are loaded as soon as we run the application and the application loads on the UI. The issue we encounter with this approach is that – 1000 routes are being loaded means that 1000 components and their data is being loaded when the application is being loaded on the first load. This makes the application load slower on the first load.

If we use Lazy Loading, then when we click on a route, it will load a specific module. So, only those modules will open for which the relevant route has been clicked. Due to this, the loading speed performance of the website is improved and the code is always optimized. Lazy Loading is very useful when we are working on a big Angular project.

Code

====

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   NUGET

PS C:\Users\anura\source\repos> ng new ng-LazyLoading
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS   [ https://sass-lang.com/documentation/syntax#scss                ]
  Sass   [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less   [ http://lesscss.org                                             ]
```

Generate a module called 'admin'. This will generate 2 files - admin-routing.module.ts and admin.module.ts.

ng g m admin --route

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   NUGET

PS C:\Users\anura\source\repos\ng-LazyLoading> ng g m admin --routing
CREATE src/app/admin/admin-routing.module.ts (248 bytes)
CREATE src/app/admin/admin.module.ts (276 bytes)
PS C:\Users\anura\source\repos\ng-LazyLoading>
```

```
EXPLORER: NG-LAZYLOADING
> .angular
> .git
> .vscode
> node_modules
v src
  v app
    v admin
      admin-routing.module.ts   U
      admin.module.ts           U
    app-routing.module.ts
    app.component.css
    app.component.html
```

So, if we think about creating an application for an Admin, we will have a couple of pages in it - Login, Listing, Profile, Edit etc.

For our needs to understand Lazy Loading, we will just create 2 components - login and listing.

ng g c admin/login

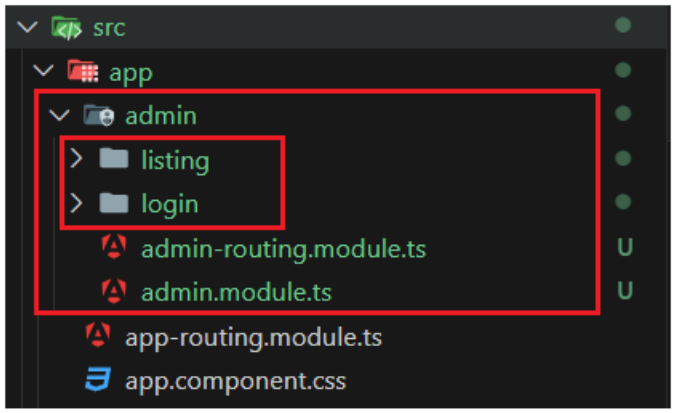
```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   NUGET

PS C:\Users\anura\source\repos\ng-LazyLoading> ng g c admin/login
CREATE src/app/admin/login/login.component.html (20 bytes)
CREATE src/app/admin/login/login.component.spec.ts (552 bytes)
CREATE src/app/admin/login/login.component.ts (198 bytes)
CREATE src/app/admin/login/login.component.css (0 bytes)
UPDATE src/app/admin/admin.module.ts (356 bytes)
PS C:\Users\anura\source\repos\ng-LazyLoading>
```

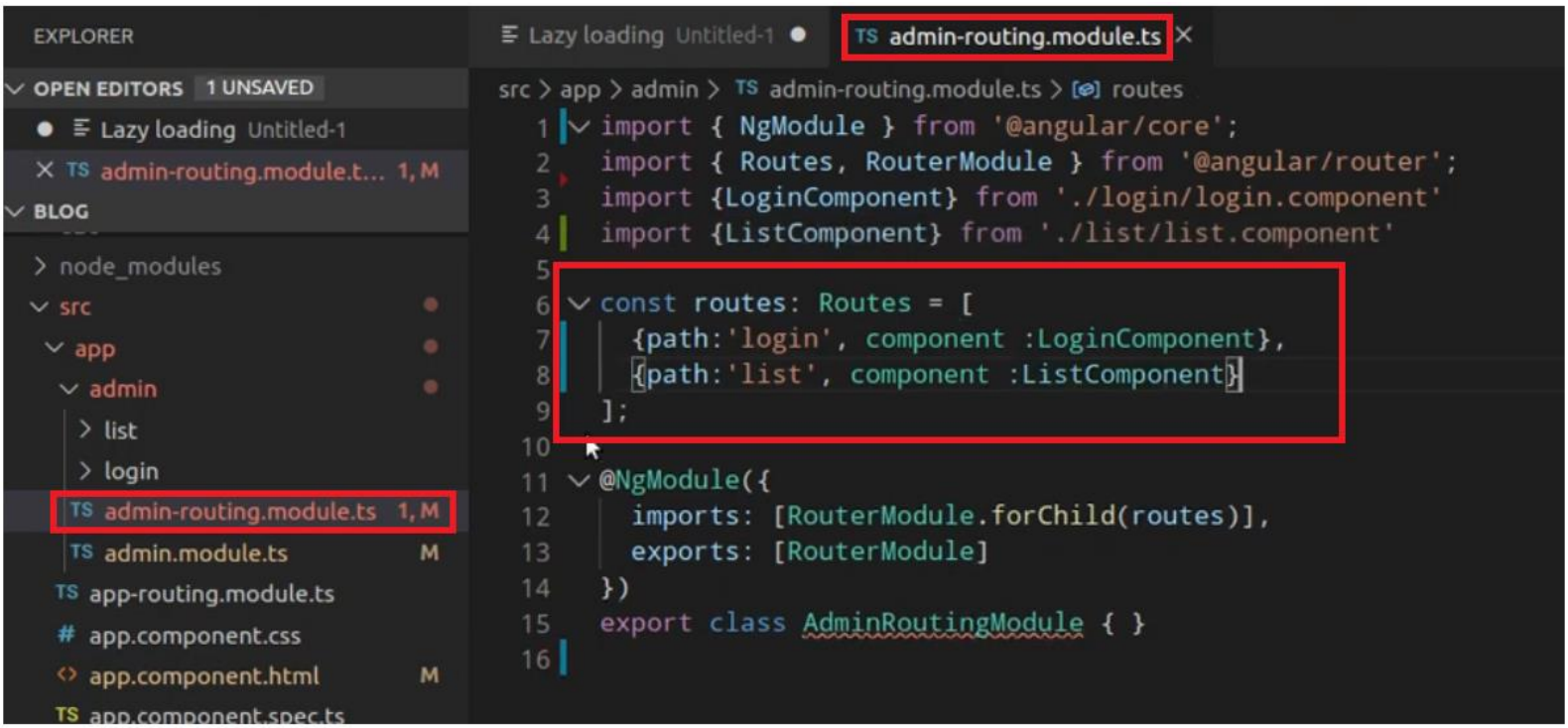
ng g c admin/listing

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    NUGET

PS C:\Users\anura\source\repos\ng-LazyLoading> ng g c admin/listing
CREATE src/app/admin/listing/listing.component.html (22 bytes)
CREATE src/app/admin/listing/listing.component.spec.ts (566 bytes)
CREATE src/app/admin/listing/listing.component.ts (206 bytes)
CREATE src/app/admin/listing/listing.component.css (0 bytes)
UPDATE src/app/admin/admin.module.ts (442 bytes)
PS C:\Users\anura\source\repos\ng-LazyLoading>
```



So, we need to add routing for our admin module.



It is very important to remember that whenever we do Lazy Loading for a module, we should never import that module (in our case it is the 'admin.module.ts') directly anywhere in the project. So, in order to Lazy Load our admin module, we will go to the 'app-routing.module.ts' and within the Routes, we will add the path for 'admin' and then use the 'loadChildren()' property which has an Arrow function using which we will call the import(). We will pass the path of our admin module in the Import() and apply '.then()' on it and pass the AdminModule to it. So, now this admin module will be loaded only when it is needed.

```
const routes: Routes = [
  { path:'admin', loadChildren: ()=>import('./admin/admin.module').then(mod => mod.AdminModule) }
];
```

EXPLORER: NG-LAZYLOADING

- .angular
- .git
- .vscode
- node_modules
- src
  - app
    - admin
      - listing
      - login
      - admin-routing.module.ts
      - admin.module.ts
      - app-routing.module.ts M
      - app.component.css
      - app.component.html

app-routing.module.ts M

src > app > app-routing.module.ts > ...

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

const routes: Routes = [

  { path:'admin', loadChildren: ()=>import('./admin/admin.module').then(mod => mod.AdminModule) }

];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

In 'app.component.html', we will add the router links for the login and list components of our admin module.

EXPLORER: NG-LAZYLOADING

- .angular
- .git
- .vscode
- node_modules
- src
  - app
    - admin
      - listing
      - login
      - admin-routing.module.ts
      - admin.module.ts
      - app-routing.module.ts M
      - app.component.css
      - app.component.html M
      - app.component.spec.ts

app-routing.module.ts M | app.component.html M

src > app > app.component.html > ...

```
{{ title }}

<ul>
  <li>
    <a routerLink="admin/login">Login</a>
  </li>
  <li>
    <a routerLink="admin/listing">Listing</a>
  </li>
</ul>

<router-outlet></router-outlet>
```

We can see that when we click on the Login link, the Login component is getting loaded on the UI. When we click on the List link, the List component is getting loaded on the UI.

NgLazyLoading

localhost:4200

Lazy Loading in Angular

- Login
- Listing

NgLazyLoading

localhost:4200/admin/login

Lazy Loading in Angular

- Login
- Listing

login works!

NgLazyLoading

localhost:4200/admin/listing

Lazy Loading in Angular

- Login
- Listing

listing works!

So, the question is what happened with the Lazy Loading? So, to verify Lazy Loading for 'admin' module, we will add a **console.warn("admin module")** in the admin module. That way, we can see exactly when the 'admin' module loads.

EXPLORER | n-routing.module.ts | app-routing.module.ts | app.component.html | admin.module.ts

OPEN EDITORS 2 UNSAVED
- Lazy loading Untitled-1
- admin-routing.module.t... 1, M
- app-routing.module.ts ... 1, M
- app.component.html src/... M
- admin.module.ts src/ap... 1, M

BLOG
- node_modules
- src
  - app
    - admin
      - list
      - login
      - admin-routing.module.ts 1, M
      - admin.module.ts 1, M
    - app-routing.module.ts 1, M
    - app.component.css
    - app.component.html M

src > app > admin > admin.module.ts > ...

```
1  import { NgModule } from '
2  import { CommonModule } fr
3
4  import { AdminRoutingModul
5  import { LoginComponent }
6  import { ListComponent } f
7  console.warn("admin module")
8
9  @NgModule({
10   declarations: [LoginComponent, ListComponent],
11   imports: [
12     CommonModule,
13     AdminRoutingModule
14   ]
15 })
16 export class AdminModule { }
17
```

warn(message?: any, ...optionalParams: any[]): void

The {@link console.warn()} function is an alias for {@link console.error()}.

NgLazyLoading | localhost:4200

Lazy Loading in Angular

- Login
- Listing

When we run the application, the console DID NOT log "admin module". That means our 'admin' module and its components i.e., login and list were not loaded.

Elements | Console | Sources | Network | Performance | Memory | Application | Security | Lighthouse | Recorder | Performance insights

top | Filter

```
[webpack-dev-server] Server started: Hot Module Replacement disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
Angular is running in development mode.
>
```

NgLazyLoading | localhost:4200/admin/login

Lazy Loading in Angular

- Login
- Listing

login works!

Now, when we clicked on 'Login' component's link, we can see that the admin module was called because the console.log("admin module") is logging 'admin module' in the console.

Also note that if we click any of these links 'Login' or 'List', the console will not log 'admin module' again. That is because the admin module got loaded when we clicked 'Login' the first time. After that, if you click on the 'Login' or 'List' links, it does not get called again because it has already loaded the first time when we clicked on 'Login' component's link.

Elements | Console | Sources | Network | Performance | Memory | Application | Security | Lighthouse | Recorder | Performance insights

top | Filter

```
[webpack-dev-server] Server started: Hot Module Replacement disabled, Live Reloading enabled, Progress disabled, Overlay enabled.
Angular is running in development mode.
▶ admin module
>
```

Let us create another module – user. Then we will create a login, listing components in this module.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    NUGET

PS C:\Users\anura\source\repos\ng-LazyLoading> ng g m user --routing
CREATE src/app/user/user-routing.module.ts (247 bytes)
CREATE src/app/user/user.module.ts (272 bytes)
PS C:\Users\anura\source\repos\ng-LazyLoading>
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    NUGET

PS C:\Users\anura\source\repos\ng-LazyLoading> ng g c user/login
CREATE src/app/user/login/login.component.html (20 bytes)
CREATE src/app/user/login/login.component.spec.ts (552 bytes)
CREATE src/app/user/login/login.component.ts (198 bytes)
CREATE src/app/user/login/login.component.css (0 bytes)
UPDATE src/app/user/user.module.ts (352 bytes)
PS C:\Users\anura\source\repos\ng-LazyLoading>
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    NUGET

PS C:\Users\anura\source\repos\ng-LazyLoading> ng g c user/listing
CREATE src/app/user/listing/listing.component.html (22 bytes)
CREATE src/app/user/listing/listing.component.spec.ts (566 bytes)
CREATE src/app/user/listing/listing.component.ts (206 bytes)
CREATE src/app/user/listing/listing.component.css (0 bytes)
UPDATE src/app/user/user.module.ts (438 bytes)
PS C:\Users\anura\source\repos\ng-LazyLoading>
```

Add routes for login, listing components which are in the user module to the uer-routing.module.ts file.

user-routing.module.ts M

src > app > user > user-routing.module.ts > ...

```
1  import { NgModule } from '@angular/core';
2  import { RouterModule, Routes } from '@angular/router';
3  import { LoginComponent } from './login/login.component';
4  import { ListingComponent } from './listing/listing.component';
5
6  const routes: Routes = [
7    {path:'login', component:LoginComponent},
8    {path:'listing', component:ListingComponent},
9  ];
10
11 @NgModule({
12   imports: [RouterModule.forChild(routes)],
13   exports: [RouterModule]
14 })
15 export class UserRoutingModule { }
16
```

Adding Lazy Loading for user module.

app-routing.module.ts M

src > app > app-routing.module.ts > ...

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

const routes: Routes = [

  { path:'admin', loadChildren: ()=>import('./admin/admin.module').then(mod => mod.AdminModule) },
  { path:'user', loadChildren: ()=>import('./user/user.module').then(mod => mod.UserModule) }

];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

UI change for admin module's login and listing template – adding 'admin - ' to identify that this is the user's login or listing UI template.

login.component.html

src > app > admin > login > login.component.html > ...

```
<p>admin - login works!</p>
```

listing.component.html

src > app > admin > listing > listing.component.html > ...

```
<p>admin - listing works!</p>
```

UI change for user module's login and listing template – adding 'user - ' to identify that this is the user's login or listing UI template.

login.component.html M | listing.component.html M

src > app > user > login > login.component.html > ...

```
<p>user - login works!</p>
```

login.component.html M | listing.component.html M

src > app > user > listing > listing.component.html > ...

```
<p>user - listing works!</p>
```

When we run the application, we can see that the Admin or User modules were not loaded at the start of the application – the proof of that is that the 'console.warn()' command present in both these modules were not executed for admin and user modules.

localhost:4200

Lazy Loading in Angular

Admin Links

- Login
- Listing

User Links

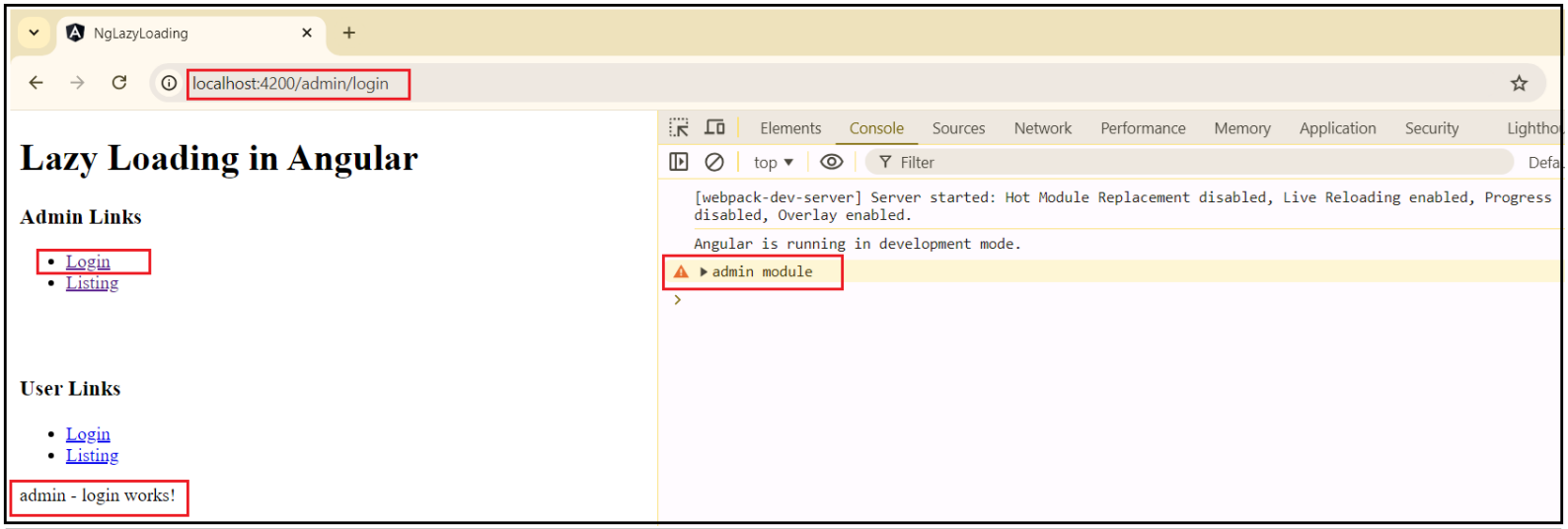
- Login
- Listing

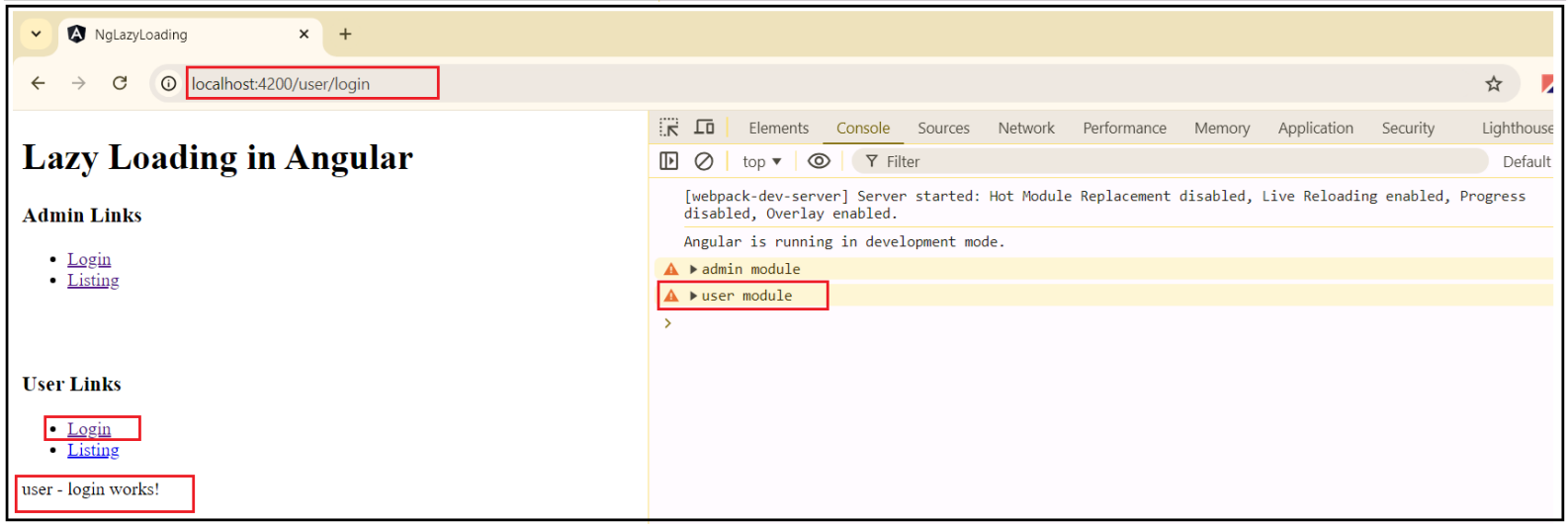
Elements | Console | Sources | Network | Performance | Memory | Application | Security | Lighthouse

top | Filter | Default levels | No Issues

```
[webpack-dev-server] Server started: Hot Module Replacement disabled, Live Reloading enabled, Progress disabled, Overlay enabled.   index.js:485
Angular is running in development mode.   core.mjs:26656
>
```

When we run the application, we can see that console has not thrown a message 'admin module' or 'user module'. That is because we are loading these modules and their components in a Lazy Loading way.





What is forRoot() and forChild()?

In Angular, forRoot() and forChild() are static methods commonly used with modules that provide services or functionality that need to be configured differently at the root and feature module levels.

- **forRoot()** :
  - Used in the root module (usually AppModule) to configure services for the entire application.
  - It typically provides singleton services and ensures that only one instance of the service is available throughout the app.
  - The forRoot method often accepts configuration options to customize the service's behavior.
  - Example: RouterModule.forRoot(routes) configures the root-level routing for the application.
- **forChild()** :
  - Used in feature modules to configure services specific to that module.
  - It allows you to provide different configurations or instances of a service for different feature modules.
  - It ensures that services are scoped to the feature module and its children.
  - Example: RouterModule.forChild(routes) configures routing for a specific feature module.

- **Key Differences between forRoot() and forChild() :**
  - **Scope** : forRoot is application-wide, while forChild is module-specific.
  - **Services** : forRoot provides singleton services, while forChild can provide scoped instances.
  - **Configuration** : forRoot often accepts configuration options, while forChild might have different configuration parameters depending on the module's needs.
  - **Usage** : forRoot is used once in the root module, while forChild can be used in multiple feature modules.

app-routing.module.ts

src > app > app-routing.module.ts > ...

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

const routes: Routes = [

  { path:'admin', loadChildren: ()=>import('./admin/admin.module').then(mod => mod.AdminModule) },
  { path:'user', loadChildren: ()=>import('./user/user.module').then(mod => mod.UserModule) }

];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

admin-routing.module.ts

src > app > admin > admin-routing.module.ts > ...

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { LoginComponent } from './login/login.component';
import { ListingComponent } from './listing/listing.component';

const routes: Routes = [
  {path:'login', component:LoginComponent},
  {path:'listing', component:ListingComponent},
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class AdminRoutingModule { }
```

user-routing.module.ts

src > app > user > user-routing.module.ts > ...

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { LoginComponent } from './login/login.component';
import { ListingComponent } from './listing/listing.component';

const routes: Routes = [
  {path:'login', component:LoginComponent},
  {path:'listing', component:ListingComponent},
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class UserRoutingModule { }
```

What is the importance of 'loadChildren'?

- **loadChildren** is a **property** in Angular which allows us to implement Lazy Loading in an Angular applications.
  - Using 'loadChildren' property optimizes the application's performance - it will only load the nested route subtree when user navigates to a particular URL that matches the route path of a component which lies within a child module.
  - It helps in keeping the nested routes table separate.
  - We need to specify a routing module for loadChildren property. This module must define the routes and should import all relevant modules.
  - If you use **import(<module-path>).then(module => module.<routing-module>)**, Angular will create a separate js bundle that will be loaded only when a child route is activated. And you get better performance, code readability, and maintainability.
  - If you use **() => <routing-module>**, angular will not create a separate js bundle, but the routes table would be kept separate. The result is better code readability and maintainability.

- By default, NgModules are eagerly loaded.
- As soon as the application loads, so do all the NgModules.
- Lazy loading reduces the bundle sizes.
- Lazy loading decreases load time.
- To lazy load Angular modules, we use loadChildren