

Angular – Modules and Lazy Loading

Ground Rules

For a successful class, please:

- Arrive on time.
- Turn off cell phones.
- Assist your colleagues; show respect to all individuals regardless of their skill and knowledge level.
- Do not use class time to surf the net, check e-mail, or use instant messaging.
- Adhere to attendance policy as directed by your local training coordinator.
- Make the most of face-to-face training; ask questions and share your own insights and expertise.
- Leave the room only during break times or in the event of an emergency.
- Manage expectations for your other responsibilities.



Module Objectives

At the end of this module, you should be able to:

- Understand an overview of Modules
- Understand features of modules
- Understand an introduction to Lazy Loading



Topic List

#No	Module Topics
1	Modules Overview
2	Feature Modules
3	Introduction to Lazy Loading



Topic List

#No	Module Topics
1	Modules Overview
2	Feature Modules
3	Introduction to Lazy Loading

Module Overview

Module Overview

What is a Module?

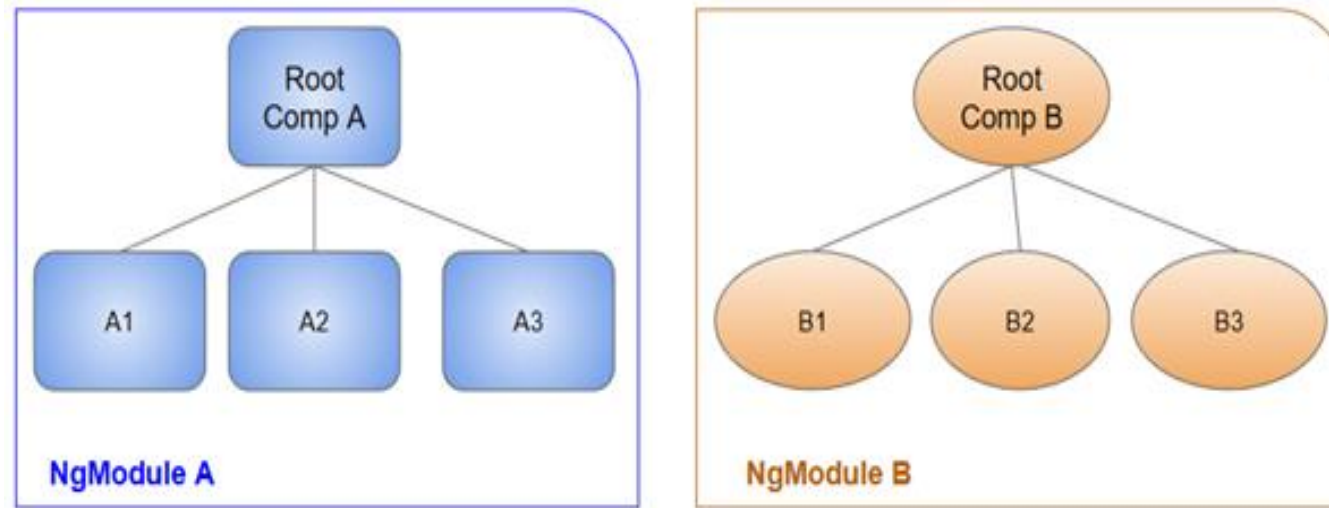
Angular applications are Modular with its own modularity system called “**NgModules**”.

- Creates an application by grouping Angular’s building blocks like Components, Directives, Pipes , Services with other modules
- In every Angular application there has to be at least one NgModule class
- The AppModule is called as the **Root Module**
- These NgModules provide a compilation context for the components
- A root component is bootstrapped to every root NgModule



Module Overview

What is a Module?



Source ref : <https://angular.io/guide/architecture-modules>

Module Overview

Modules loading in Angular

- In Angular, the modules loading can be categorized in two ways
- They are
 - Eagerly Loaded Modules
 - Lazy Loaded Modules

Eagerly Loaded Modules	Lazy Loaded Modules
Imported in the root AppModule	Can be imported only when it is required
Bundles gets downloaded initially along with root module	Bundles are downloaded until the corresponding module is not needed
Suited for smaller apps	Optimal when the application is bigger



Topic List

#No	Module Topics
1	Modules Overview
2	Feature Modules
3	Introduction to Lazy Loading

Feature Modules

Feature Modules

- Feature Modules are a special kind of Ng Modules in Angular
- They serve the purpose of organizing code for a specific feature
- They can be created using an Angular CLI command
 - **ng generate module**
- A user defined feature module is imported from **CommonModule** rather than the **BrowserModule**.



Feature Modules

Benefits

- Feature modules are used for

Separates code for a specific functionality

Partition the application into key focus area

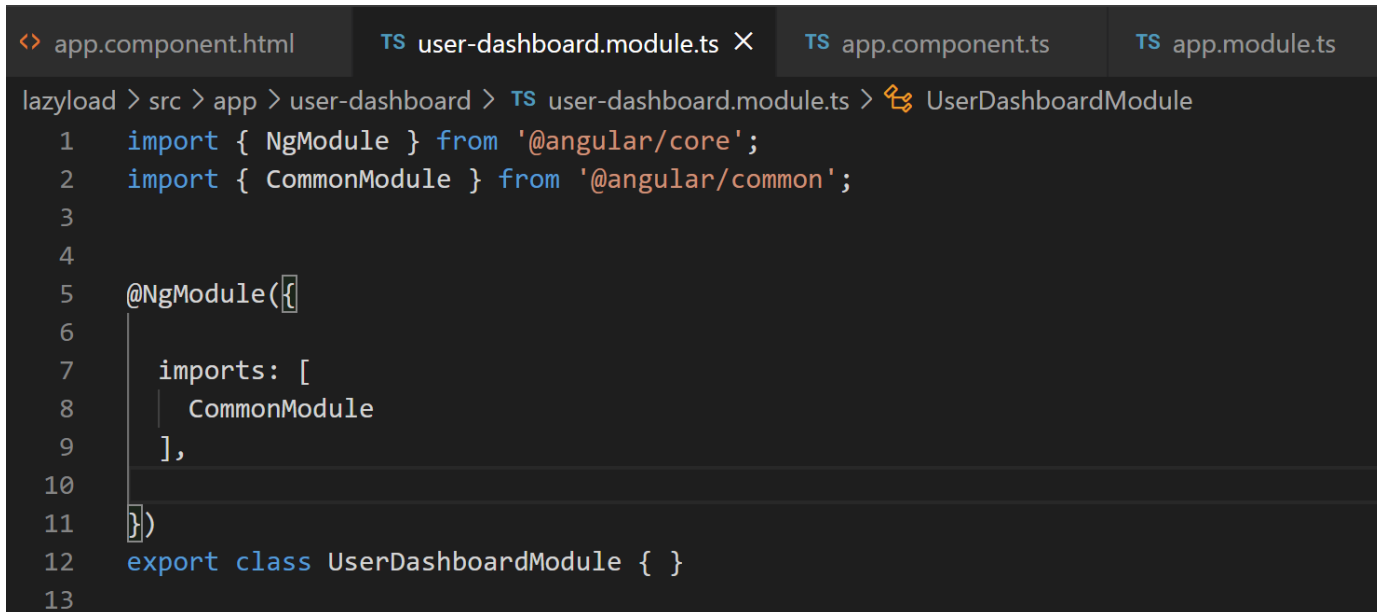
Collaborates with root module and other modules

Separates features from each other

Feature Modules

DEMO

- **Step 1** : Create a new project `ng new FeatureModule`
- **Step 2** : Create a new feature module using the Angular CLI command
 - `ng generate module UserDashboard`
- The CLI creates a folder name “UserDashboard” with a file “user-dashboard.module.ts”



The screenshot shows an IDE with four tabs: `app.component.html`, `TS user-dashboard.module.ts` (active), `TS app.component.ts`, and `TS app.module.ts`. The active tab displays the following TypeScript code for `user-dashboard.module.ts`:

```
lazyload > src > app > user-dashboard > TS user-dashboard.module.ts > UserDashboardModule
1  import { NgModule } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3
4
5  @NgModule({
6
7    imports: [
8      CommonModule
9    ],
10
11  })
12  export class UserDashboardModule { }
13
```



Feature Modules

DEMO

- **Step 3** : : Create a new component in the same path where feature module is created

ng generate component user-dashboard/UserDashboard

This creates the UserDashboardComponent folder

The screenshot shows an IDE with several tabs: `app.component.html`, `TS user-dashboard.module.ts`, `TS app.component.ts`, and `TS app.module.ts`. The active file is `user-dashboard.module.ts`, which contains the following code:

```
1 import { NgModule } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { UserDashboardComponent } from './user-dashboard/user-dashboard.component';
4
5 @NgModule({
6   declarations: [UserDashboardComponent],
7   imports: [
8     CommonModule
9   ],
10  exports: [UserDashboardComponent]
11 })
12 export class UserDashboardModule { }
13
```

Two callout boxes provide additional context:

- A callout pointing to the `declarations` array in the `@NgModule` decorator: "Angular will associate this component with feature module".
- A callout pointing to the `exports` array in the `@NgModule` decorator: "Add this entry to make this visible in the App component".

In the top right corner, a file explorer shows the project structure:

- `app`
 - `user-dashboard`
 - `user-dashboard`
 - `# user-dashboard.c...` (U)
 - `<> user-dashboard.c...` (U)
 - `TS user-dashboard.c...` (U)
 - `TS user-dashboard.c...` (U)
 - `TS user-dashboard.m...` (U)
 - `TS app-routing.module.ts`
 - `# app.component.css`

Feature Modules

DEMO

- **Step 4 :** Import the feature module in the app.module.ts by adding it in the Imports array

```
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { HttpClientModule } from '@angular/common/http';
import { AppComponent } from './app.component';
import { UserDashboardModule } from './user-dashboard/user-dashboard.module';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    UserDashboardModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Add this entry to import
the feature module



Feature Modules

DEMO

- **Step 5 :** Render the feature module's component template

```
<div>
  <h1 style="color:green">
    Welcome To User Dashboard Feature Module Compone
nt
  </h1>
</div>
```

- **Step 6:** Update the app.component.html with the feature component selector value

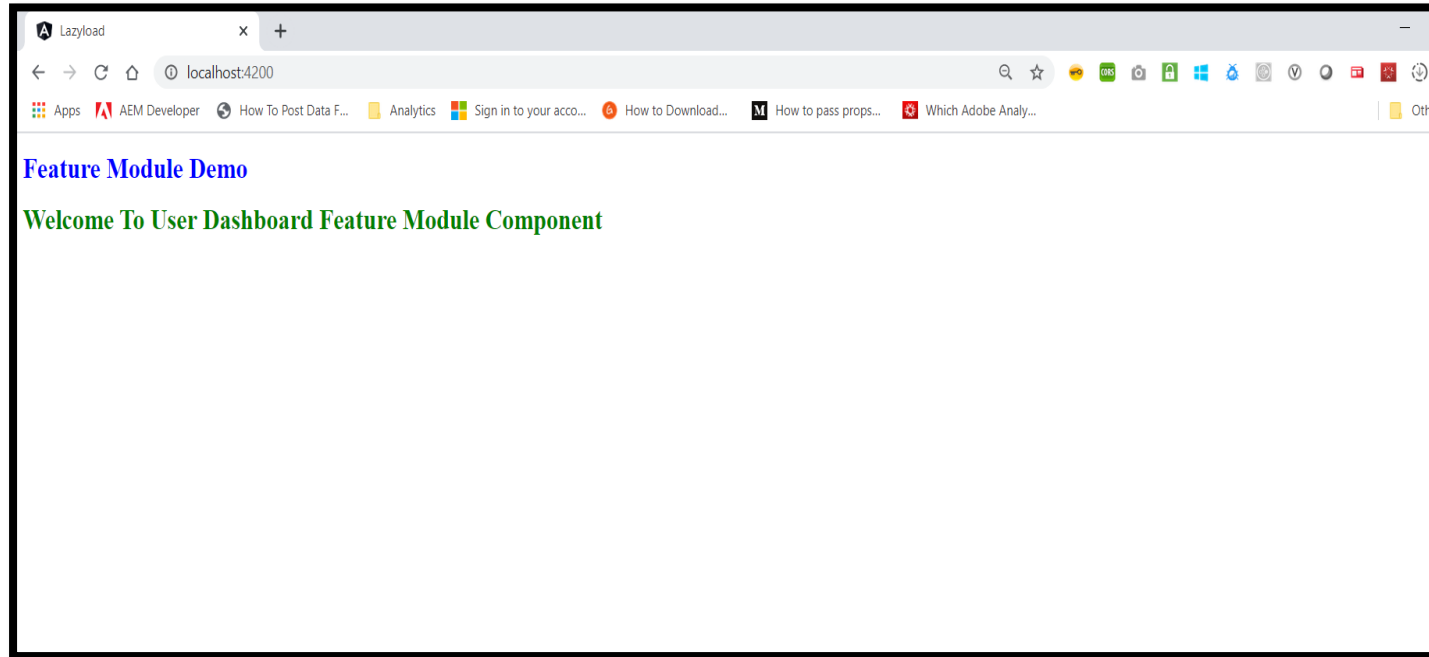
```
<h1 style="color:blue">{{title}}</h1>

<app-user-dashboard></app-user-dashboard>
```



Feature Modules

OUTPUT



Topic List

#No	Module Topics
1	Modules Overview
2	Feature Modules
3	Introduction to Lazy Loading

Introduction to Lazy Loading

Lazy Loading

- NgModules are eagerly loaded by default
- Once an Angular app is loaded, all the NgModules gets loaded irrespective of their need in the application
- If this is the case, then this will decrease the performance for larger applications with more number of routes
- Hence, a solution to this problem is **Lazy Loading**
- Lazy loading is a design pattern that loads the NgModules as and when required
- This process helps to keep the initial bundle sizes smaller to decrease the load times



Lazy Loading

Steps to create Lazy Loading

- Let's understand here the step- by-step procedure to setup the Lazy loaded feature module
 - Create the feature module with routing using Angular CLI
 - Configure the routes

Lazy Loading

DEMO

- Let's create two lazy loadable modules named "Employee" and "Department"
- **Step 1: Create the feature module with routing using Angular CLI**

- - Create feature module "employee" and its corresponding component files using the below command

```
ng generate module employee --route employee --module app.module
```

- **Step 2: Configure the routes**

- The route is declared in the app-routing.module.ts file by Angular CLI for the lazy load modules
- loadChildren() is used to declare the route.

```
const routes: Routes = [  
  { path: 'employee',  
    loadChildren: () => import('./employee/employee.module').then(m => m.EmployeeMod  
ule)  
  },  
];
```



Lazy Loading

DEMO

- Create another feature module “department” and its corresponding component files using the below command

```
ng generate module department --route department --module app.module
```

- After the two feature modules are created below is the updated app-routing.module.ts file

```
const routes: Routes = [  
  {  
    path: 'employee',  
    loadChildren: () => import('./employee/employee.module').then(m => m.Employee  
Module)  
  },  
  {  
    path: 'department',  
    loadChildren: () => import('./department/department.module').then(m => m.Depart  
mentModule) }  
];
```


Lazy Loading

DEMO

- The feature modules **employee.module.ts** and **department.module.ts** files are done with the necessary changes by the Angular CLI

[employee.module.ts](#)

[department.module.ts](#)

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { EmployeeRoutingModule } from './employee-routing.module';
import { EmployeeComponent } from './employee.component';

@NgModule({
  declarations: [EmployeeComponent],
  imports: [
    CommonModule,
    EmployeeRoutingModule
  ]
})
export class EmployeeModule { }
```

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DepartmentRoutingModule } from './department-routing.module';
import { DepartmentComponent } from './department.component';

@NgModule({
  declarations: [DepartmentComponent],
  imports: [
    CommonModule,
    DepartmentRoutingModule
  ]
})
export class DepartmentModule { }
```

Lazy Loading

DEMO

- The feature modules **employee-routing.module.ts** and **department-routing.module.ts** files are done with the necessary changes by the Angular CLI

employee-routing.module.ts

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

const routes: Routes = [
  { path: '', component: EmployeeComponent }];

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

department-routing.module.ts

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

const routes: Routes = [{ path: '', component: DepartmentComponent }];

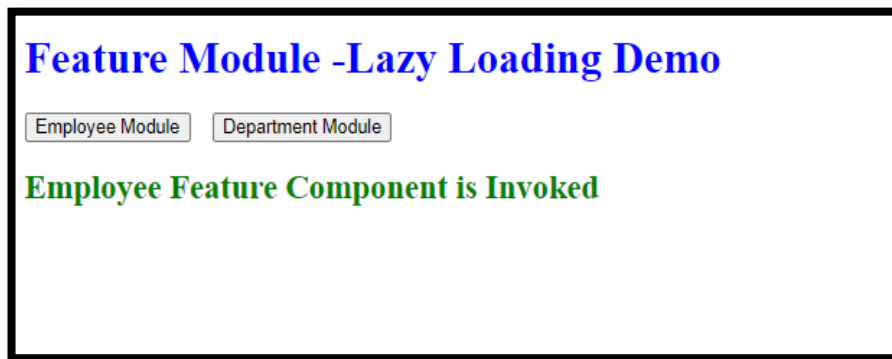
@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class DepartmentRoutingModule { }
```

Lazy Loading

OUTPUT

- The feature modules **employee-routing.module.ts** and **department-routing.module.ts** files are done with the necessary changes by the Angular CLI

When EmployeeModule is clicked



When DepartmentModule is clicked



Module Summary

Now, you should be able to:

- Understand an overview of Modules
- Understand features of modules
- Understand an introduction to Lazy Loading



Reference

Heading	Description
Lazy Loading	Link

Thank You

