Angular Application with Lazy Loading

lazy loading is the process of loading modules(images, videos, documents, JS, CSS, etc) on-demand.

The most important concepts of application performance are: **Response Time**, and **Resources Consumption**.

The prospect of Lazy Loading in Angular helps reduce the risk of some of the web app performance problems to a minimal.

Lazy Loading does well to check the concepts we listed above:

Response Time:
This is the amount of time it takes the web application to load and the UI interface to be responsive to users. Lazy loading optimises response time by code splitting and loading the desired bundle.

Resources Consumption:
Web apps should not take long time to load. So, to reduce the amount of resources loading, lazy loading loads the code bundle necessary at a time.

Lazy loading speeds up our application load time by splitting it into multiple bundles and loading them on demand.

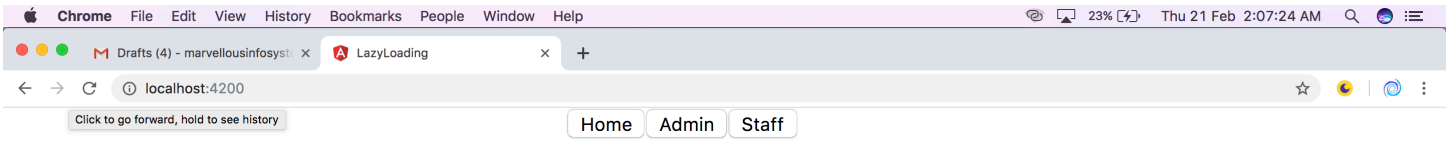
Advantages of lazy loading:

- High performance in bootstrap time on initial load.
- Modules are grouped according to their functionality.
- Smaller code bundles to download on initial load.
- Activate/download a code module by navigating to a route.

Working of Lazy Loading :
Lazy Loading generally, is a concept where we delay loading of an object until it is needed.
In Angular, all the JavaScript components declared in the declarations array app.module.ts are bundled and loaded in one fell swoop when a user visits our site.
But in case of Lazy loading specific module sets loaded whenever required or accessed by the user.

Steps to create Angular application with Lazy Loading

Consider attached Angular application which contains two modules as Admin and Staff



Marvellous Infosystems

Lazy Loading in Angular

Step 1 :
Creating an Angular project
ng new Marvellous_LazyLoading

Step 2 :
Creating a Feature Module
Our application contains 2 feature modules as Admin and staff.

We now need to create a feature module using the following command:
$ ng gennerate module Admin
$ ng gennerate module Staff

Step 3 :
We also need to create components inside our feature module:

$ ng generate component Admin/admin-dashboard
$ ng generate component Admin/admin-task
$ ng generate component Staff/Staff-dashboard

These commands will generate three components inside the Admin & Staff module.

Step 4 :
Using loadChildren()
In the main routing file app-routing.module.ts, we need to use the loadChildren() method to lazy load the feature module:

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

const routes: Routes = [
  {
    path: 'admin',
```

```
    loadChildren: './admin/admin.module#AdminModule'
  },
  {
    path: 'staff',
    loadChildren: './staff/staff.module#StaffModule'
  },
  {
    path: '',
    redirectTo: '',
    pathMatch: 'full'
  }
];

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

The loadChildren() method takes the path to the module, appended to # appended to the module's class name.

Step 5 :
Routing Components Inside the Admin Feature Module

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AdminDashboardComponent } from './admin-dashboard/admin-dashboard.component';

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

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

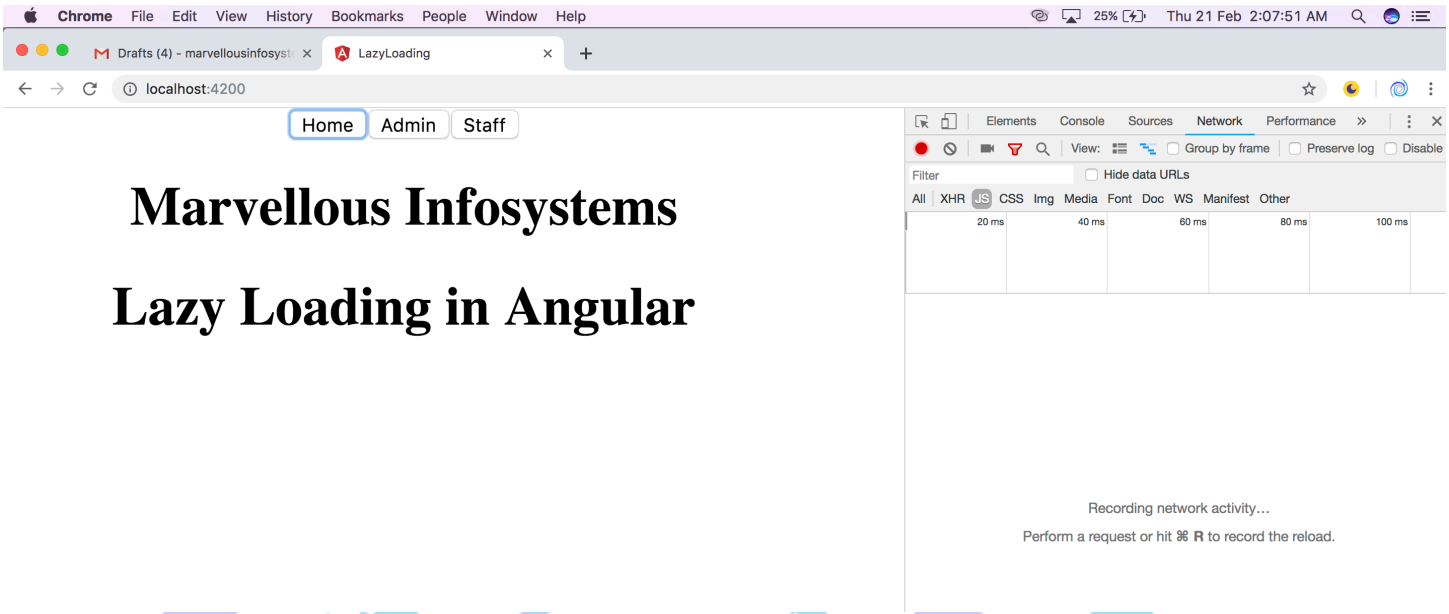
Routing Components Inside the Staff Feature Module

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { StaffDashboardComponent } from './staff-dashboard/staff-dashboard.component';

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

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

export class StaffRoutingModule { }

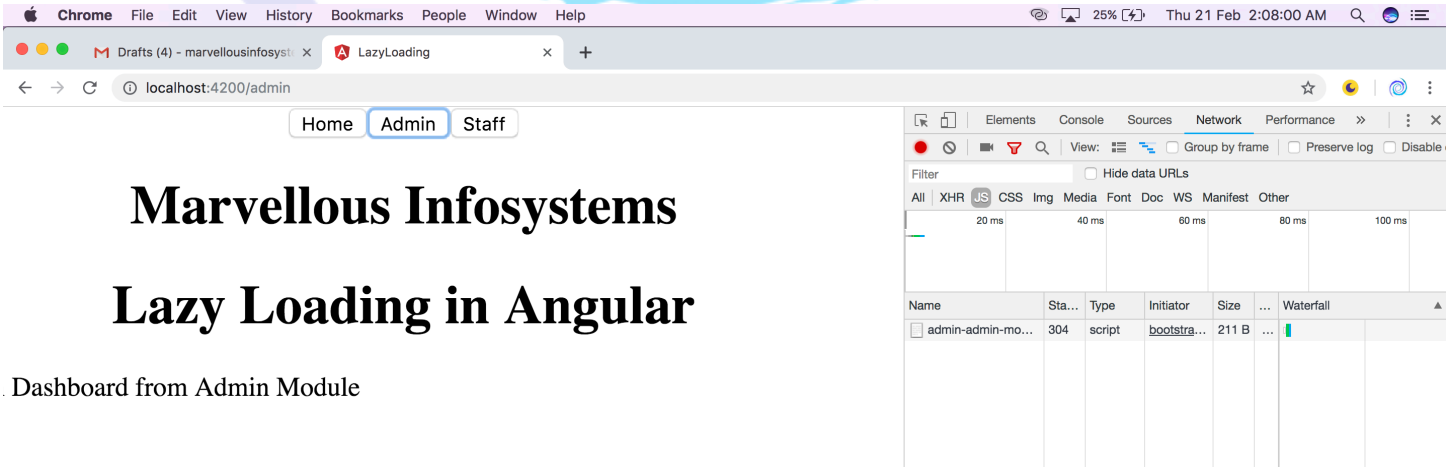
In the feature module, we include the routes with RouterModule's forChild() method instead of the forRoot() method.

Once everything is working and compiling as expected, open chrome dev tools / Network tab and reload the page



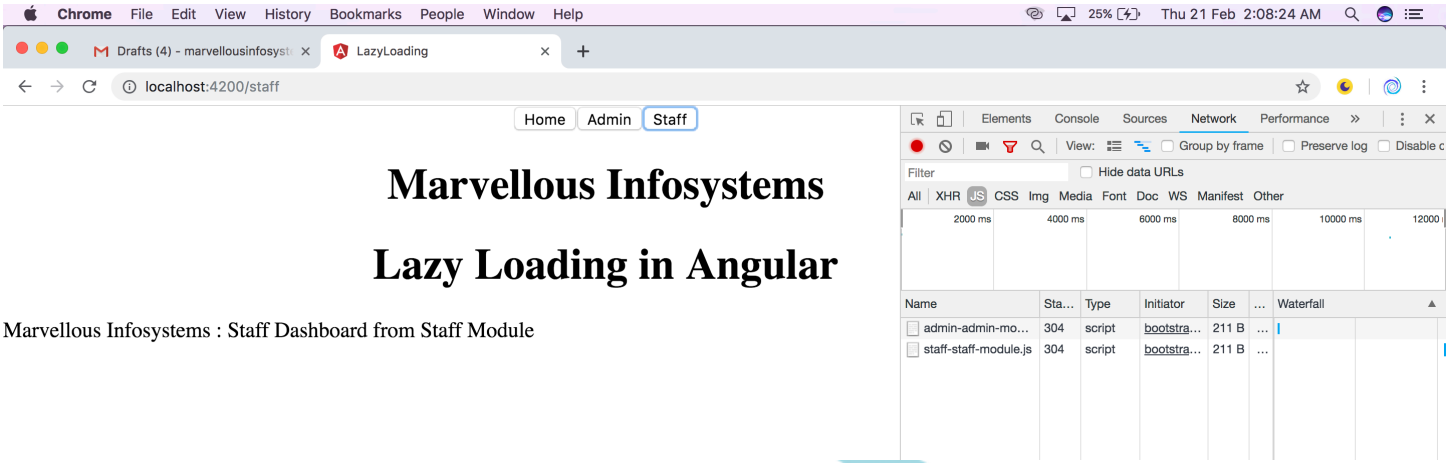
we will see all the required files are loaded.

Once we click on **admin** button we will see new network request being made to fetch admin specific chunk.



Dashboard from Admin Module

Now when we click on **staff** button we will see new network request being made to fetch staff specific chunk.



Marvellous Infosystems : Staff Dashboard from Staff Module