## Why Lazy load?

The Angular apps get bigger in size as we add more and more features. The [Angular Modules](https://www.tektutorialshub.com/angular/angular-modules/) help us to manage our app by creating separate modules for each new feature. But, as the app gets bigger in size, slower it loads. That is because of angular loads the entire application upfront.

The slow loading app does not leave a good impression on the user. By Loading only a part of the app (i.e lazy loading), the app appears to run faster to the user. The faster loading app gives you a performance boost and also results in a good user experience.

# Lazy loading in Angular

Lazy loading is the technique where angular loads the [Modules](https://www.tektutorialshub.com/angular/angular-modules/) only on a need basis rather than all at once. It is also called on-demand loading. By default, [Angular](https://www.tektutorialshub.com/angular-tutorial/)Loads the[modules](https://www.tektutorialshub.com/angular/angular-modules/) eagerly. Lazy Loading of [Angular Modules](https://www.tektutorialshub.com/angular/angular-modules/) reduces the initial load time of the app. We use the loadChilden method of the [Angular Router](https://www.tektutorialshub.com/angular/angular-routing-navigation/) to lazy load them when the user navigates to a route.

## How Lazy loading works

In Angular, the Lazy loading works at the module level. i.e. you can lazy load only the Angular Modules. We cannot lazy load the Individual components.

The Lazy loading works via the [Angular Router Module](https://www.tektutorialshub.com/angular/angular-routing-between-modules/). The loadChildren method of the Angular Router is responsible to load the Modules

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

loadChildren is where we configure the Lazy Loading.

### loadChildren

We need to provide call back function to loadChildren argument. The call back must load the AdminModule. We use the dynamic import syntax using the import method. The import method loads the module from the path, which we provide as the argument to it.

|  |  |
| --- | --- |
| 1  2  3 | import('./admin/admin.module').then(m => m.AdminModule) |

When the user navigates to the admin URL or to any of its child routes like admin/dashboard, the router will fetch the AdminModule and loads the routes and components of the AdminModule

The lazy loaded module loads only for the first visit of the URL, it will not load when we revisit that URL again.

When we define an AdminModule to lazy loaded, the angular creates a separate bundle for the entire module.

App-routing:

const routes: Routes = [

  {

    path: '',

    redirectTo: 'home',

    pathMatch: 'full'

  },

  {

    path: 'tabs',

    loadChildren: () => import('./pages/tabs/tabs.module').then(m => m.TabsPageModule)

  },

  {

    path: 'home',

    loadChildren: () => import('./pages/home/home.module').then( m => m.HomePageModule)

  },

}

**Tabs-module-routing:**

const routes: Routes = [

   {

    path: 'tabs',

    component: TabsPage,

    children: [

      {

        path: 'dashboard',

        children: [

          {

            path: '',

            loadChildren: () => import('../dashboard/dashboard.module').then(m => m.DashboardPageModule)

          },

          {

            path: 'dios',

            loadChildren: () => import('../dios/dios.module').then( m => m.DiosPageModule)

          }

        ]

      },

      {

        path: 'vehicle',

        children: [

          {

            path: '',

            loadChildren: () => import('../vehicle/vehicle.module').then(m => m.VehiclePageModule)

          },

          {

            path: 'vehicle-filter',

            loadChildren: () => import('../vehicle-filter/vehicle-filter.module').then(m => m.VehicleFilterPageModule)

          },

          {

            path: 'vehicle-detail',

            loadChildren: () => import('../vehicle-details/vehicle-details.module').then(m => m.VehicleDetailsPageModule)

          },

          {

            path: 'test-drive',

            loadChildren: () => import('../test-drive/test-drive.module').then( m => m.TestDrivePageModule)

          },

          {

            path: 'dios',

            loadChildren: () => import('../dios/dios.module').then( m => m.DiosPageModule)

          }

        ]

      },

}

Vehicle-routing.module.ts:

const routes: Routes = [

  {

    path: '',

    component: VehiclePage

  }

];

# Angular Preloading Strategy

The Angular loads all the modules, when the user requests for the first time. This will make app loading slowly as it need to download all the modules. We can solve this problem by [lazy loading](https://www.tektutorialshub.com/angular/angular-lazy-loading/) those modules. The Angular allows us further optimize our app using a technique called **PreLoading**.

## What is Angular Preloading Strategy ?

Preloading in Angular means loading the Lazy loaded Modules in the background asynchronously, while user is interacting with the app. This will help boost up the loading time of the app

The Angular apps are modular and allows us build apps in chunks of modules. We can load these modules lazily, when the user navigates to a route. We need to mark the modules to be lazy loaded using the loadChildren property of the router.

By Lazy loading the modules, we can reduce the initial download size of the app, and thus making app load quickly. This is very useful in case of big apps. But when user navigates to a lazy loaded part of the app, the angular will have to download the module from the server, which means that user will have to wait for the download to finish.

By Preloading the lazy loaded module, the user do not have to wait for the module to be downloaded as the module is already downloaded in the background.

## How to Enable Preloading

To make use of Preloading, first we need to enable lazy loading of the Modules. Mark the modules with the loadChildren, when you define routes



## Preloading Strategies

The Angular provides two built in strategies out of the box. one is PreloadAllModules and other one is NoPreloading

### NoPreloading

This will disables all the preloading. This is default behavior i.e. if you don not specify the preloadingStrategy, then the angular assumes you do not want preloading

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | RouterModule.forRoot(routes,     {        preloadingStrategy: NoPreloading     } |

### PreloadAllModules

This strategy will preload all the lazy loaded modules.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | RouterModule.forRoot(routes,     {        preloadingStrategy: PreloadAllModules     }) |

## Custom preloading strategy

With PreloadAllModules all the modules are preloaded, which may actually create a bottleneck if the application has large no of modules to be loaded.

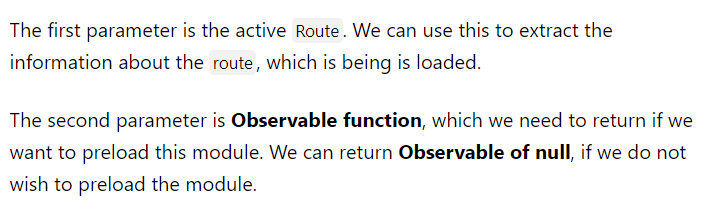
The better way strategy would be

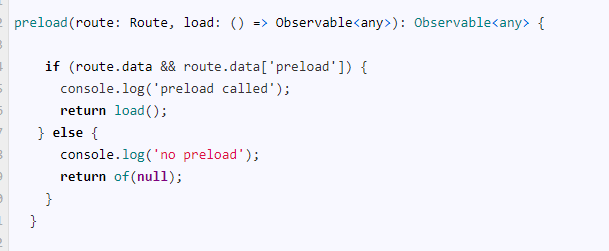
1. Eagerly Load the modules required at startup. For Example authentication module, core module, shared module etc
2. Preload all frequently used modules, may be after some delay
3. Lazy load remaining modules

First create a class, which implements the built in PreloadingStrategy class

The class must implement the method preload(). In this method, we determine whether to preload the module or not. The method signature is as follows







Example:pending