



# Angular Advanced Module - Lazy loading



Peter Kassenaar  
[info@kassenaar.com](mailto:info@kassenaar.com)

# What is lazy loading

- **Deferred loading** of modules, until the user needs them.
  - Modules are loaded once the user navigates to them.
- OR: for optimal user experience:
  - Load the **minimum setup** for the application to work, so the user can interact with the app.
  - Then asynchronously load other modules.
  - They are instantly available if the user navigates to them
- Only *modules* can be loaded lazily, *not components*.
- Lazy loading works in conjunction with the **router**.
- It is considered **best practice** nowadays to use LL from the start

# Victor Savkin – creator of the router



Sign in / Sign up



Victor Savkin

Co-founder of Narwhal Technologies (nrwl.io), where we provide Angular consulting to large teams who wa...

Oct 12, 2016 · 3 min read

## Angular Router: Preloading Modules



ANGULAR  
ROUTER



*Victor Savkin is a co-founder of [nrwl.io](https://nrwl.io), providing Angular consulting to enterprise teams. He was previously on the Angular core team at Google, and built the dependency injection, change detection, forms, and router modules.*

 Never miss a story from **Angular**, when you sign up for Medium.  
Learn more

GET UPDATES

<https://vsavkin.com/angular-router-preloading-modules-ba3c75e424cb>

# Official documentation

The screenshot shows the Angular official documentation website. The top navigation bar is blue with the Angular logo and links for FEATURES, DOCS, RESOURCES, EVENTS, and BLOG. A search bar and social media icons are on the right. The left sidebar contains a table of contents with 'Best Practices' expanded, showing links to Security, Accessibility, Keeping Up-to-Date, Property Binding Best Practices, Lazy Loading Feature Modules (selected), Lightweight Injection Tokens for Libraries, Angular Tools, Tutorials, Release Information, and Reference. Below the sidebar is a version selector showing 'stable (v13.1.2)'. The main content area is titled 'Lazy-loading feature modules' and includes a paragraph explaining that NgModules are eagerly loaded by default and how lazy loading helps with bundle sizes. A light blue callout box contains a link to a 'live example / download example'. Below this is the 'Lazy loading basics' section, which introduces the basic procedure for configuring a lazy-loaded route and provides a code snippet for the AppRoutingModule. The code snippet shows a route configuration for 'items' that uses 'loadChildren' to import './items/items.module' and then the 'ItemsModule'. The right sidebar lists 'Lazy-loading feature modules' with a list of sub-topics including Lazy loading basics, Step-by-step setup, Set up an app, Create a feature module with routing, Add another feature module, Set up the UI, Imports and route configuration, Inside the feature module, Verify lazy loading, forRoot() and forChild(), Preloading, Preloading modules, Preloading component data, Troubleshooting lazy-loading modules, and More on NgModules and routing.

Introduction

Getting Started >

Understanding Angular >

Developer guides >

Best Practices >

Security

Accessibility

Keeping Up-to-Date

Property Binding Best Practices

Lazy Loading Feature Modules

Lightweight Injection Tokens for Libraries

Angular Tools >

Tutorials >

Release Information >

Reference >

stable (v13.1.2)

## Lazy-loading feature modules

By default, NgModules are eagerly loaded, which means that as soon as the application loads, so do all the NgModules, whether or not they are immediately necessary. For large applications with lots of routes, consider lazy loading—a design pattern that loads NgModules as needed. Lazy loading helps keep initial bundle sizes smaller, which in turn helps decrease load times.

For the final sample application with two lazy-loaded modules that this page describes, see the [live example / download example](#).

### Lazy loading basics

This section introduces the basic procedure for configuring a lazy-loaded route. For a step-by-step example, see the [step-by-step setup](#) section on this page.

To lazy load Angular modules, use `loadChildren` (instead of `component`) in your `AppRoutingModule` `routes` configuration as follows.

```
AppRoutingModule (excerpt)

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

In the lazy-loaded module's routing module, add a route for the component.

Routing module for lazy loaded module (excerpt)

#### Lazy-loading feature modules

- Lazy loading basics
- Step-by-step setup
  - Set up an app
  - Create a feature module with routing
  - Add another feature module
  - Set up the UI
  - Imports and route configuration
  - Inside the feature module
  - Verify lazy loading
  - `forRoot()` and `forChild()`
- Preloading
  - Preloading modules
  - Preloading component data
- Troubleshooting lazy-loading modules
- More on NgModules and routing

<https://angular.io/guide/lazy-loading-ngmodules>

# How to lazy load : Angular 4.x – 7.x

Add or edit `app-routing.module.ts`

- Don't point directly to components
- Point to Modules instead. Use `loadChildren()`
- Note the (ugly) stringnotation

```
const routes: Routes = [  
  {path: '', redirectTo: 'customers', pathMatch: 'full'},  
  {path: 'customers', loadChildren: './customer/customer.module#CustomerModule'},  
  {path: 'products', loadChildren: './products/products.module#ProductsModule'},  
];  
  
export const AppRoutingModule = RouterModule.forRoot(routes);
```

# How to lazy load : Angular 8.x+

Add or edit `app-routing.module.ts`


- Use the browser `import()` statement
- More in line with other frameworks
- More Typesafe.

```
const routes: Routes = [  
  {path: '', redirectTo: 'customers', pathMatch: 'full'},  
  // New notation (Angular 8+):  
  {  
    path: 'customers',  
    loadChildren: () => import('./customer/customer.module')  
      .then(mod => mod.CustomerModule)  
  },  
  {  
    path: 'products',  
    loadChildren: () => import('./products/products.module')  
      .then(mod => mod.ProductsModule)  
  },  
];  
export const AppRoutingModule = RouterModule.forRoot(routes);
```

Edit `app.module.ts` (no more loading of modules)

```
// import routing module that defines the LL
import {AppRoutingModule} from './app.routing.module';

@NgModule({
  ...
  imports      : [
    BrowserModule,
    AppRoutingModule
  ],
  bootstrap    : [AppComponent]
})
export class AppModule {
}
```



Edit separate modules,  
add `RouterModule.forChild()` with various components.

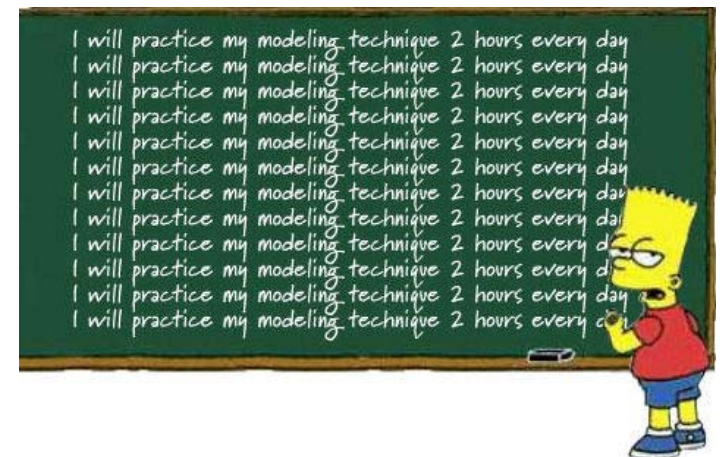
```
import {RouterModule, Routes} from '@angular/router';  
  
// lazy loaded routes for this module  
const customerRoutes: Routes = [  
  {path: '', component: CustomerComponent}  
];  
  
@NgModule({  
  imports : [  
    ...  
    RouterModule.forChild(customerRoutes)  
  ],  
  ...  
})  
export class CustomerModule {  
}  
console.log('CustomerModule loaded lazily...');
```





# Workshop

- Open `../110-lazy-loading`.
  - Create a new module
  - Create a new component inside this new module and give it some UI.
  - Add a route to the new component
  - Use the new module in the root module and lazy load it
  - Add a link to navigate to the lazy loaded module.
- 
- *OR:*
  - Add LL from scratch to your own application, using the steps described in this module.
  - Add a new (dynamic) child route to Module





# Preloading strategies

# Preloading Strategies

- Optimize Lazy Loading even further: preloading strategies
  - Load all modules in background
  - Load only modules *you want to load* in the background
- Default preloading: `PreloadAllModules`

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

```
const config: ExtraOptions = {  
  preloadingStrategy: PreloadAllModules  
};
```

```
export const AppRoutingModule = RouterModule.forRoot(routes, config);
```

The image shows two overlapping Chrome DevTools windows. The top window displays the Console tab with three messages: 'Angular is running in the development mode. Call enableProdMode() to enable the production mode.' (core.es5.js:2925), 'CustomerModule loaded lazily...' (customer.module.ts:26), and 'ProductsModule loaded lazily...' (products.module.ts:26). The bottom window displays the Network tab with a search filter 'ch'. It shows a list of network requests with columns for Name, Status, Type, Initiator, Size, Time, and Waterfall. Three red arrows point to the first three rows of the table: '2.chunk.js', '0.chunk.js', and '1.chunk.js'. The 'Waterfall' column shows green bars for these requests, indicating successful completion.

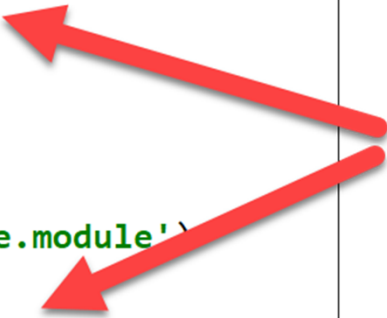
Name	Sta...	Type	Initiator	Size	Time	Waterfall
2.chunk.js	200	scri...	bootstra...	5.4 ...	94 ...	
0.chunk.js	200	scri...	bootstra...	7.1 ...	93 ...	
ng-validate.js	200	scri...	content...	(fro...	2 ms	
1.chunk.js	200	scri...	bootstra...	5.3 ...	6 ms	
backend.js	200	scri...	content...	(fro...	54 ...	

<https://angular.io/api/router/PreloadAllModules>

# Custom preloading strategy

- Define which module(s) are loaded lazily, while others are loaded on demand
- Solution: compose a strategy that *only* preloads routes when a custom `data.preload` flag is set to `true`

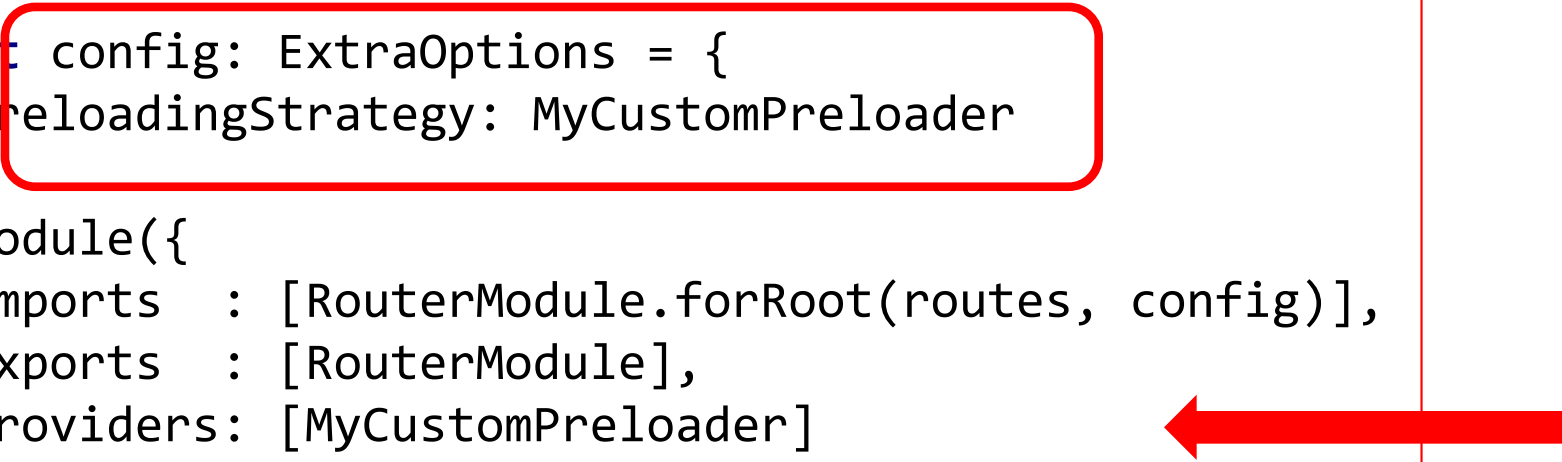
```
path: 'products',
loadChildren: () => import('./products/products.module')
  .then(module => module.ProductsModule),
data: {preload: true} // preload flag
},
{
  path: 'big-module',
  loadChildren: () => import('./very-big-module/very-big-module.module')
    .then(module => module.VeryBigModule)
  // Note: NO flag for preloading
},
```



# Steps

1. Create new module, with a (potential) heavy load
2. Add data property and set { preload:true } to every route you want to load lazily
3. Assign custom preloader to preloadingStrategy:

```
...  
const config: ExtraOptions = {  
  preloadingStrategy: MyCustomPreloader  
};  
@NgModule({  
  imports : [RouterModule.forRoot(routes, config)],  
  exports : [RouterModule],  
  providers: [MyCustomPreloader]  
})  
export class AppRoutingModule {  
}
```



# Define custom loader

```
// app.routing.loader.ts
import { PreloadingStrategy, Route } from '@angular/router';

import { Observable, of } from 'rxjs';

export class MyCustomPreloader implements PreloadingStrategy {
  preload(route: Route, load: Function): Observable<any> {
    // only preload the route if data attribute is set and preload===true
    return route.data && route.data.preload ? load() : of(null);
  }
}
```

# Run the app

Run the app. The first 2 modules should be loaded lazily, the third module should be loaded on demand

Multiple Modules x

localhost:4200/big-module

## Example application with Lazy Loading modules

CUSTOMERS PRODUCTS VERY BIG MODULE

Big Module

Some very big list

- Item 0
- Item 1
- Item 2

Console



- Angular is running in the development mode. Call enableProdMode() to enable the production mode. [core.es5.js:2925](#)
- CustomerModule loaded lazily... [customer.module.ts:26](#)
- ProductsModule loaded lazily... [products.module.ts:26](#)
- Very big module, loaded on demand [very-big-module.module.ts:20](#)

Example: ../120-custom-preloading



# More information

[TRAINING](#) [WORK](#) [BLUEPRINT](#)



My name is [Cory Rylan](#), Senior Front End Developer at [Vintage Software](#) and [Angular Boot Camp](#) instructor. I specialize in creating fast, scalable, and responsive web applications.

[Follow @coryryan](#)

## Custom Preloading and Lazy Loading Strategies with Angular

Cory Rylan  
Apr 30, 2017 - 5 min read

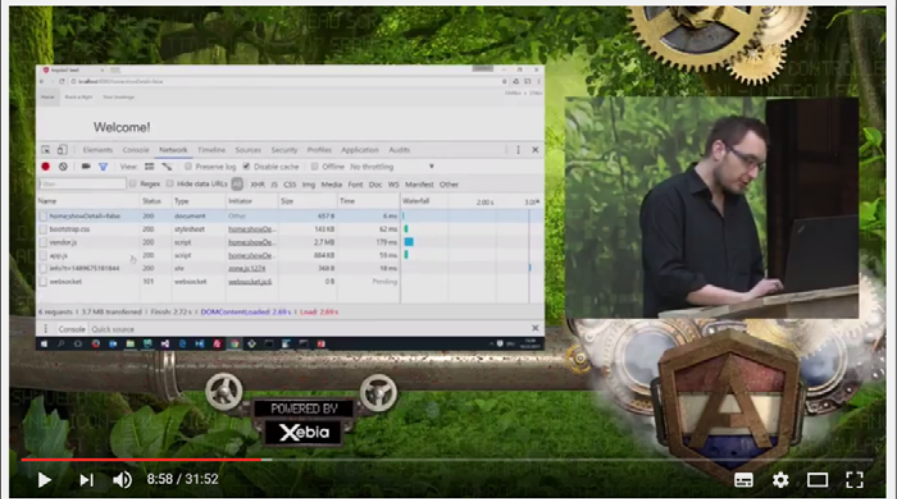
[angular](#) [performance](#)

This article is for versions of Angular 2, Angular 4 and later.

Angular has many features that allow us to configure apps to be as fast and high performing as possible. One of the critical features that enable responsive fast Angular apps is the ability to lazy load code with the Angular Router. This allows our initial bundles to remain small ensuring faster download and start-up times for our users.

<https://coryryan.com/blog/custom-preloading-and-lazy-loading-strategies-with-angular>

## Manfred Steyer - Improving Startup Performance with Lazy Loading in Angular



NG-NL 2017: Manfred Steyer - Improving Startup Performance with Lazy Loading in Angular

NG-NL

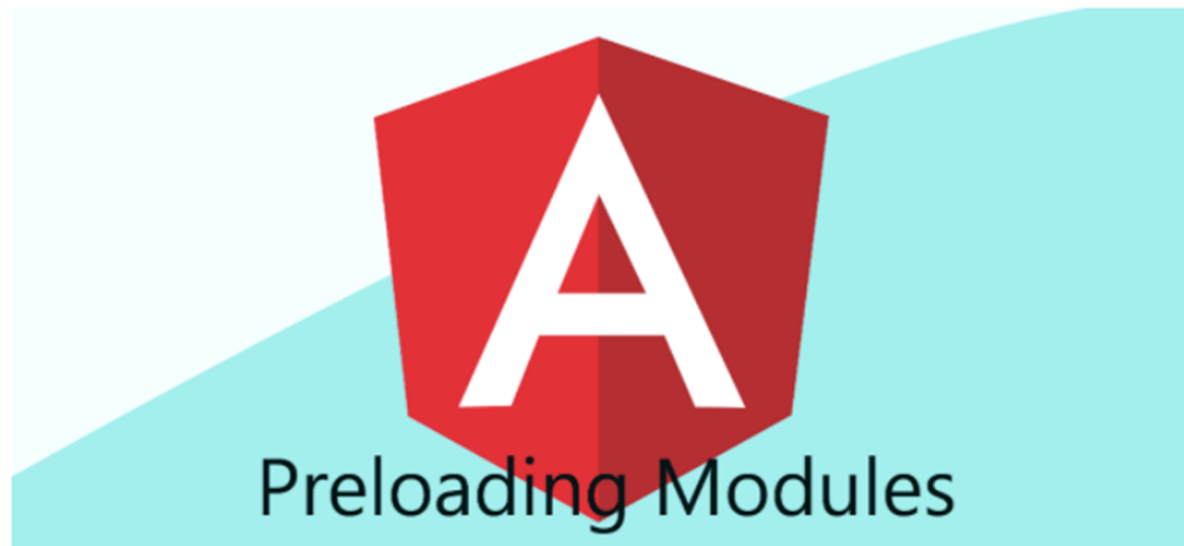
<https://www.youtube.com/watch?v=n6EMOeCDfjc>

# Angular | custom preloading strategy



Muthu Devendra [Follow](#)

Feb 3, 2019 · 3 min read

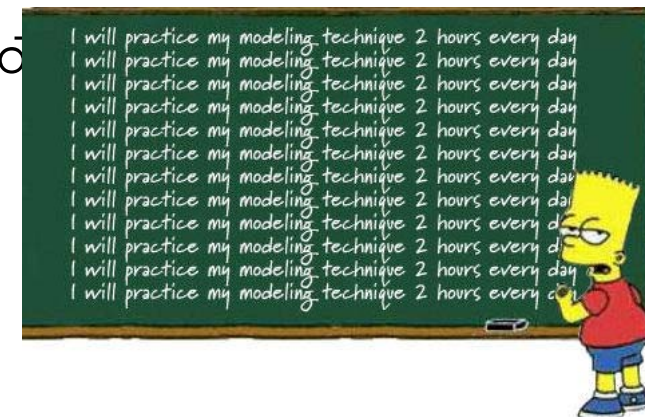


In my previous post I talked about three angular module loading types which are eager loading, lazy loading and preloading strategy. if you haven't read it already you can find it in [this](#) link. Today I will talk about preloading strategy and how to write your own preloading strategy.

<https://medium.com/@muthudevendra/angular-custom-preloading-strategy-32abe99944f8>

# Workshop

- Add a new module w/ component to your application.
- Add the module to the routing section of your application. Add a link to navigate to the route.
  - Use the `PreloadAllModules` strategy
- Let *other* modules be loaded lazily by adding a `data` property
- Write a custom preloading service class, e.g.  
`preload.service.ts`
- Add the custom preloader to `app.routing.module`
  - Note: make sure this is (now) actually a Module, as it has to import and provide `app.preloader.ts`
- Example: `../120-custom-preloading`



# Naming your JavaScript-chunks on code splitting

1. `/* webpackChunkName: "<your-name>" */`

```
const routes: Routes = [
  {path: '', redirectTo: 'customers', pathMatch: 'full'},
  {
    path: 'customers',
    loadChildren: () => import(/* webpackChunkName: "Customers" */
      './customer/customer.module')
      .then(mod => mod.CustomerModule)
  },
  {
    path: 'products',
    loadChildren: () => import(/* webpackChunkName: "Products" */
      './products/products.module')
      .then(mod => mod.ProductsModule)
  },
];
export const AppRoutingModule = RouterModule.forRoot(routes);
```

Creates `Customers.js`, `Products.js`, and so on