The background is a solid teal color with various white line drawings and sketches overlaid. These include architectural elements like a brick wall, a staircase, and a building facade. There are also mechanical or technical drawings, such as a circular component with concentric rings and a ladder-like structure. A small crescent moon is visible in the upper right corner.

# Reactive Angular met RxJS Routing & 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.io website with the 'Lazy Loading Feature Modules' page. The header is blue with the Angular logo and navigation links: FEATURES, DOCS, RESOURCES, EVENTS, and BLOG. A search bar contains the word 'lazy'. The left sidebar lists the site's structure, with 'FUNDAMENTALS' and 'NgModules' expanded. The main content area has a title 'Lazy Loading Feature Modules' with an edit icon. Below the title is a 'Prerequisites' section stating a basic understanding of Feature Modules, JavaScript Modules vs. NgModules, Frequently Used Modules, Types of Feature Modules, and Routing and Navigation. It includes links to a 'live example' and a 'download example'. The 'High level view' section lists three steps: 1. Create the feature module, 2. Create the feature module's routing module, and 3. Configure the routes. A right sidebar shows a table of contents for the 'Lazy Loading Feature Modules' section, including links to 'High level view', 'Set up an app', 'Create a feature module with routing', 'Add a component to the feature module', 'Add another feature module', 'Set up the UI', 'Configure the routes' (with sub-links for app and feature module levels), 'Confirm it's working', and 'More on NgModules and routing'.

ANGULAR FEATURES DOCS RESOURCES EVENTS BLOG

lazy

GETTING STARTED

TUTORIAL >

FUNDAMENTALS >

Architecture >

Components & Templates >

Forms >

Observables & RxJS >

Bootstrapping

NgModules >

NgModules Introduction

JS Modules vs NgModules

Frequently Used NgModules

Types of Feature Modules

Entry Components

Feature Modules

Providers

## Lazy Loading Feature Modules

### Prerequisites

A basic understanding of the following:

- [Feature Modules](#).
- [JavaScript Modules vs. NgModules](#).
- [Frequently Used Modules](#).
- [Types of Feature Modules](#).
- [Routing and Navigation](#).

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

### High level view

There are three main steps to setting up a lazy loaded feature module:

1. Create the feature module.
2. Create the feature module's routing module.
3. Configure the routes.

#### Lazy Loading Feature Modules

- High level view
- Set up an app
- Create a feature module with routing
- Add a component to the feature module
- Add another feature module
- Set up the UI
- Configure the routes
  - Routes at the app level
  - Inside the feature module
  - Configure the feature module's routes
- Confirm it's working
- `forRoot()` and `forChild()`
- 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);
```

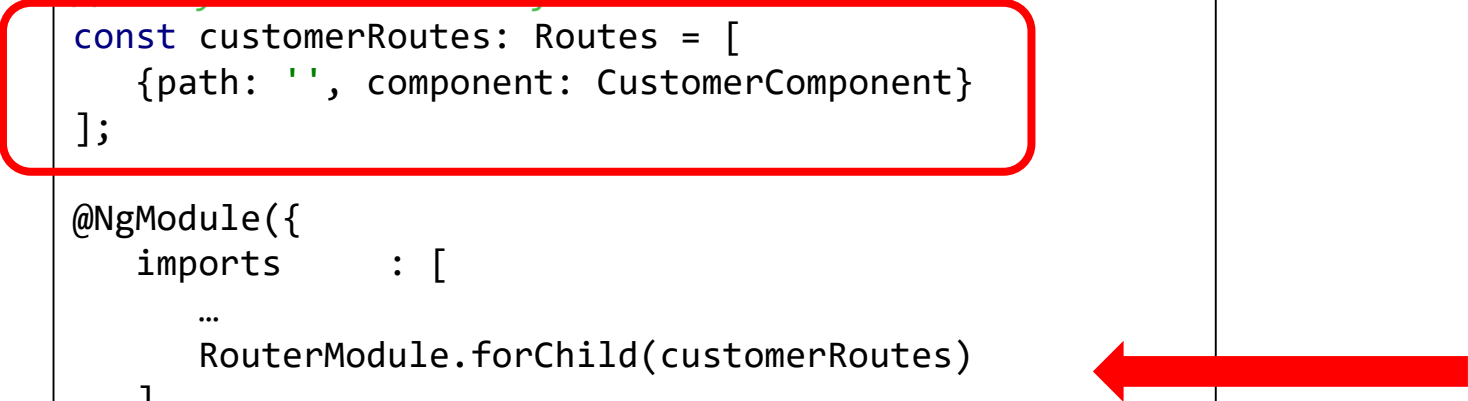
## Edit `app.module.ts` (no more loading of modules)

```
// 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.

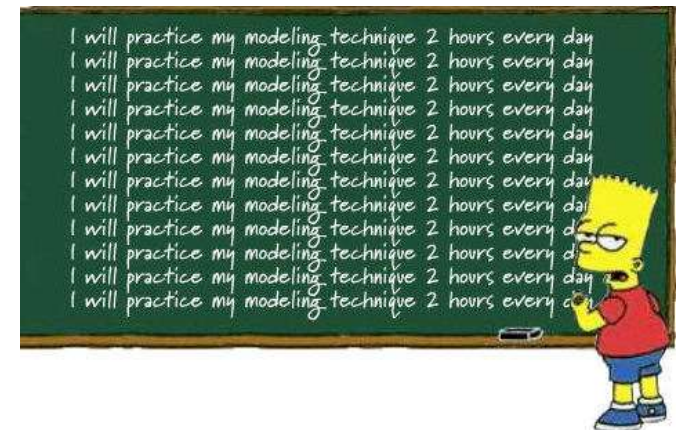
```
import {RouterModule, Routes} from '@angular/router';  
  
// lazy loaded routes for this module  
const customerRoutes: Routes = [  
  {path: '', component: CustomerComponent}  
];  
  
@NgModule({  
  imports      : [  
    ...  
    RouterModule.forChild(customerRoutes)  
  ],  
  ...  
})  
export 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 panels from the Chrome DevTools interface. The top panel is the Console, and the bottom panel is the Network tab.

**Console Panel:**

- 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`)

**Network Panel:**

Search:  | ☐ Regex | ☐ Hide data URLs

Filter: All | XHR | JS | CSS | Img | Media | Font | Doc | WS | Manifest | Other

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 ...		

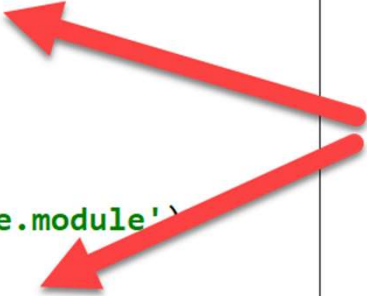
Three red arrows point to the rows for `2.chunk.js`, `0.chunk.js`, and `1.chunk.js` in the Network panel.

<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`


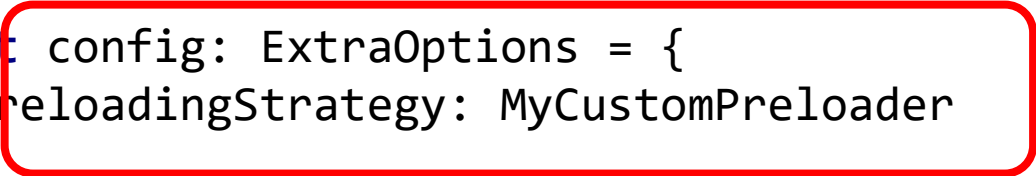
```
path: 'products',
loadChildren: () => import('./products/products.module')
  .then(module => module.ProductsModule),
data: {preload: true} // preload flag
},
{
  path: 'big-module',
  loadChildren: () => import('./very-big-module/very-big-module.module')
    .then(module => module.VeryBigModule)
  // 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

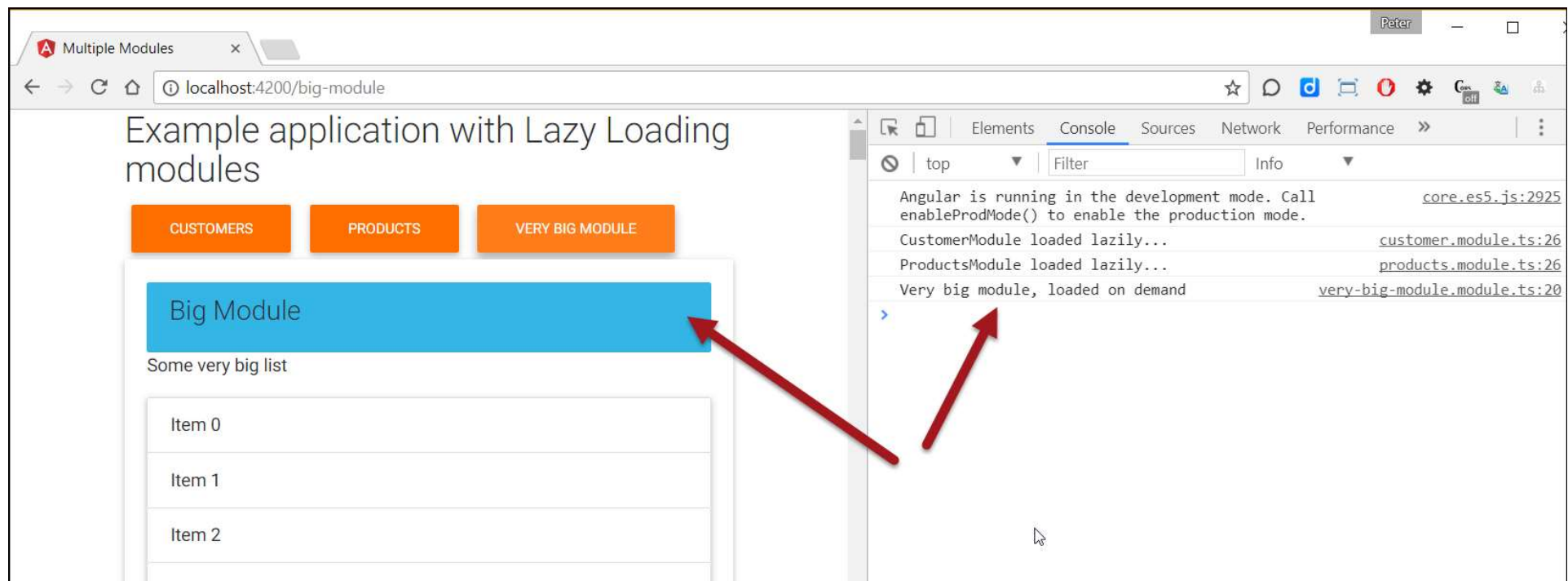
```
// app.routing.loader.ts
import { PreloadingStrategy, Route } from '@angular/router';

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



The screenshot shows a web browser window with the URL `localhost:4200/big-module`. The page title is "Example application with Lazy Loading modules". It features three orange buttons: "CUSTOMERS", "PRODUCTS", and "VERY BIG MODULE". Below these buttons is a blue box labeled "Big Module" with the text "Some very big list" underneath it. A list of items is visible below the blue box, including "Item 0", "Item 1", and "Item 2". The Chrome DevTools console is open on the right side of the browser window. The console shows the following messages:

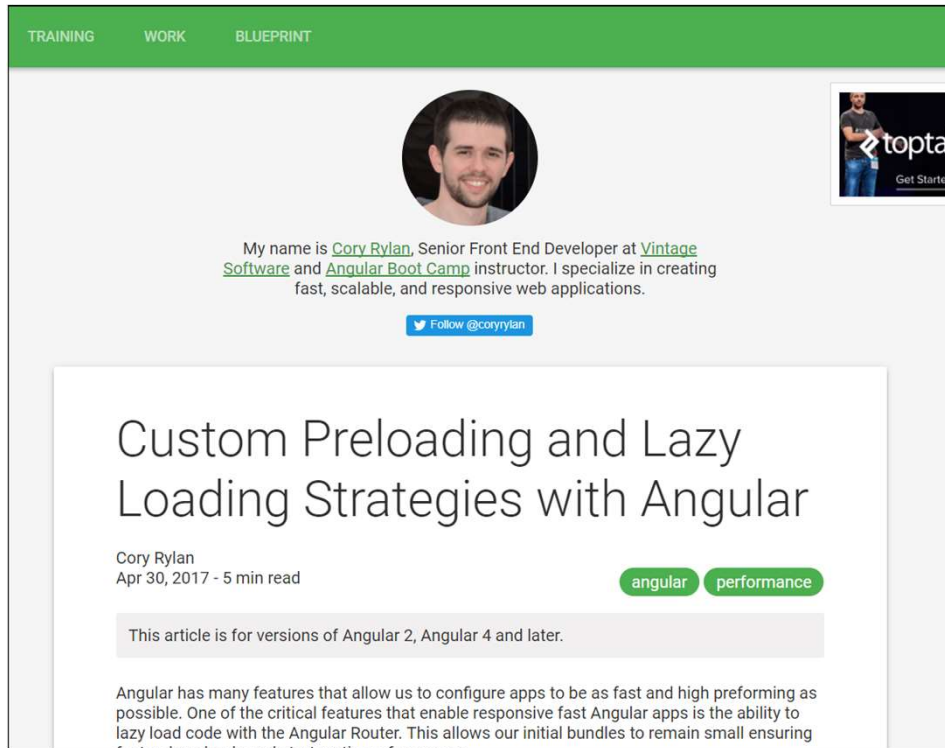
- Angular is running in the development mode. Call `enableProdMode()` to enable the production mode. (`core.es5.js:2925`)
- CustomerModule loaded lazily... (`customer.module.ts:26`)
- ProductsModule loaded lazily... (`products.module.ts:26`)
- Very big module, loaded on demand (`very-big-module.module.ts:20`)

Two red arrows point from the console messages to the application UI. One arrow points from the "Very big module, loaded on demand" message to the "Big Module" blue box. The other arrow points from the "CustomerModule loaded lazily..." message to the "CUSTOMERS" button.

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

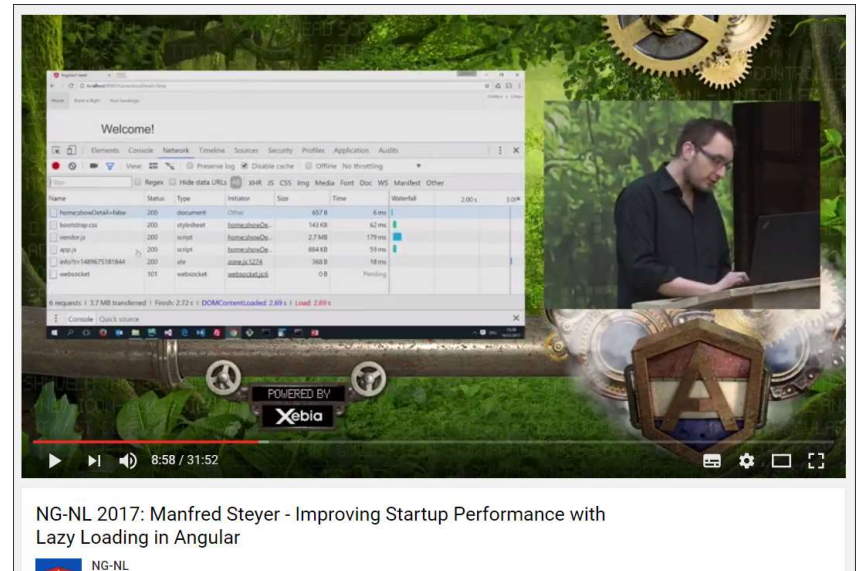
angular performance

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 fast downloads and startup times for users.

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

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



Welcome!

Name	Status	Type	Vendor	Size	Time	Bundle	Manifest	Other
homebrewDetail-fiber	200	document	Chrome	657 B	6 ms			
bootstrap.css	200	stylesheet	homebrewDe...	143 KB	62 ms			
vendor.js	200	script	homebrewDe...	2.7 MB	179 ms			
app.js	200	script	homebrewDe...	864 KB	59 ms			
index148675181844	200	svg	angular.1229	164 B	18 ms			
websocket	501	websocket	websocket.js	0 B	0 ms	Pending		

6 requests | 3.7 MB transferred | Finish: 2.72 s | DOMContentLoaded: 2.69 s | Load: 2.69 s

POWERED BY Xebia

8:58 / 31:52

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.
- 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.ts`.
  - Note: make sure this is (now) actually a Module, as it has to import and provide `app.preloader.ts`
- Example: `../120-custom-preloading`

