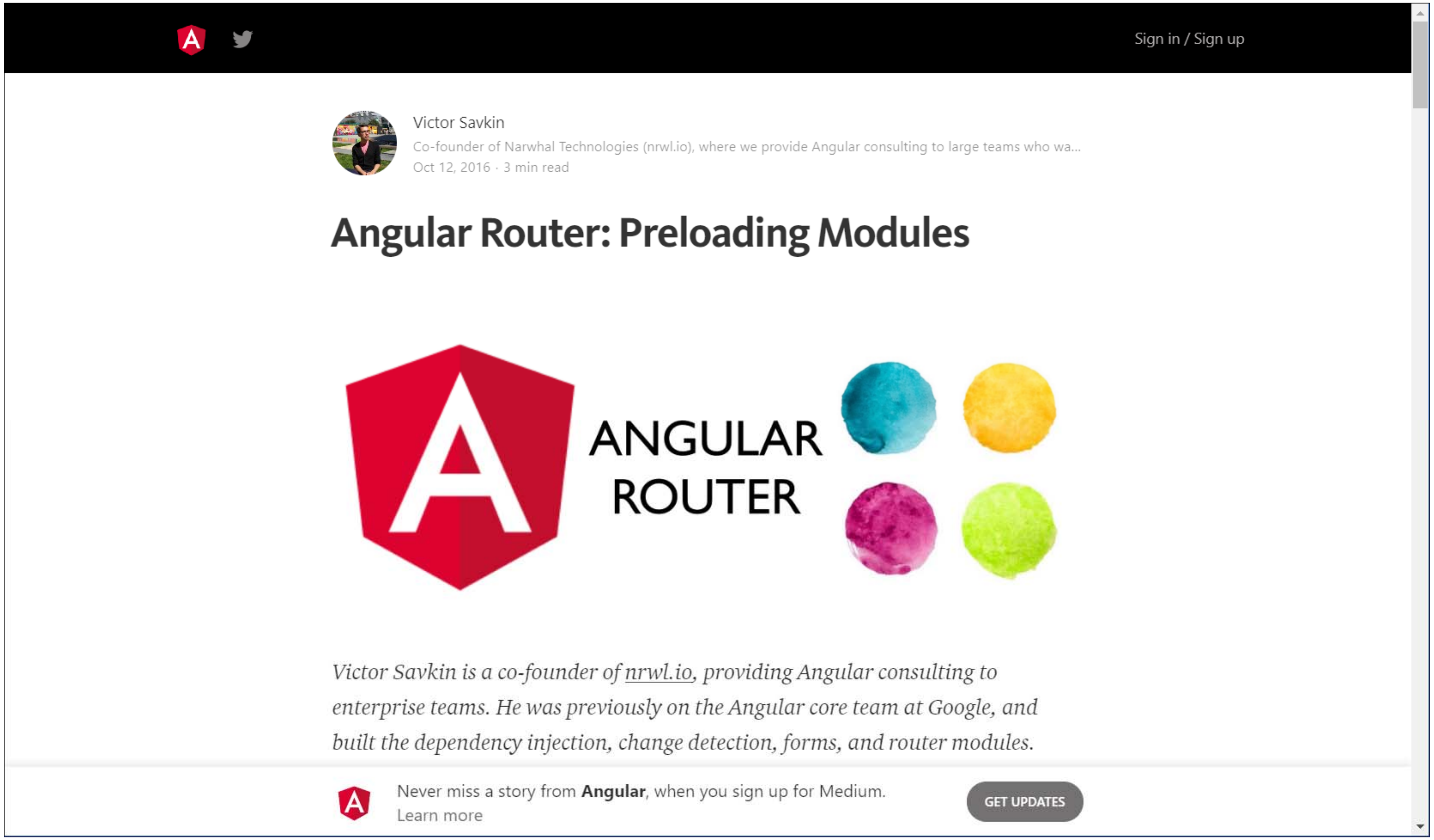
Angular Advanced
02 – Lazy loading

Peter Kassenaar –
info@kassenaar.com

What is lazy loading

- Deferred loading of modules, until the user needs them.
    - OR: for optimal user experience:
    - Load the minimum setup for the application to work, so the user can interact with the app.
    - Then asynchrounously 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

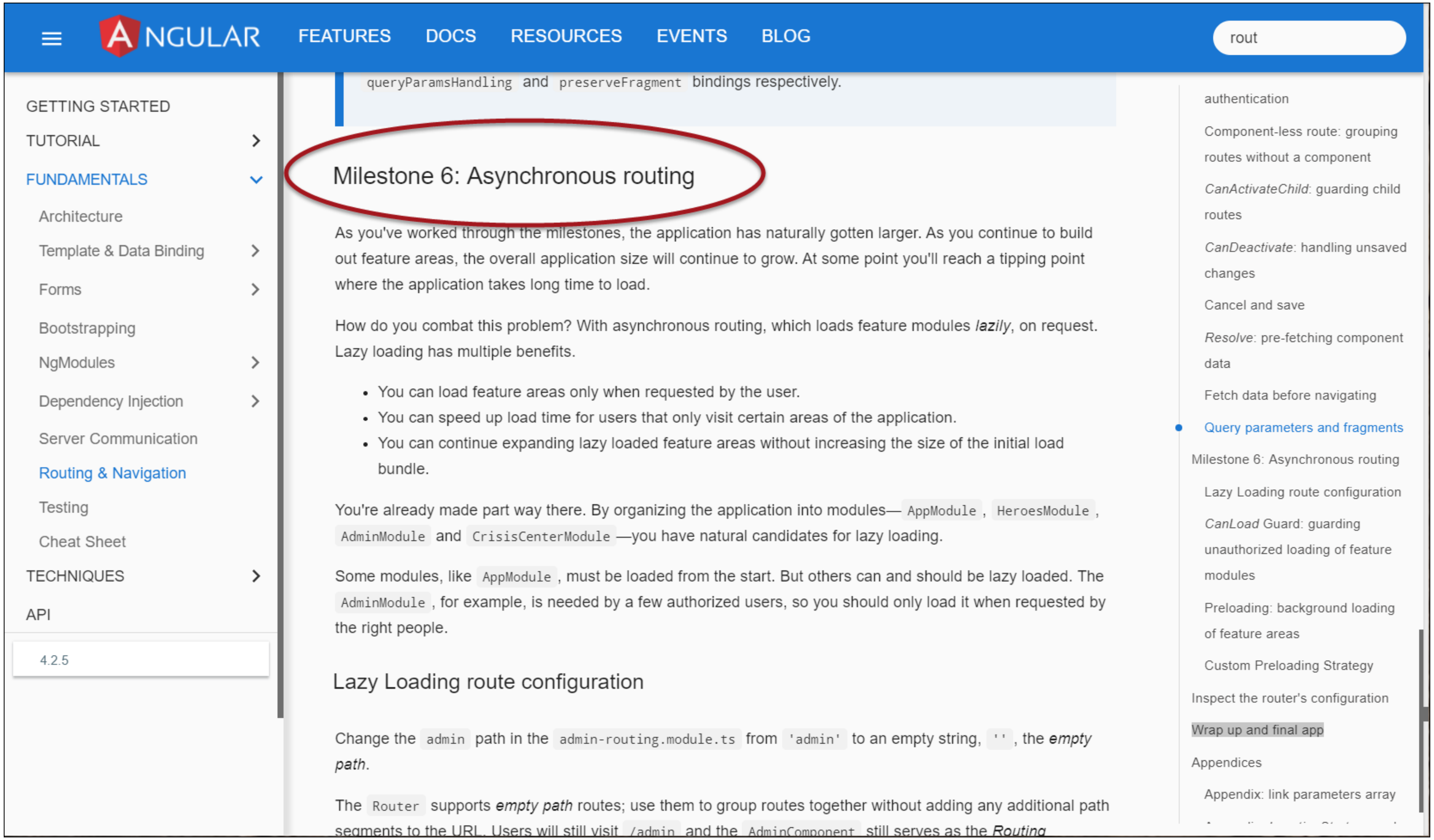
Victor Savkin is a co-founder of nrwl.io, providing Angular consulting to enterprise teams. He was previously on the Angular core team at Google, and built the dependency injection, change detection, forms, and router modules.

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


ANGULAR
FEATURES
DOCS
RESOURCES
EVENTS
BLOG
rout
GETTING STARTED
TUTORIAL
FUNDAMENTALS
Architecture
Template & Data Binding
Forms
Bootstrapping
NgModules
Dependency Injection
Server Communication
Routing & Navigation
Testing
Cheat Sheet
TECHNIQUES
API
4.2.5
authentication
Component-less route: grouping routes without a component
CanActivateChild: guarding child routes
CanDeactivate: handling unsaved changes
Cancel and save
Resolve: pre-fetching component data
Fetch data before navigating
Query parameters and fragments
Milestone 6: Asynchronous routing
Lazy Loading route configuration
CanLoad Guard: guarding unauthorized loading of feature modules
Preloading: background loading of feature areas
Custom Preloading Strategy
Inspect the router's configuration
Wrap up and final app
Appendices
Appendix: link parameters array

`queryParamsHandling` and `preserveFragment` bindings respectively.

Milestone 6: Asynchronous routing

As you've worked through the milestones, the application has naturally gotten larger. As you continue to build out feature areas, the overall application size will continue to grow. At some point you'll reach a tipping point where the application takes long time to load.

How do you combat this problem? With asynchronous routing, which loads feature modules *lazily*, on request. Lazy loading has multiple benefits.

- You can load feature areas only when requested by the user.
- You can speed up load time for users that only visit certain areas of the application.
- You can continue expanding lazy loaded feature areas without increasing the size of the initial load bundle.

You're already made part way there. By organizing the application into modules— `AppModule` , `HeroesModule` , `AdminModule` and `CrisisCenterModule` —you have natural candidates for lazy loading.

Some modules, like `AppModule` , must be loaded from the start. But others can and should be lazy loaded. The `AdminModule` , for example, is needed by a few authorized users, so you should only load it when requested by the right people.

Lazy Loading route configuration

Change the `admin` path in the `admin-routing.module.ts` from `'admin'` to an empty string, `''` , the *empty path*.

The `Router` supports *empty path* routes; use them to group routes together without adding any additional path segments to the URL. Users will still visit `/admin` and the `AdminComponent` still serves as the *Routing*

https://angular.io/guide/router#asynchronous-routing

How to lazy load

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

- Don’t point directly to components
- Point to Modules instead. Use `loadChildren()`

```
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);
```

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 {
}
```

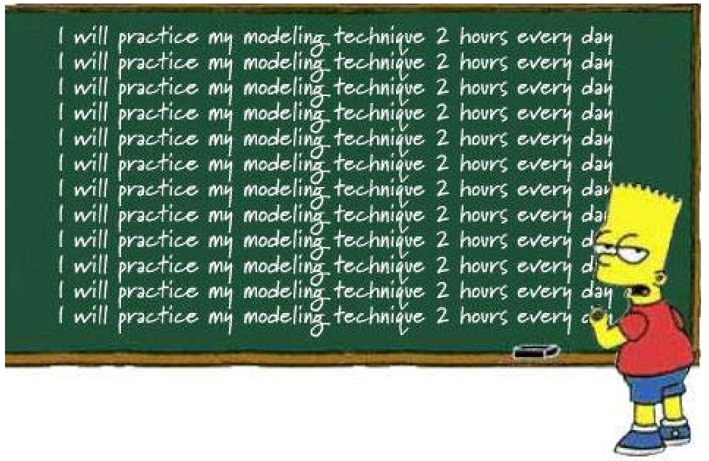
Edit separate modules, add `RouterModule.forChild()` with various components.

```
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

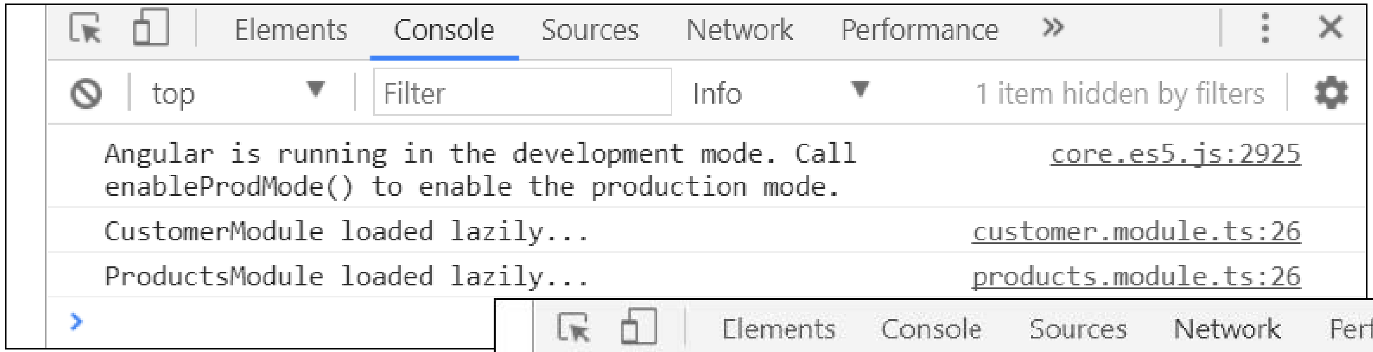


Preloading strategies

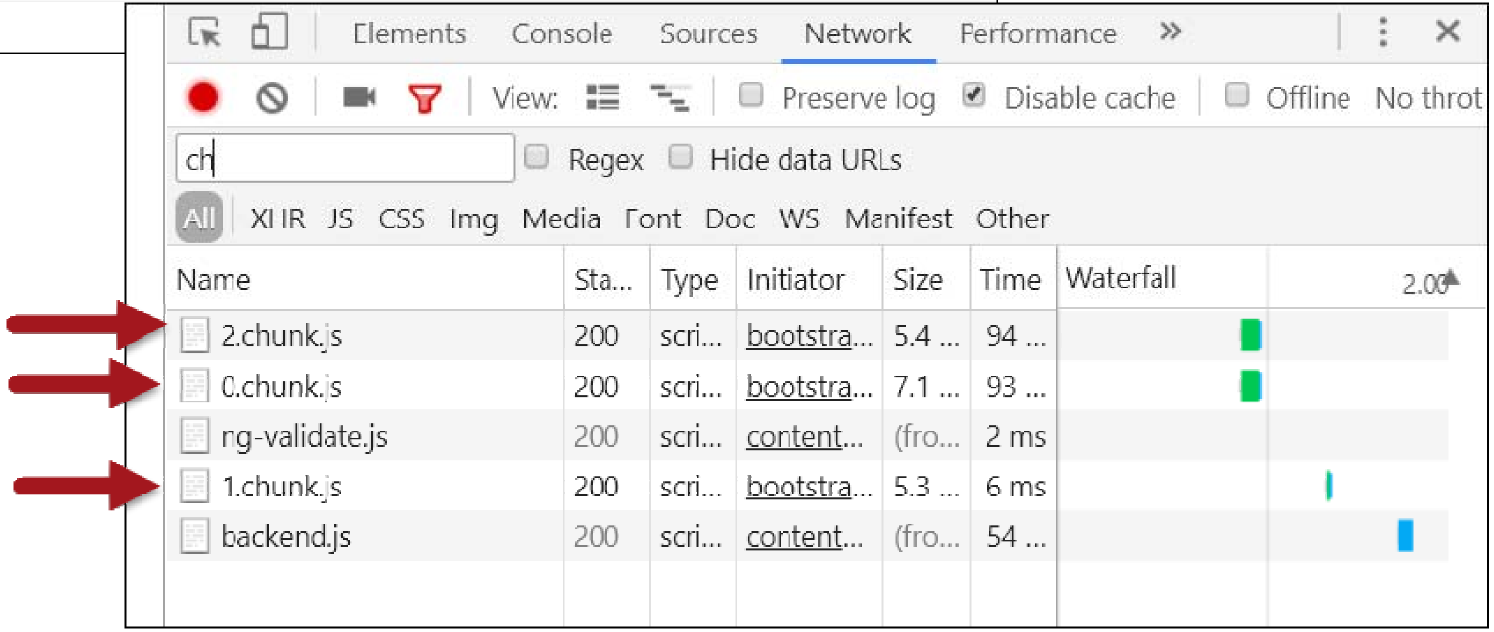
Preloading Strategies

- 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);
```



```
Angular is running in the development mode. Call          core.es5.js:2925
enableProdMode() to enable the production mode.
CustomerModule loaded lazily...                    customer.module.ts:26
ProductsModule loaded lazily...                    products.module.ts:26
```



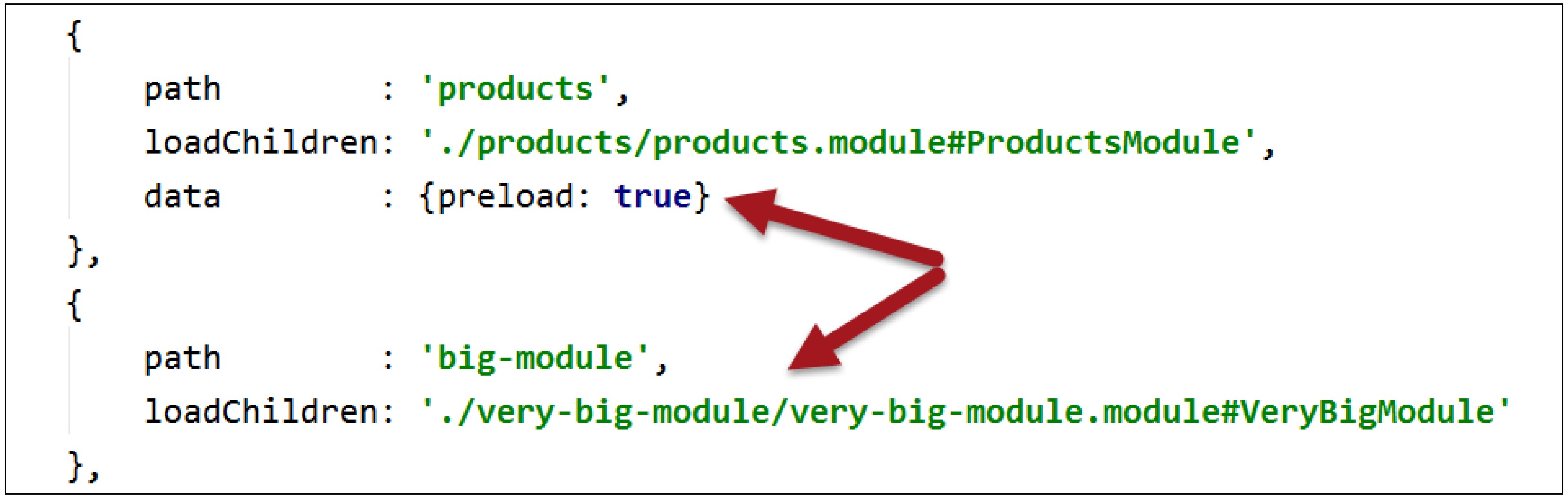
Name	Sta...	Type	Initiator	Size	Time
2.chunk.js	200	scri...	bootstra...	5.4 ...	94 ...
0.chunk.js	200	scri...	bootstra...	7.1 ...	93 ...
ng-validate.js	200	scri...	content...	(fro...	2 ms
1.chunk.js	200	scri...	bootstra...	5.3 ...	6 ms
backend.js	200	scri...	content...	(fro...	54 ...

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: './products/products.module#ProductsModule',
    data        : {preload: true}
},
{
    path        : 'big-module',
    loadChildren: './very-big-module/very-big-module.module#VeryBigModule'
},
```

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 {
}
```

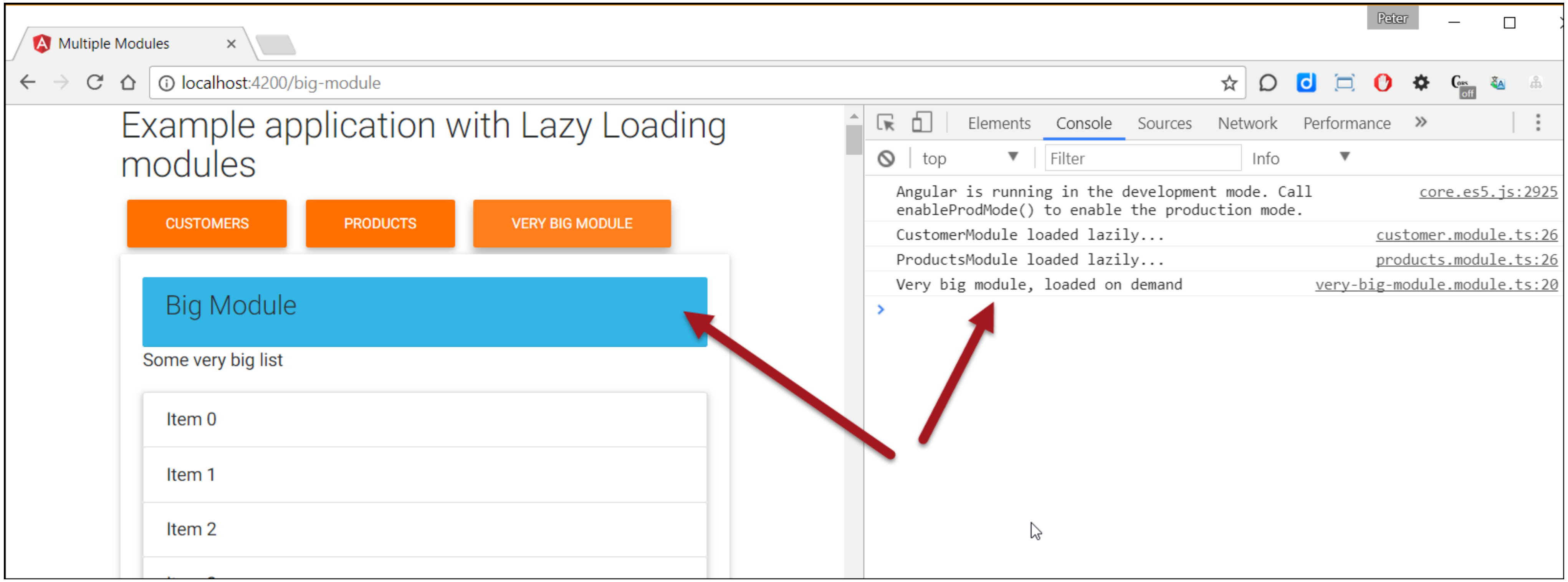
Define custom loader

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

import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/of';

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() : Observable.of(null);
   }
}
```

Official documentation https://angular.io/guide/router#custom-preloading-strategy

Run the app

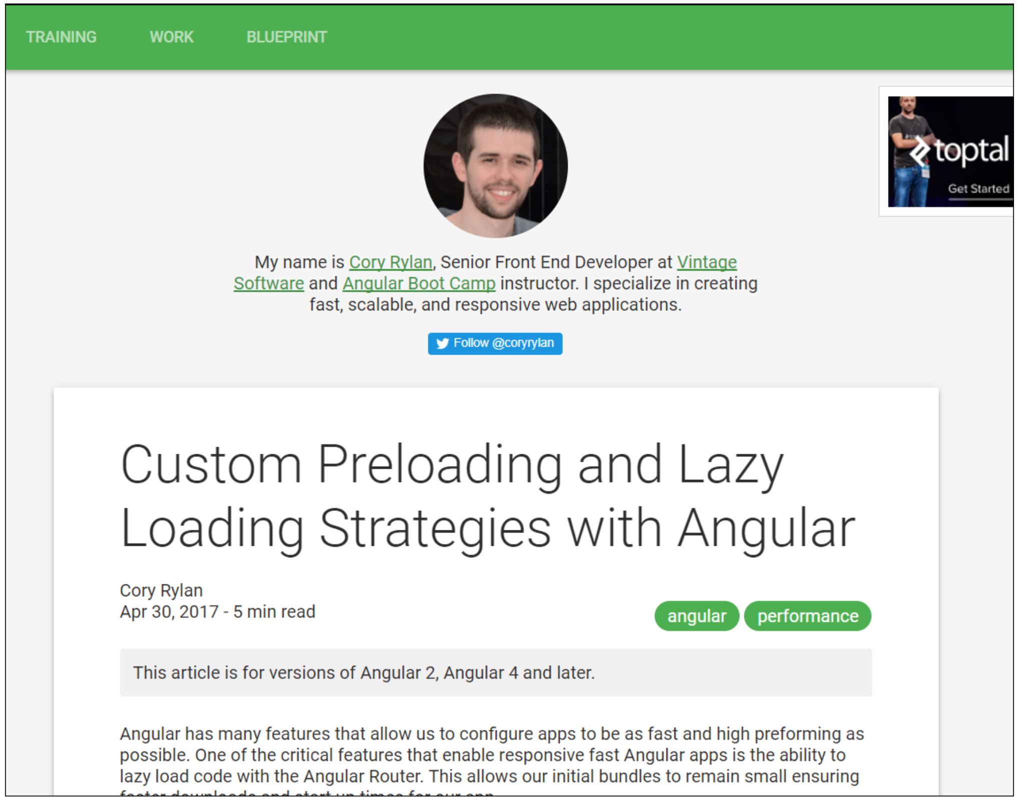
Run the app. The first 2 modules should be loaded lazily, the third module should be loaded on demand



```
Angular is running in the development mode. Call          core.es5.js:2925
enableProdMode() to enable the production mode.
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



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.

Custom Preloading and Lazy Loading Strategies with Angular

Cory Rylan
Apr 30, 2017 - 5 min read

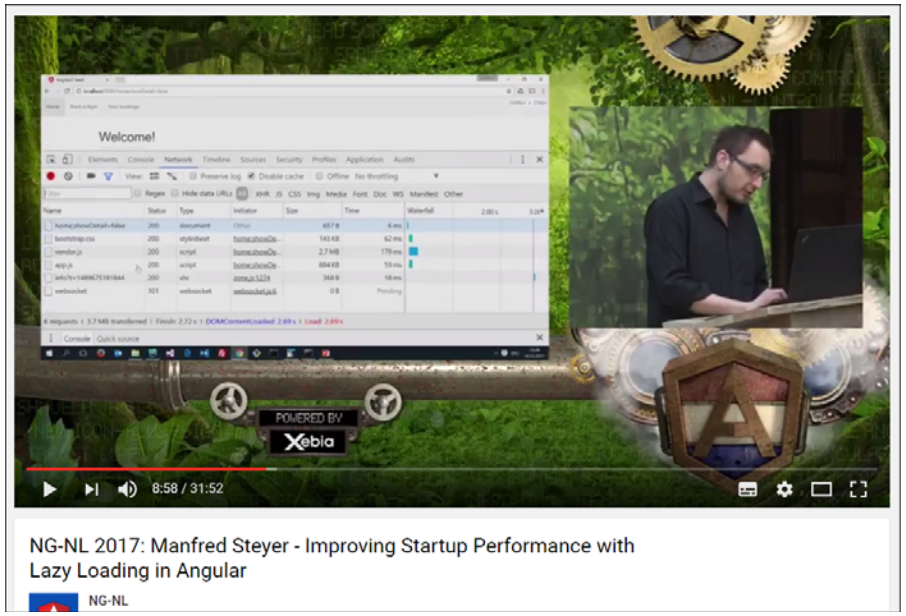
angular performance

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 preforming 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

https://coryrylan.com/blog/custom-preloading-and-lazy-loading-strategies-with-angular

Manfred Steyer - Improving Startup Performance with Lazy Loading in Angular



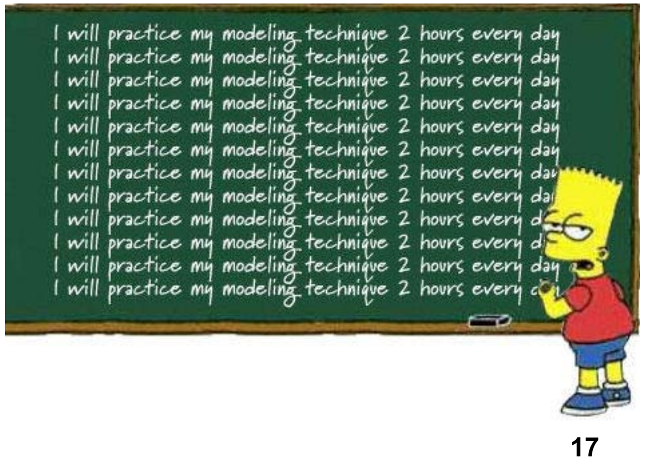
NG-NL 2017: Manfred Steyer - Improving Startup Performance with Lazy Loading in Angular

NG-NL

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

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* components be loaded lazily by adding a `data` property
- Write a custom preloading class, in `app.preloader.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`



Dynamically Loading *Components*

HOME ANGULAR RXJS WEBPACK

Angular In Depth

Sign in Get started

Chaz Gatian Follow
Web developer passionate about new technologies
Jan 10 · 6 min read

Dynamically Loading Components with Angular CLI

This post and the code for it was a team effort, including my teammates Zack

Beware!

Deep stuff.

https://blog.angularindepth.com/dynamically-loading-components-with-angular-cli-92a3c69bcd28