Lazy routing problem

Simple solution for lazy routes and bundle size optimisation

Oleksandr Honcharov

The problem

Imagine we have site routing, that is configurable from the backend. Each page has its own slug, one of predefined types and unique config.

For example: we have 2 types of page - blog and promotion (their structure is different). Each page type will load its own config from backend via resolvers and build UI based on that config.

So do we need 1 very smart lazy module (with components), that will know about each page type and store UIs for each page type? No, it isn’t scalable. The other disadvantage is that we have to load components related to promotion page in case user opens blog page.

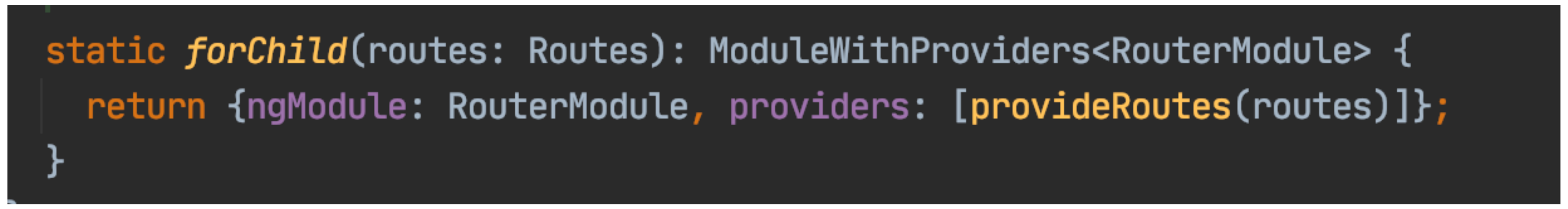
We need 1 lazy module per page type. But how can we recognise, which exactly module should be lazy-loaded?

I wanted somehow to add routes to Router at the application initialisation phase. And after few days of research I found, how it can be done.

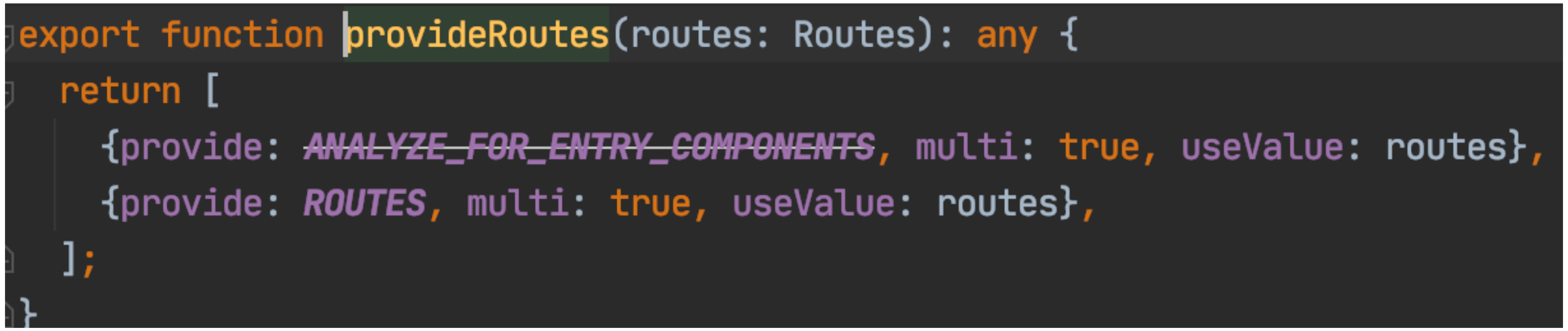
But before I explain, what I did, let me show few Angular features:

Feature 1

Routes are just object, that provided to modules with ROUTES injection token (angular RouterModule “under the hood” code):



```
static forChild(routes: Routes): ModuleWithProviders<RouterModule> {
  return {ngModule: RouterModule, providers: [provideRoutes(routes)]};
}
```



```
export function provideRoutes(routes: Routes): any {
  return [
    {provide: ~~ANALYZE_FOR_ENTRY_COMPONENTS~~, multi: true, useValue: routes},
    {provide: ROUTES, multi: true, useValue: routes},
  ];
}
```

That means, if we provide object with that token - module will use our routes object.

Feature 2

There are 2 types of injectors in angular. Node Injector and R3 injector.

The first is “component injector”, injector, which stores providers for component only (and its children). It is not interesting for us at the moment.

The second is “module injector”. It is created for each lazy module, to store all dependencies that related to that lazy module and modules, our lazy module imports.

Also each R3 injector, as well as NodeInjector, has a reference to parent, and the root R3 injector references to NullInjector (that’s why if you haven’t registered service and try to use DI mechanism to load it, you see “NullInjectorError”)

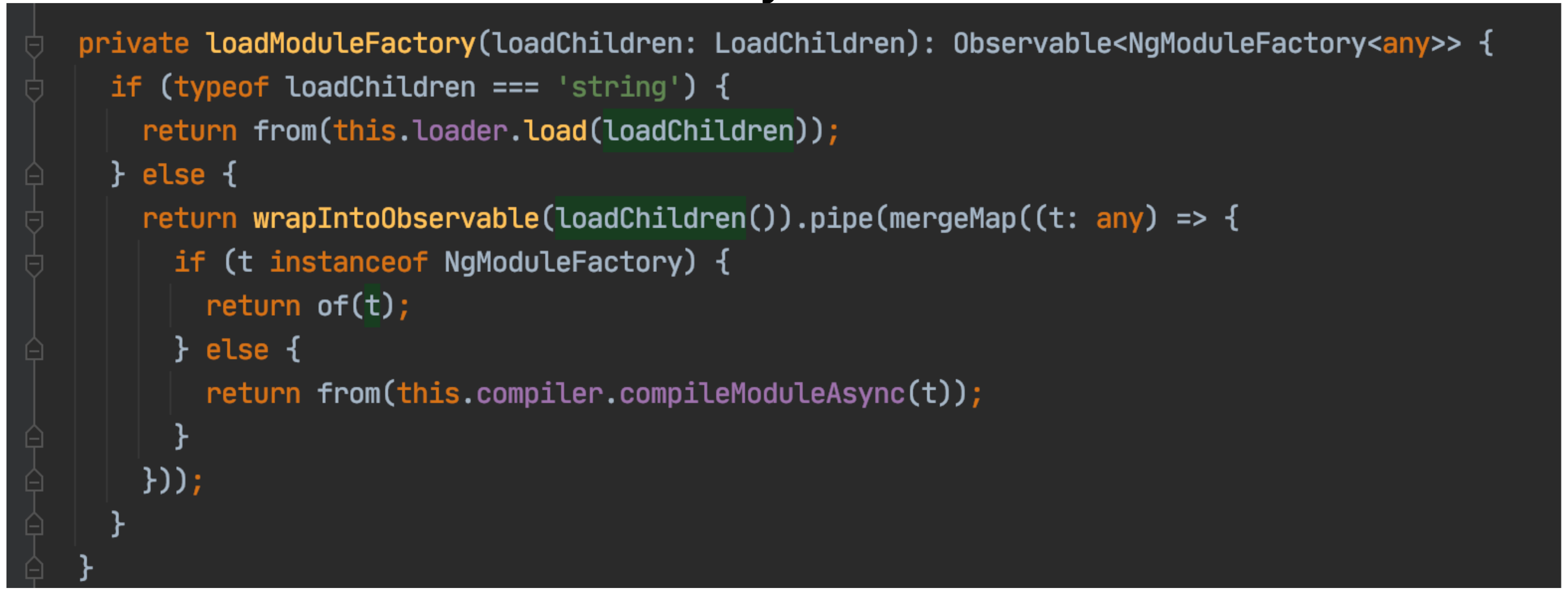
That means, if we provide token in parent injector (parent module) - child module will find it.

Feature 3

When we import module, we can import either module, or moduleWithProviders.

We can write our own forChild method for any module, and return not just module, but module with providers (like Router.forChild does).

What if we could do the same with our lazy load module in loadChildren? Let’s check it:



```
private loadModuleFactory(loadChildren: LoadChildren): Observable<NgModuleFactory<any>> {
  if (typeof loadChildren === 'string') {
    return from(this.loader.load(loadChildren));
  } else {
    return wrapIntoObservable(loadChildren()).pipe(mergeMap((t: any) => {
      if (t instanceof NgModuleFactory) {
        return of(t);
      } else {
        return from(this.compiler.compileModuleAsync(t));
      }
    }));
  }
}
```

Here you can see angular “under the hood" code, that loads lazy module (where you write “path: …,loadChildren: …“). It accepts either module, or NgModuleFactory.

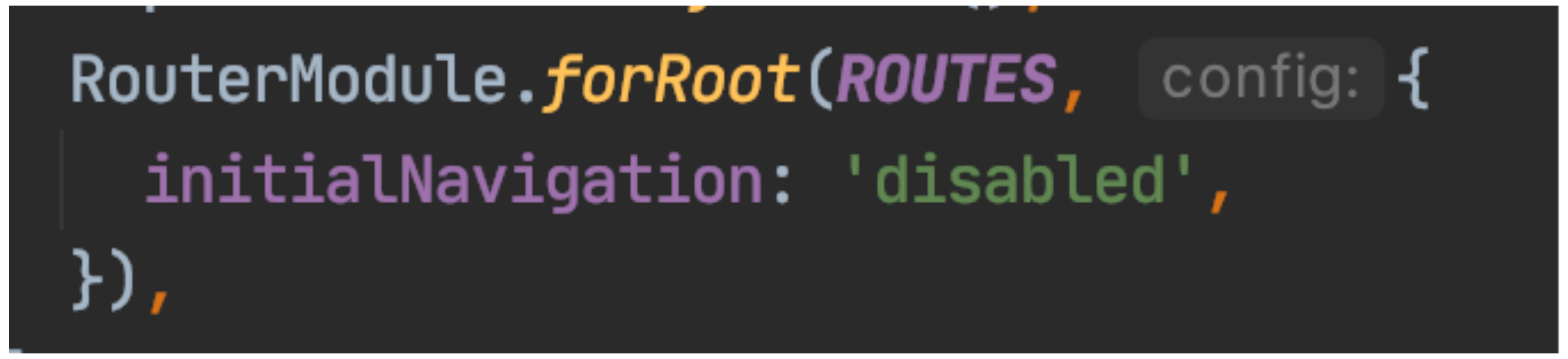
Also we know, that NgModuleFactory creates injector with reference to parent module inside.

Okay, how it is going to help us?

First step: routes for App Module

For app module we have app initialiser (we don't have such feature for lazy load modules, yet). Code in appInitialiser executes before AppModule initialisation. So we can do asynchronous calls to backend and put our routes to router.config, then call resetConfig()

1. Disable initial navigation:



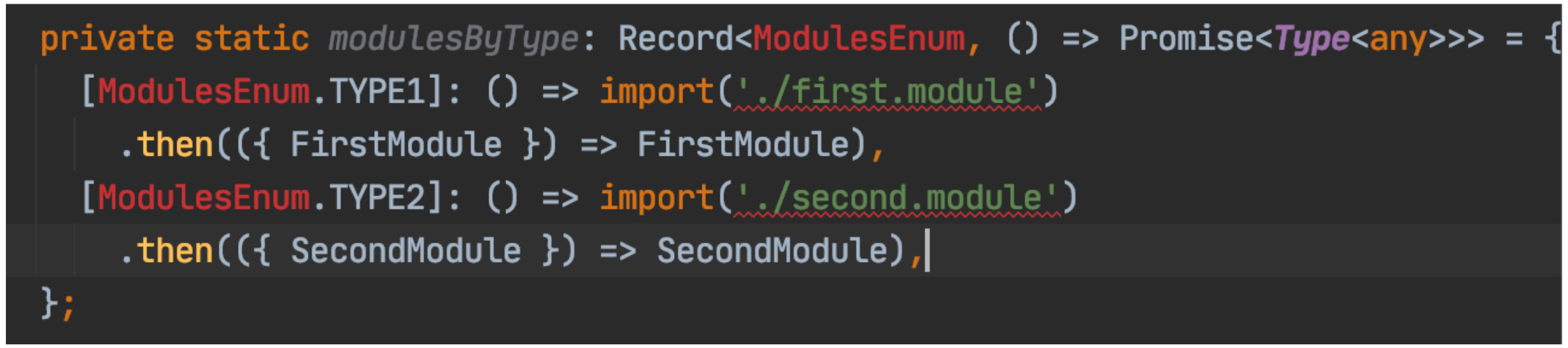
```
RouterModule.forRoot(ROUTES, config: {
  initialNavigation: 'disabled',
}),
```

2. In app initialiser GET first level routing from BE, convert it*, call router.resetConfig(config); with right config inside

3. Call router.initialNavigation();

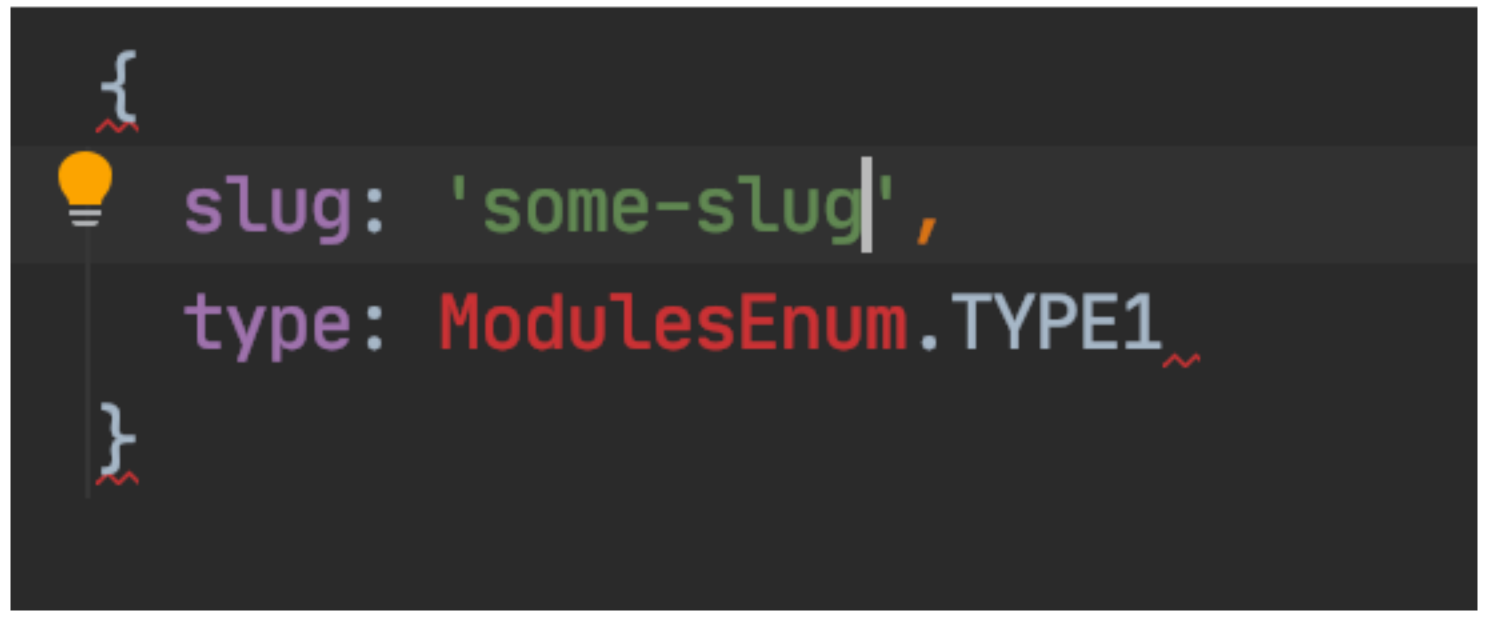
*How can we convert response to routes?

We need to create such dictionary:



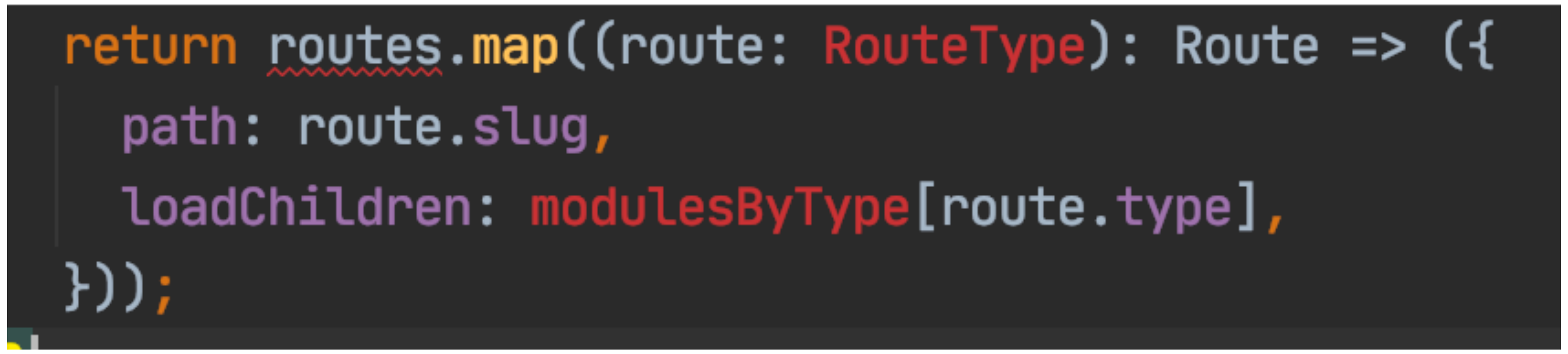
```
private static modulesByType: Record<ModulesEnum, () => Promise<Type<any>>> = {
  [ModulesEnum.TYPE1]: () => import('./first.module')
    .then(({ FirstModule }) => FirstModule),
  [ModulesEnum.TYPE2]: () => import('./second.module')
    .then(({ SecondModule }) => SecondModule),
};
```

And then, when we get response from backend, for example array of such items:



```
{
  slug: 'some-slug',
  type: ModulesEnum.TYPE1
}
```

We can easily replace it to Route objects:



```
return routes.map((route: RouteType): Route => ({
  path: route.slug,
  loadChildren: modulesByType[route.type],
}));
```

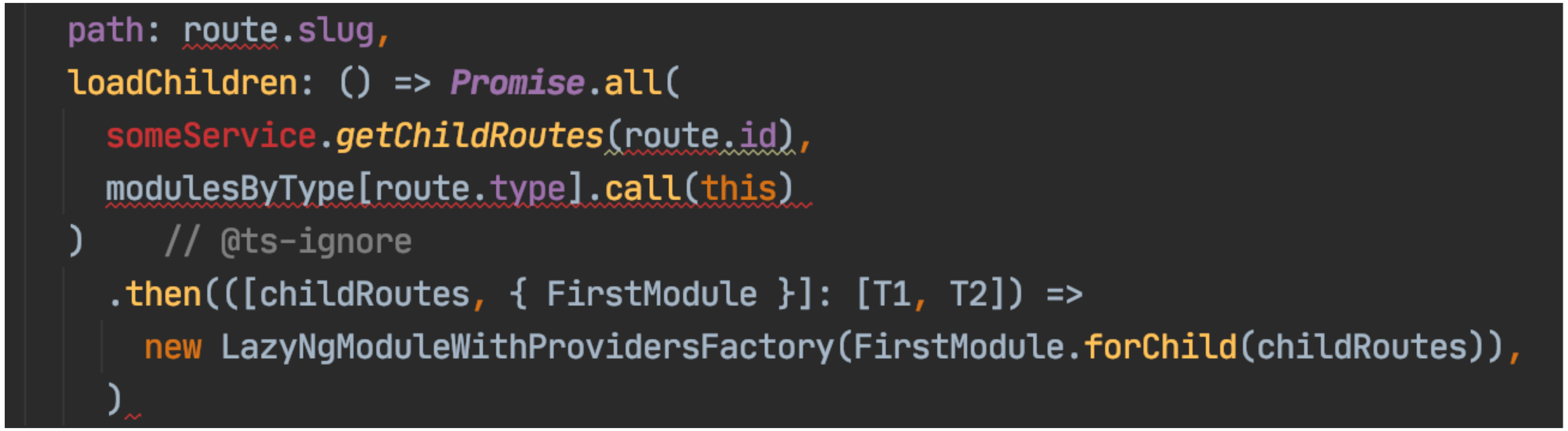
So that is how we can get correct config for router. We added routes on fly to our application, we load only that module, which we need, we still have native 404 page. Great!

Okay, but what if we need to go deeper?

Second step: Routing from BE for any child module

loadChildren is asynchronous function, isn't it? So why can't we make a request to backend along with module importing?

1:

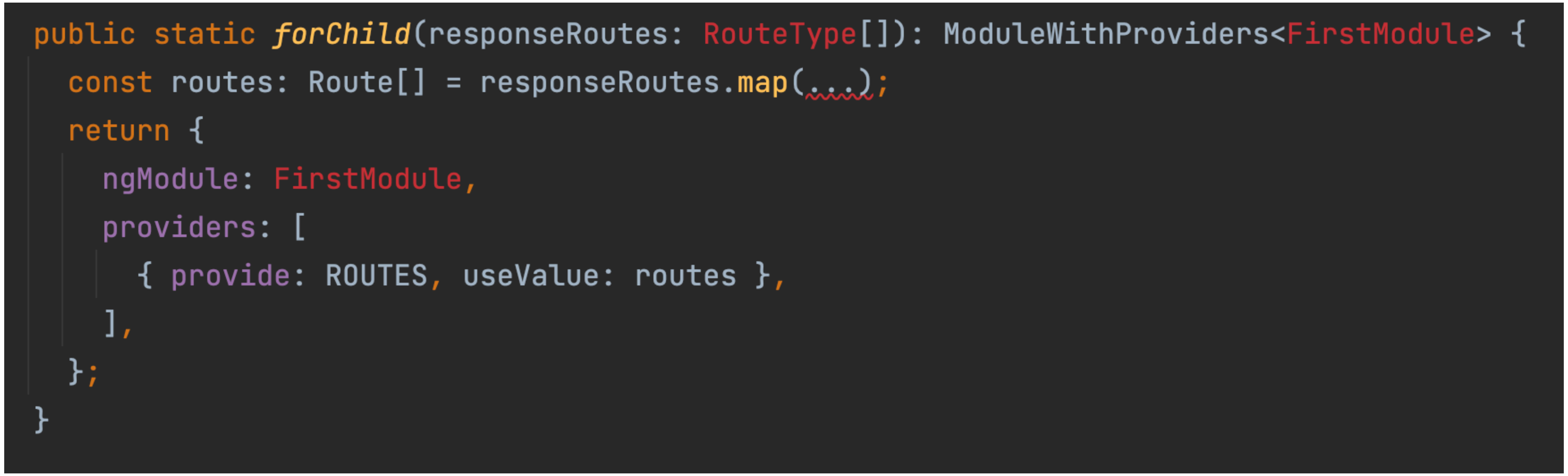


```
path: route.slug,
loadChildren: () => Promise.all(
  someService.getChildRoutes(route.id),
  modulesByType[route.type].call(this)
)    // @ts-ignore
  .then(([childRoutes, { FirstModule }]: [T1, T2]) =>
    new LazyNgModuleWithProvidersFactory(FirstModule.forChild(childRoutes)),
  )
```

*getChildRoutes is just a method of an API service, that returns agreed structure of child routes, which we will convert to Angular's Routes later.

If we look at the penultimate line, we will notice static forChild method of Module. That's how we can configure ROUTES inside that module

2. FirstModule.forChild(...):



```
public static forChild(responseRoutes: RouteType[]): ModuleWithProviders<FirstModule> {
  const routes: Route[] = responseRoutes.map(...);
  return {
    ngModule: FirstModule,
    providers: [
      { provide: ROUTES, useValue: routes },
    ],
  };
}
```

Here we just do something similar to what we did in one of previous slides: convert backend response to angular Routes

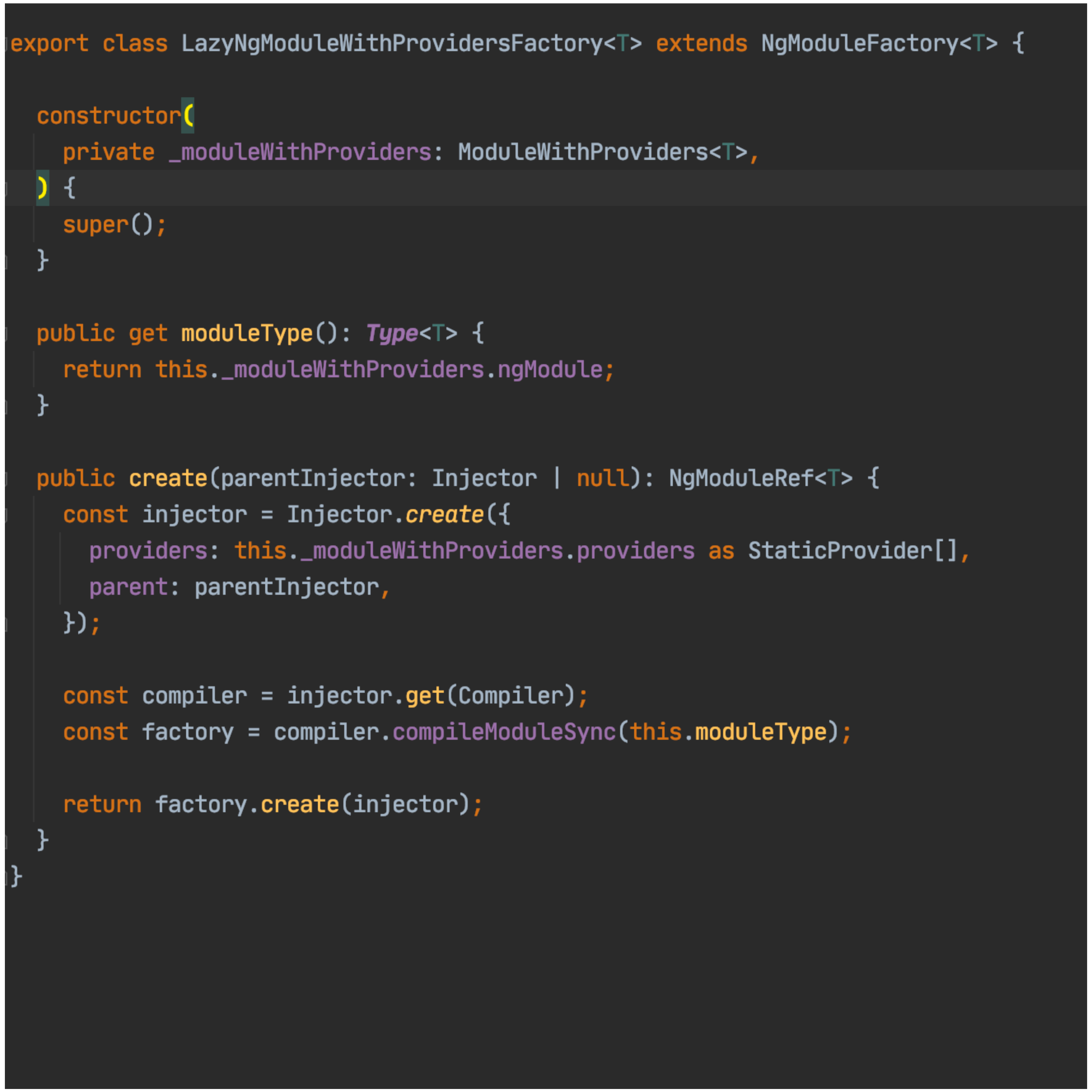
Let’s keep in mind “feature 3”, we can return in LoadChildren not module, but factory.

We can create intermediate R3 injector, put “ROUTES” token with routes inside it and make our lazy module to use that injector as a parent for its own injector.

You might noticed that in loadChildren we don’t return NgModule. Instead we return factory, when we lazy load our module (new LazyNgModuleWithProvidersFactory returns us a factory).

As we know, factory creates injector for a module. We will create the injector manually, provide our dependencies and use it as parent for our lazy module.

How? Watch next slide:



```
export class LazyNgModuleWithProvidersFactory<T> extends NgModuleFactory<T> {

  constructor(
    private _moduleWithProviders: ModuleWithProviders<T>,
  ) {
    super();
  }

  public get moduleType(): Type<T> {
    return this._moduleWithProviders.ngModule;
  }

  public create(parentInjector: Injector | null): NgModuleRef<T> {
    const injector = Injector.create({
      providers: this._moduleWithProviders.providers as StaticProvider[],
      parent: parentInjector,
    });

    const compiler = injector.get(Compiler);
    const factory = compiler.compileModuleSync(this.moduleType);

    return factory.create(injector);
  }
}
```

LazyNgModuleWithProvidersFactory

So, when Angular calls loadChildren - we returns our factory, that creates injector inside it, and sets it as a parent injector for our lazy loaded module. Then we provide ROUTES token with routes inside that injector.

That’s why, when our lazy loaded module try to find ROUTES token and can’t find it in its own injector, it goes to parent injector, and find ROUTES inside it.

In conclusion

Such approach has a lot of benefits - for example:

- If you have a lot of page types - you will be loading only module you need, instead of loading all of them
- You have native 404 pages and clean router tree
- It is very flexible, so you can combine it even with the microfrontend approach

External resources

- https://github.com/angular/angular/tree/master/packages/router
- https://github.com/angular/angular/issues/34351
- https://github.com/angular/angular/issues/17606

Thank you for your attention!