Lukas Marx
August 05, 2017

Angular

Lern how to split your Angular App into Modules [Includes Lazy-Loading]

Want to get a better understanding of all the different building blocks of an angular application?

Don't look any further!

In this article, we will take a close look at modules.

Modules in angular are a great way to share and reuse code across your application.

Shared modules do not only make your app tidier, but can reduce the actual size of an application by far.

Following this tutorial, we will create your first custom module and discover all of its components.

Afterward, we will learn how to use our module to enable lazy-loading, that leads to smaller applications and therefore happier users.

Ready? Let's go!

App Module

In Angular, everything is organized in modules. So even if you don't know what modules are and how to use them, you are already using them in your application. The most prominent of them is the AppModule.

This module is the root module of your application and is absolutely necessary to run your application. In here you define, which components your app is using and what other modules you might want to use. So how does it look like?

Here is a basic AppModule generated by the angular-cli.

src/app/app.module.ts

```ts
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'

import { AppComponent } from './app.component'

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

To generate your own module, open a terminal at the root of your application-project.

Use the following command to generate the files for the new module. These files already contain the first draft of a module.

```
ng generate module [name]
```

What is an Angular Module?

Simply put, a module is just a class, just like components and services.

The code in angular is generally organized in modules. You can think of modules like packages or bundles containing the required code for a specific use case.

The most prominent Module is the App-Module, because it comes with every new application, generated by the cli.

However, chances are that the App-Module is not the only module you have encountered so far. There are many other modules that come with angular out of the box.

Examples are the Http-Client-Module, contains a very useful Http-Client (surprise!) and the Forms-Module that contains UI components and directives to HTML-Forms.

As we have seen in the example above, we need to import a module first, before we can use it.

The root module of your application is the App-Module. This

module imports other modules, which can import other modules them self.

Just like with components, the resulting structure is a module-tree.

@NgModule

Inside of the @NgModule operator, we define all the properties of the module. For that, we provide a simple Javascript object as the parameter. Let's take a closer look, at what each property of that object actually does:

Bootstrap

Defines the root-component of the Application. Only use this in the AppModule.

Exports

We define the components, directives or pipes we want to export here. That means, that our module is providing these to other modules when they get imported. Otherwise, these components stay module internal and can not be accessed from the outside.

Declarations

Inside of the declarations array, we define all the components, directives and pipes, that are declared and used inside this module. If a component (or directive or pipe) is not added to the declarations array and you use it in your module/application, angular will throw an error at runtime. Also, a component (or ... you got it) can only be declared in one module. If you want to use your component in multiple modules, you need to bundle that component into a separate module and import that in the module.

Imports

Speaking of importing... Your module can import as many sub-modules as you like. Don't have defined any custom modules yet? No problem, we will get to that. But even if you don't have any modules, you still need to import some angular modules. As I mentioned earlier, Angular is built with modularity in mind. While many features are contained in angular's core, some features are bundled into their own module. For example, if you want to use the HttpClient, you will need to import the HttpClientModule.

Providers

We define any @Injectables, required by the module, here. Any sub-components or modules can then get the same instance of that @Injectable via dependency injection. In the case of the AppModule, these @Injectables are application-scoped.

[object Object]

Building a custom Module

So, let's pretend, we have created an awesome application. That app has only one module, which is AppModule.

Now, we want to add a login area to our application. It will contain a login-page and a register-page. Maybe also a help-page to help the user with the authentication process. Each page is represented by its own component.

undefined

```
LoginComponent
RegisterComponent
HelpComponent
```

We also need a service to make the Http-request.

undefined

```
AuthenticationService
```

Since these pages are completely separate and have nothing to do with the content-pages of our application, we decide to bundle them into a separate module. We call that module AuthentictionModule

TS src/app/authentication/authentication.module.ts

```ts
import { NgModule } from '@angular/core'
import { CommonModule } from '@angular/common'

@NgModule({
  imports: [CommonModule],
})
export class AuthenticationModule {}
```

Now, we add our components to the declarations section. We also add them to the exports sections, because we want to

use them outside of the module.

TS src/app/authentication/authentication.module.ts

```
import { HelpComponent } from
'./help/help.component'
import { RegisterComponent } from
'./register/register.component'
import { LoginComponent } from
'./login/login.component'
import { NgModule } from '@angular/core'
import { CommonModule } from '@angular/common'

@NgModule({
  imports: [CommonModule],
  declarations: [LoginComponent,
RegisterComponent, HelpComponent],
  exports: [LoginComponent, RegisterComponent,
HelpComponent],
})
export class AuthenticationModule {}
```

That was already it. Now we can import the module in our AppModule and use it's components e.g. in the AppComponent.

TS src/app/app.module.ts

```
import { AuthenticationModule } from
'./authentication/authentication.module'
import { BrowserModule } from '@angular/platform-
browser'
import { NgModule } from '@angular/core'

import { AppComponent } from './app.component'

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, AuthenticationModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}

;<app-login />
```

Lazy-Loading Modules

It turns out, you can do more with modules, than just organizing your components. It is also possible to lazy-load modules. So what does that mean?
Angular appears to be quite heavy in download size. Depending on your use case, that can be a big issue.

Especially on mobile, it can take a while to download only the application. A way to reduce loading time, is to split your application into modules.

When you load your modules in a lazy way, it is not included in the initial application. Instead, it is only downloaded, when it is needed. Why should we download components we do not show anyway?

So how does it work?

Let's modify the previous example to use lazy loading. To implement that, we need to add routing to our application.

First, we configure the routing module with our route configuration. To do so, create a new file called app.routing.ts next to the app.module.

TS src/app/app.routing.ts

```
import { ContentComponent } from
'./content/content.component'
import { Routes, RouterModule } from
'@angular/router'
import { ModuleWithProviders } from
'@angular/core'

export const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo:
'content' },
  { path: 'content', component: ContentComponent
},
  {
    path: 'login',
    loadChildren:
'./authentication/authentication.module#Authenticatio
  },
]

export const routing: ModuleWithProviders =
RouterModule.forRoot(routes)
```

Components, that are not lazy-loaded are specified with a path and a component property. If we want to lazy load a module at a specific route, we do so by using the loadChildren property. Here we specify the path and the name of the module, separated by a #.

Afterward, we can import the configured module in our AppModule. We also remove the import of our AuthenticationModule, since that is lazy loaded.

TS src/app/app.module.ts

```
import { routing } from './app.routing'
import { BrowserModule } from '@angular/platform-
browser'
import { NgModule } from '@angular/core'

import { AppComponent } from './app.component'
import { ContentComponent } from
'./content/content.component'

@NgModule({
  declarations: [AppComponent, ContentComponent],
  imports: [
    BrowserModule,
    routing, //import routing
  ],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Next, we need to have a router-outlet somewhere in our application. So let's place one in the AppComponent.

src/app/app.component.html

```
<router-outlet></router-outlet>
```

If we now hit that route, our module is loaded. But there is nothing on the screen. That is because Angular does not know, which component to show. To fix that, we have to define sub-routes for our authentication module. That looks almost exactly like the app.routing. Except this time, we call forChild() instead of forRoot(). Of course, the routes are different, as well.

TS src/app/authentication/authentication.routing.ts

```
import { AuthenticationComponent } from
'./authentication.component'
import { HelpComponent } from
'./help/help.component'
import { RegisterComponent } from
'./register/register.component'
import { LoginComponent } from
'./login/login.component'
import { Routes, RouterModule } from
'@angular/router'
import { ModuleWithProviders } from
'@angular/core'

export const routes: Routes = [
  { path: '', component: LoginComponent }, //
default route of the module
  { path: 'login', component: LoginComponent },
  { path: 'register', component:
RegisterComponent },
  { path: 'help', component: HelpComponent },
]

export const routing: ModuleWithProviders =
RouterModule.forChild(routes)
```

All that's left now, is to import the routing into our AuthenticationModule. When we hit the route now, the login component is shown. That is because we configured it to be the default route.

TS src/app/authentication/authentication.module.ts

```
import { AuthenticationComponent } from
'./authentication.component'
import { routing } from
'./authentication.routing'
import { HelpComponent } from
'./help/help.component'
import { RegisterComponent } from
'./register/register.component'
import { LoginComponent } from
'./login/login.component'
import { NgModule } from '@angular/core'
import { CommonModule } from '@angular/common'

@NgModule({
  imports: [
    CommonModule,
    routing, // import routing
  ],
  declarations: [
    AuthenticationComponent,
    LoginComponent,
    RegisterComponent,
    HelpComponent,
  ],
})
export class AuthenticationModule {}
```

Angular Modules are not Javascript Modules

Don't get confused with angular modules and Javascript modules. Angular modules are the classes, marked with @NgModule. At the other hand, when we import some module using the Typescript import keyword, we are importing a Javascript module. That Javascript module can contain Angular modules.

Angular Module

TS src/app/app.module.ts

```
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
```

```
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Javascript Module

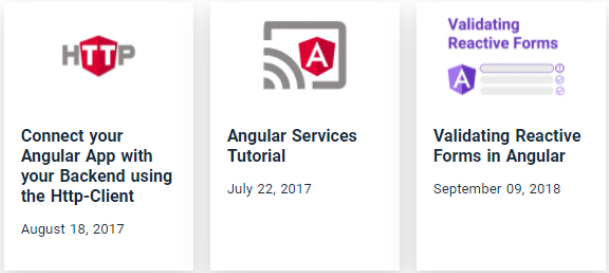
```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
```

Conclusion

Now you know how angular modules work, how to organize your application and increase your app's performance with lazy loading.

If you liked this article, please consider sharing it with your developer-friends and colleges.

Also, if you want to be informed about new, amazing content, follow me on twitter @malcoded.



Furkan Katman 10 months

Amazing explanation about multiple ngmodules. Thanks god I saw your article :))

Ahmed Rebai 12 months

Hi, I have a question about submodules, if I will generate another module inside the AuthModule with other components, how the routing should be implemented? there are who are talking about PreloadAllModules strategy, thx

A

Akshay shelke 13 months

The artical was very understandable,100000 likes to this artical from my side. Appreciated, Thank you.

D

Dave Hickson 16 months

Very well written article which helped much more than most of the others I've read on this topic. Much appreciated, thanks.

Srinivasan Hariharan 17 months

Thanks.. very nice article. One point I wanted to ask, which is as follows:
Eventhough i had modified my sample code into modules, eveerything is working as expected. But, while running the application, in the chrome developer network tab, i dont find the modules are downloaded separately, when i access my link related to my modules. Am i missing something.

Yassine Abainou 18 months

Thanks for this useful article

salim ahammed 20 months

Very nice article.Thanx for uploading it

Prasanna Mohite 20 months

really nice article.. Appreciate your efforts.

Samuel Deering 22 months

Thanks great article.

D

Daniel Patfield 23 months

One last question. After creating my modules and importing/exporting properly. And adding router-outlet - all the links to each module are encapsulated in their respective module, correct? Or do I still need to list them all at the AppModule level?

D

Daniel Patfield 23 months

Another question - if I have 3 separate areas (4 including authentication) as modules - do I need a "ContentComponent" still?

D

Daniel Patfield 23 months

Great article! Question(s)... I am going to create a site with 3 separate areas to access - will it serve me better to have each area created as a module?

keerthika mahendralingam 23 months

Great article.. Thanks for sharing..

krishna ananthi 2 years

Good one

Chidubem Nkwo 2 years

Great Article, Thanks for sharing

Mohanraj Ponnambalam 2 years

Excellent Post... Keep writing...

Emanuele Gravela 2 years

Great! But when we organize a module maybe we should also pay attention to the provider and no providers are declared in the above module!

J

Johannes Sebolela 2 years

Great article. I needed all the topics provided therein.

A

Abhilash Manjunath 2 years

Thank you so much for this article. The best guide for a beginner. Please do come up with such articles.

RAKOTOZAFY M. J. Njarasoa 2 years

Nice article. Bit error: Angular modules are the classes, marked with @NgModule instead of @NgModel.

Lukas Marx 2 years

Thank you, I've fixed that!

V

Vasantha Kumar 2 years

Nice article. Thank you.

Станислав Викторович 2 years

Not anuogh information...

Явор Милинов 2 years

Thank you for that guide.

Shiva Digital Web 2 years

Finally understood what modules are and how to use them in Angular. Please keep writing about Angular and any of the technologies that you are interested in. Thanks for this nice write!!