Angular : sous-module

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Angular

Pour créer un nouveau sous-module

```
ng generate module module-name
```

Angular

Pour créer un nouveau sous-module

```
ng generate module module-name
```

Ou utiliser le raccourci

```
ng g m module-name
```

Angular

Pour créer un sous-module `vehicule` (ce dernier ne sera pas enregistré dans `app.module.ts`)

```
ng g m vehicule
```

Angular

Pour créer un sous-module `vehicule` (ce dernier ne sera pas enregistré dans `app.module.ts`)

```
ng g m vehicule
```

Pour créer un sous-module `vehicule` et l'enregistrer dans `app.module.ts`

```
ng g m vehicule --module=app
```

Angular

Pour créer un sous-module `vehicule` (ce dernier ne sera pas enregistré dans `app.module.ts`)

```
ng g m vehicule
```

Pour créer un sous-module `vehicule` et l'enregistrer dans `app.module.ts`

```
ng g m vehicule --module=app
```

Pour créer un sous-module `vehicule`, l'enregistrer dans `app.module.ts` et créer son module de routage (l'ordre des options n'a pas d'importance)

```
ng g m vehicule --module=app --routing
```

Angular

Pour bien structurer le projet, on regroupe les modules dans un répertoire `modules`

```
ng g m modules/vehicule --module=app --routing
```

Angular

Pour bien structurer le projet, on regroupe les modules dans un répertoire `modules`

```
ng g m modules/vehicule --module=app --routing
```

Le résultat est :

```
CREATE
src/app/modules/vehicule/vehicule-routing.module.ts
(252 bytes) CREATE
src/app/modules/vehicule/vehicule.module.ts (288
bytes) UPDATE src/app/app.module.ts (680 bytes)
```

Angular

Constats

- Deux fichiers créés : `vehicule.module.ts` et `vehicule-routing.module.ts`
- Une mise à jour effectuée : enregistrement de ce sous-module dans `app.module.ts`
- Si l'option `--module=app` n'a pas été précisée, il faut déclarer `vehicule.module.ts` dans `app.module.ts`

Angular

Pour déclarer `vehicule.module.ts` **dans** `app.module.ts`

```
import { VehiculeModule } from './vehicule/vehicule.module';
// + les autres imports

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

Angular

Deux méthodes différentes pour la création de nouveaux composants dans un sous-module

- Se placer dans le sous-module et créer le nouveau composant
- Préciser le nom du module au moment de la création du composant

Angular

Exemple avec la première méthode

```
cd src/app/modules/vehicule ng g c voiture
```

Angular

Exemple avec la première méthode

```
cd src/app/modules/vehicule ng g c voiture
```


Le résultat est

```
CREATE src/app/modules/vehicule/voiture/voiture.component.html (22 bytes) CREATE
src/app/modules/vehicule/voiture/voiture.component.spec.ts (635 bytes) CREATE
src/app/modules/vehicule/voiture/voiture.component.ts (273 bytes) CREATE
src/app/modules/vehicule/voiture/voiture.component.css (0 bytes) UPDATE
src/app/modules/vehicule/vehicule.module.ts (446 bytes)
```

Angular

Exemple avec la deuxième méthode

```
ng g c modules/vehicule/camion
```

Angular

Exemple avec la deuxième méthode

```
ng g c modules/vehicule/camion
```

Le résultat est

```
CREATE src/app/modules/vehicule/camion/camion.component.html (21 bytes) CREATE
src/app/modules/vehicule/camion/camion.component.spec.ts (628 bytes) CREATE
src/app/modules/vehicule/camion/camion.component.ts (269 bytes) CREATE
src/app/modules/vehicule/camion/camion.component.css (0 bytes) UPDATE
src/app/modules/vehicule/vehicule.module.ts (364 bytes)
```

Remarque

Les deux mises à jour effectuées correspondent à la déclaration de ces deux composants dans `vehicule.module.ts`

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

import { VehiculeRoutingModule } from './vehicule-routing.module';
import { CamionComponent } from './camion/camion.component';
import { VoitureComponent } from './voiture/voiture.component';

@NgModule({
  declarations: [CamionComponent, VoitureComponent],
  imports: [
    CommonModule,
    VehiculeRoutingModule
  ]
})
export class VehiculeModule { }
```

Angular

Question 1 : pourquoi `CommonModule` ?

C'est le module contenant les pipes et les directives **Angular**

Angular

Question 1 : pourquoi `CommonModule` ?

C'est le module contenant les pipes et les directives **Angular**

Question 2 : `CommonModule` n'est pas importé dans `app.module.ts`, pourquoi ?

Dans `app.module.ts`, on importe `BrowserModule` et ce dernier importe `CommonModule`

Angular

Commençons par créer un module de routage pour le sous-module `vehicule` (si on n'a pas ajouté l'option `--routing` à la création)

```
ng generate module vehicule/vehicule-routing --flat
--module=vehicule
```

Angular

Commençons par créer un module de routage pour le sous-module `vehicule` (si on n'a pas ajouté l'option `--routing` à la création)

```
ng generate module vehicule/vehicule-routing --flat
--module=vehicule
```

Explication

- `vehicule/vehicule-routing` : le module de routage appelé `vehicule-routing` sera créé dans le répertoire du module `vehicule`
- `--flat` pour ne pas créer un répertoire `vehicule-routing`
- `--module=vehicule` pour déclarer le module créé dans `vehicule.module.ts`

Angular

Deux solutions

- Définir les routes dans `app-routing.module.ts`
- Définir une base pour le module dans `app-routing.module.ts` et une route pour chaque composant de ce module dans `vehicule-routing.module.ts`

Angular

Dans `app-routing.module.ts`**, définissons les routes** `/vehicule/camion`**,** `/vehicule/voiture` **et** `/vehicule`

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

// + les autres imports de components

const routes: Routes = [
  { path: 'stagiaire', component: StagiaireComponent },
  { path: 'stagiaire/:nom/:prenom', component: StagiaireComponent },
  { path: 'adresse', component: AdresseComponent },
  { path: 'trainee', redirectTo: '/stagiaire' },
  { path: 'error', component: ErrorComponent },
  {
    path: 'vehicule', children: [
      { path: 'camion', component: CamionComponent },
      { path: 'voiture', component: VoitureComponent },
      { path: '', component: VoitureComponent }
    ]
  },
  { path: '', redirectTo: '/stagiaire', pathMatch: 'full' },
  { path: '**', redirectTo: '/error' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Angular

Rien à ajouter dans `vehicule-routing.module.ts`

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';


const routes: Routes = [];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class VehiculeRoutingModule { }
```

Ajoutons le constructeur suivant dans `vehicule.module.ts`

```
export class VehiculeModule {
  constructor() { console.log('vehicule-module'); }
}
```

Ajoutons le constructeur suivant dans `vehicule.module.ts`

```
export class VehiculeModule {
  constructor() { console.log('vehicule-module'); }
}
```

Ajoutons le constructeur suivant dans `app.module.ts`

```
export class AppModule {
  constructor() { console.log('app-module'); }
}
```

Ajoutons le constructeur suivant dans `vehicule.module.ts`

```
export class VehiculeModule {
  constructor() { console.log('vehicule-module'); }
}
```

Ajoutons le constructeur suivant dans `app.module.ts`

```
export class AppModule {
  constructor() { console.log('app-module'); }
}
```


Explication

- Allez sur `/vehicule/camion` et `/vehicule/voiture` et vérifiez que leurs composants respectifs s'affichent
- Allez aussi les routes précédentes (`/adresse` par exemple) et vérifier que dans tous les cas les deux messages `app-module` et `vehicule-module` sont affichés dans la console.

Angular

Deuxième solution : dans `app-routing.module.ts`, commentons la dernière partie ajoutée et les deux dernières routes

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

// + les autres imports de components

const routes: Routes = [
  { path: 'stagiaire', component: StagiaireComponent },
  { path: 'stagiaire/:nom/:prenom', component: StagiaireComponent },
  { path: 'adresse', component: AdresseComponent },
  { path: 'trainee', redirectTo: '/stagiaire' },
  { path: 'error', component: ErrorComponent },
  // {
  //   path: 'vehicule', children: [
  //     { path: 'camion', component: CamionComponent },
  //     { path: 'voiture', component: VoitureComponent },
  //     { path: '', component: VoitureComponent }
  //   ]
  // },
  // { path: '', redirectTo: '/stagiaire', pathMatch: 'full' },
  // { path: '**', redirectTo: '/error' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Angular

Ajoutons le routage dans `vehicule-routing.module.ts`

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

import { VoitureComponent } from './voiture/voiture.component';
import { CamionComponent } from './camion/camion.component';

const routes: Routes = [
  { path: 'camion', component: CamionComponent },
  { path: 'voiture', component: VoitureComponent },
];
@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class VehiculeRoutingModule { }
```

Ainsi, nos chemins sont `/voiture` et `/camion`.

Angular

Lazy loading

- On doit définir la base dans `app-routing.module.ts`
- Faire référence à `vehicule-routing.module.ts` avec la clé `loadChildren`

Modifions le contenu du fichier `app-routing.module.ts` (solution utilisée jusqu'à la version 8 d'Angular)

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
// + les autres imports de components

const routes: Routes = [
  { path: 'stagiaire', component: StagiaireComponent },
  { path: 'stagiaire/:nom/:prenom', component: StagiaireComponent },
  { path: 'stagiaire', component: StagiaireComponent },
  { path: 'adresse', component: AdresseComponent },
  { path: 'trainee', redirectTo: '/stagiaire' },
  { path: 'error', component: ErrorComponent },
  { path: 'vehicule', loadChildren: './modules/vehicule/vehicule.module
    #VehiculeModule' },
  { path: '', redirectTo: '/stagiaire', pathMatch: 'full' },
  { path: '**', redirectTo: '/error' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Angular

Dé-commentons les routes dans `vehicule-routing.module.ts`

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
// + les autres imports de components

const routes: Routes = [
  { path: 'camion', component: CamionComponent },
  { path: 'voiture', component: VoitureComponent },
  { path: '', component: VoitureComponent }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class VehiculeRoutingModule { }
```

Angular

Dans `app.module.ts`, supprimer l'importation du module `VehiculeModule`

```
@NgModule({
  declarations: [
    AppComponent,
    AdresseComponent,
    StagiaireComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {
  constructor() { console.log('app-module'); }
}
```

Angular

Ainsi, en saisissant

- `/vehicule` : le composant `voiture` sera affiché
- `/vehicule/voiture` : le composant `voiture` sera affiché
- `/vehicule/camion` : le composant `camion` sera affiché

Angular

Ainsi, en saisissant

- `/vehicule` : le composant `voiture` sera affiché
- `/vehicule/voiture` : le composant `voiture` sera affiché
- `/vehicule/camion` : le composant `camion` sera affiché

Remarque

- Vérifier dans la console qu'en lançant l'application sur `localhost:4200`, seul le message `app-module` est affiché
- Le message `vehicule-module` est affiché seulement si on demande une route d'un composant du module `vehicule` ⇒ **lazy loading** (chargement paresseux : charger les modules à la demande)

Angular

On peut aussi utiliser les promesses (solution utilisée depuis Angular 8) : seul mode de chargement supporté par Ivy

```
const routes: Routes = [
  { path: 'stagiaire', component: StagiaireComponent },
  { path: 'stagiaire/:nom/:prenom', component: StagiaireComponent },
  { path: 'adresse', component: AdresseComponent },
  { path: 'trainee', redirectTo: '/stagiaire' },
  { path: 'error', component: ErrorComponent },
  {
    path: 'vehicule',
    loadChildren: () => import('./modules/vehicule/vehicule.module').
      then(m => m.VehiculeModule)
  },
  { path: '', redirectTo: '/stagiaire', pathMatch: 'full' },
  { path: '**', redirectTo: '/error' }
];
```

Angular

Exercice

- Créez deux nouveaux sous-modules `cours` et `exercice` (sans les déclarer dans `app.module.ts` et avec un module de routage pour chacun)
- Déplacez les composants `stagiaire`, `adresse` et `observable` dans le sous-module `cours` et `tableau` et `calcul` dans le sous-module `exercice`
- Déclarez les composants `stagiaire`, `adresse` et `observable` dans `cours.module.ts` et `tableau` et `calcul` dans `exercice.module.ts`
- Définissez des nouvelles routes vers les composants déplacés et qui doivent commencer par `/cours` ou `/exercice`

Angular

Nouveau contenu de `routes` **défini dans** `app-routing.module.ts`

```
const routes: Routes = [
  {
    path: 'vehicule',
    loadChildren: () => import('./modules/vehicule/vehicule.module').then(m => m.VehiculeModule
      )
  },
  {
    path: 'cours',
    loadChildren: () => import('./modules/cours/cours.module').then(m => m.CoursModule)
  },
  {
    path: 'exercice',
    loadChildren: () => import('./modules/exercice/exercice.module').then(m => m.ExerciceModule
      )
  },
  { path: 'error', component: ErrorComponent },
  { path: '**', redirectTo: '/error' }
];
```