

# Angular : 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`



- 1 Création de module
- 2 Création de nouveaux composants dans un module
- 3 Routage et chargement de module
  - Eager loading
  - Lazy loading
- 4 Exportation d'un composant d'un module à un autre
- 5 Composant autonome (`standalone`)

## Pour créer un nouveau sous-module

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

© Achref EL MOUADIB

# 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
```

© Achref EL MOUELHI ©

# 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
```

© Achref EL MOUËL

# 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 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
```

© Achref EL MOUELHI ©

# Angular

## Exemple avec la première méthode

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

## Depuis Angular 17, il faut aussi ajouter

```
ng g c voiture --standalone=false ng g c voiture
```

# Angular

## Exemple avec la première méthode

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

## Depuis Angular 17, il faut aussi ajouter

```
ng g c voiture --standalone=false 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 --standalone=false
```

© Achref EL MOULI

# Angular

## Exemple avec la deuxième méthode

```
ng g c modules/vehicule/camion --standalone=false
```

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` est-il dans la liste de modules importés ?

© Achref EL MOUADJIB

# Angular

## Question 1

Pourquoi `CommonModule` est-il dans la liste de modules importés ?

## Réponse

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

# Angular

## Question 2

CommonModule n'a pas été importé dans `app.module.ts`, pourquoi ?

© Achref EL MOUADJIB

# Angular

## Question 2

CommonModule n'a pas été importé dans `app.module.ts`, pourquoi ?

## Réponse

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

# Angular

## Deux modes de chargement de modules avec **Angular**

- **Eager loading** : chargement de tous les modules
- **Lazy loading** : chargement du module contenant le composant demandé

# 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
```

© Achref EL MOUETT

# 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`

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

```
const routes: Routes = [
  { path: '', component: HomeComponent },
  { 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: '/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'); }  
}
```

© Achref EL MOUELHI ©

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'); }  
}
```

© Achref EL

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

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

```
const routes: Routes = [
  { path: '', component: HomeComponent },
  { 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: '/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: 'vehicule', children: [
      { path: 'camion', component: CamionComponent },
      { path: 'voiture', component: VoitureComponent },
      { path: '', component: VoitureComponent }
    ]
  },
];

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

# Angular

## Pour tester

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

# Angular

## Lazy loading : étapes

- Définir la base dans `app-routing.module.ts`
- Référencer à `vehicule-routing.module.ts` avec la clé `loadChildren`
- Supprimer l'importation du module `VehiculeModule` dans `app.module.ts`

# Angular

**Modifions le tableau `routes` dans le fichier `app-routing.module.ts` (solution utilisée jusqu'à la version 8 d'Angular)**

```
const routes: Routes = [  
  { path: '', component: HomeComponent },  
  { 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: '/error' }  
];
```

# Angular

On peut aussi utiliser les promesses (Disponible depuis Angular 8) : unique solution supportée 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: '/error' }
];
```

# Angular

Supprimons la base des routes enfants 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 },
  { 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,
    MenuComponent
  ],
  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

© Achref EL MOUADIB

# 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

**Pour créer un module `exercice` avec un chargement `lazy` et lui associer une route dans `app-routing.module.ts`**

```
ng g m modules/exercice --route exercice --module app.module
```

© Achref EL MOUELHI

# Angular

**Pour créer un module `exercice` avec un chargement lazy et lui associer une route dans `app-routing.module.ts`**

```
ng g m modules/exercice --route exercice --module app.module
```

## Remarque

- La commande précédente crée un module `exercice`, un module de routage `exercice-routing.module` et un composant `exercice` dans le module `exercice`.
- Dans `exercice-routing.module`, une route vide sera générée pour le composant `exercice`.
- `--route` permet de créer une route dans `app-routing.module.ts` vers le module `exercice`.

# 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`.
- Supprimez la déclaration des composants déplacés dans `app.module.ts`.
- Définissez des nouvelles routes vers les composants déplacés, ces routes doivent commencer par `/cours`, `/exercice...`
- Au moins un composant de chaque module doit apparaître dans le menu.

# Angular

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

```
const routes: Routes = [
  { path: '', component: HomeComponent },
  {
    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' }
];
```

# Angular

## Question

Comment faire pour utiliser un composant `C` défini dans un module `M1` dans un module `M2`.

© Achref EL MOUELHI ©

# Angular

## Question

Comment faire pour utiliser un composant `C` défini dans un module `M1` dans un module `M2`.

## Hypothèse

Supposons qu'on veut afficher le composant `Camion` dans le module `Exercice`.

# Angular

## Question

Comment faire pour utiliser un composant `C` défini dans un module `M1` dans un module `M2`.

## Hypothèse

Supposons qu'on veut afficher le composant `Camion` dans le module `Exercice`.

**C'est-à-dire, dans `tableau.component.html`, on veut ajouter**

```
<app-camion></app-camion>
```

# Angular

## Solution en deux étapes

- Exporter le composant `Camion` dans le module `Vehicule`.
- Importer le module `Vehicule` dans le module `Exercice`.

© Achref EL M...

# Angular

## Solution en deux étapes

- Exporter le composant `Camion` dans le module `Vehicule`.
- Importer le module `Vehicule` dans le module `Exercice`.

## Remarque

Seuls les composants exportés dans `Vehicule` pourront être utilisés dans `Exercice`.

**Commençons par exporter `CamionComponent` 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
  ],
  exports: [
    CamionComponent
  ]
})
export class VehiculeModule {
  constructor() { console.log('vehicule-module'); }
}
```

# Angular

Importons `VehiculeModule` dans `exercice.module.ts`

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

import { ExerciceRoutingModule } from './exercice-routing.module';
import { TableauComponent } from './tableau/tableau.component';
import { VehiculeModule } from '../vehicule/vehicule.module';

@NgModule({
  declarations: [
    TableauComponent,
  ],
  imports: [
    CommonModule,
    ExerciceRoutingModule,
    VehiculeModule
  ]
})
export class ExerciceModule { }
```

# Angular

## Composant autonome (standalone)

- Composant indépendant de tout module
- Pouvant être utilisé dans plusieurs modules
- Pouvant être chargé dynamiquement

# Angular

Pour créer un composant autonome

```
ng g c components/bouton --standalone
```

© Achref EL MOUETI

# Angular

## Pour créer un composant autonome

```
ng g c components/bouton --standalone
```

## Résultat : le composant n'a pas été déclaré dans un module

```
CREATE src/app/components/bouton/bouton.component.html (22 bytes)
CREATE src/app/components/bouton/bouton.component.spec.ts (619 bytes)
CREATE src/app/components/bouton/bouton.component.ts (246 bytes)
CREATE src/app/components/bouton/bouton.component.css (0 bytes)
```

# Angular

## Contenu de BoutonComponent

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-bouton',
  standalone: true,
  imports: [],
  templateUrl: './bouton.component.html',
  styleUrls: ['./bouton.component.css']
})
export class BoutonComponent {

}
```

# Angular

## Contenu de BoutonComponent

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-bouton',
  standalone: true,
  imports: [],
  templateUrl: './bouton.component.html',
  styleUrls: ['./bouton.component.css']
})
export class BoutonComponent {

}
```

Dans `imports`, on déclare les modules utilisés par le composant autonome.

# Angular

## Un composant autonome peut être chargé statiquement

```
{ path: 'bouton', component: BoutonComponent },
```

© Achref EL MOUELHI

# Angular

## Un composant autonome peut être chargé statiquement

```
{ path: 'bouton', component: BoutonComponent },
```

## Mais aussi dynamiquement

```
{  
  path: 'bouton',  
  loadComponent: () => import('./components/bouton/bouton.component')  
    .then(c => c.BoutonComponent)  
},
```

# Angular

Pour utiliser le composant autonome dans `tableau.component.html`

```
<app-bouton></app-bouton>
```

© Achref EL MOUELHI ©

# Angular

Pour utiliser le composant autonome dans `tableau.component.html`

```
<app-bouton></app-bouton>
```

Pas besoin de l'exporter, mais il faut l'importer comme un module dans `exercice.module.ts`

```
@NgModule({
  declarations: [
    TableauComponent,
  ],
  imports: [
    CommonModule,
    ExerciceRoutingModule,
    VehiculeModule,
    BoutonComponent
  ]
})
export class ExerciceModule { }
```

# Angular

## Déclarons une liste dans BoutonComponent

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-bouton',
  standalone: true,
  imports: [],
  templateUrl: './bouton.component.html',
  styleUrls: ['./bouton.component.css']
})
export class BoutonComponent {
  list = [2, 3, 8, 5]
}
```

# Angular

Pour pouvoir utiliser les directives et les pipes par défaut, il faut importer

CommonModule

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

@Component({
  selector: 'app-bouton',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './bouton.component.html',
  styleUrls: ['./bouton.component.css']
})
export class BoutonComponent {
  list = [2, 3, 8, 5]
}
```

# Angular

Ainsi, nous pourrions utiliser `*ngFor` par exemple

```
<p>bouton works!</p>
<ul>
  <li *ngFor="let item of list">
    {{ item }}
  </li>
</ul>
```

# Angular

Pour utiliser Control Flow Syntax, aucun import n'est nécessaire

```
<p>bouton works!</p>
<ul>
  @for (item of list; track $index) {
    <li>{{ item }} </li>
  }
</ul>
```