

Comprendre les modules avec Angular 16

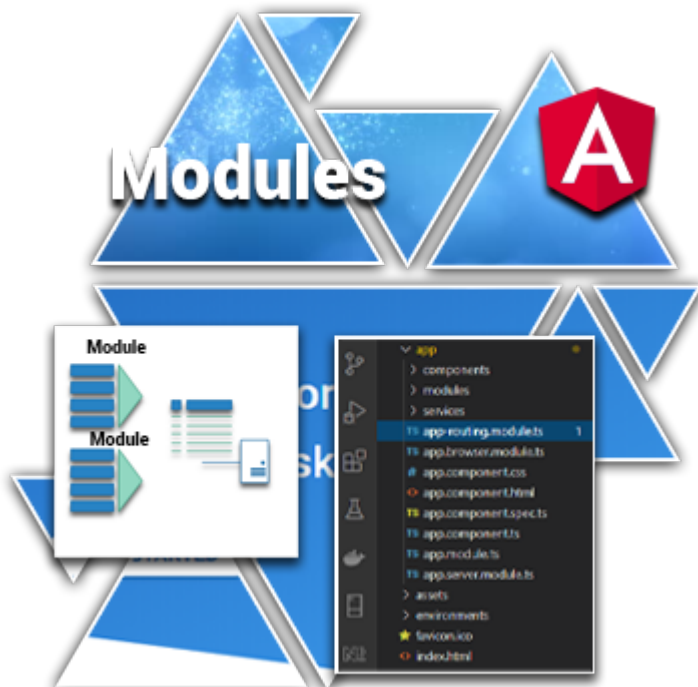
✎ Mis à jour :  Danny

Angular est un **Framework Typescript** qui permet de gérer la notion de **modules** de deux manières.

La première en utilisant les modules **ECMAScript 6** et la seconde en utilisant **NgModules** le système spécifique à Angular.

Ce sont deux fonctionnalités différentes mais complémentaires, qui permettent de mieux structurer le code de nos applications.

Dans ce guide complet nous allons voir comment utiliser les modules avec **Javascript** et **Angular NgModules**.



Ce que nous allons faire

- **Origine des modules**

Pourquoi et comment et les modules ont été intégrés au langage Javascript ?



Comment utiliser les modules avec javascript au travers d'un exemple simple.

- **Angular CLI et NgModules**

Comment utiliser NgModules et intégrer les modules avec Angular.

- **Création d'un projet From scratch**

Nous utilisons Angular CLI pour créer à partir de rien une application de test.

- **Utilisation de notre projet prototype**

Nous utiliserons un projet existant contenant les fonctionnalités essentielles.

Le projet a été généré avec Angular CLI.

Il utilise le Routing et le Lazy Loading.

Il intègre le Framework CSS Bootstrap.

- **Le routage et les modules**

Comment fonctionnent l'interpolation.

- **Le lazyloading et les modules**

Comment gérer l'architecture Angular en utilisant Angular CLI ?

- **Effectuer les Tests**

Nous testerons notre application via les tests unitaires et de bout en bout intégrés dans Angular.

- **Code source**

Le code complet du projet sur github.

Origine des modules

Pour répondre à cette question revenons quelques instants sur l'histoire de l'informatique.

Si vous voulez créer des applications Web, il vous faut utiliser un **ordinateur** .

Si vous voulez **communiquer** avec un ordinateur le plus simple est de parler le **même langage informatique** (ou *Programming Language* en anglais).

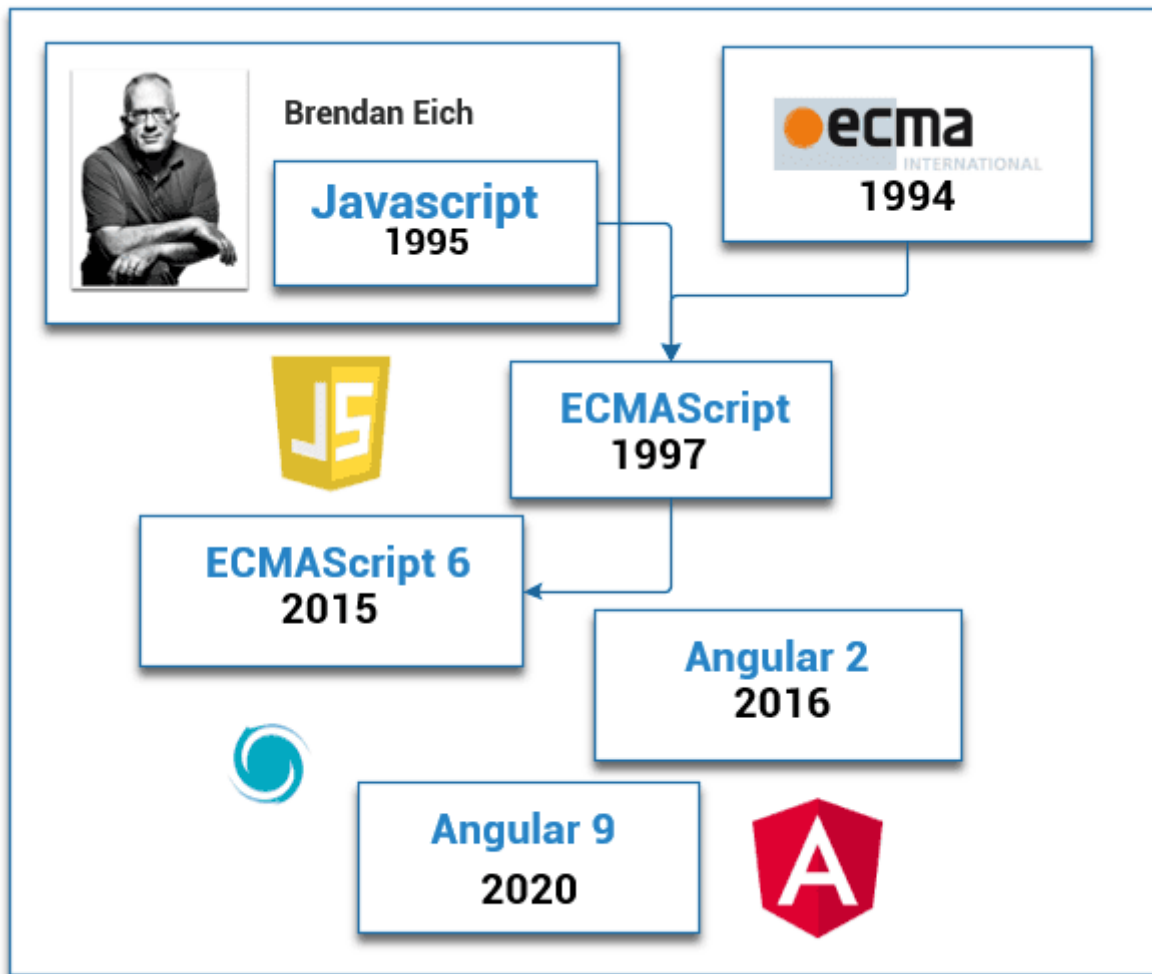
JavaScript est l'un de ces langages.

C'est précisément un **langage de programmation de scripts** .



Un ensemble de normes ont été inventées pour gérer les langages de programmation de type script.

Ce standard porte le nom de **ECMAScript**.



ECMA (pour **E**uropean **C**omputer **M**anufacturers **A**ssociation) est une organisation de normalisation.

Elle a été créée en 1961 pour standardiser les systèmes informatiques en Europe.

Elle porte le nom de **Ecma International** depuis 1994.

Au fil des années différentes **versions ECMAScript** ont été publiées, parmi celles-ci nous citerons périodiquement.

- Version 1 : 1997
- Version 2 : 1998
- **Version 6 : 2015**
- Version 7 : 2016
- Version 10 : 2019

La version qui va nous intéresser plus particulièrement est la **version 6**.

- **ES2015 pour ECMAScript Édition 2015**

La raison de notre intérêt.

L'introduction de nombreux concepts comme les promesses, les classes, les itérateurs, les générateurs.

Et surtout ce qui fait l'objet de ce tutoriel.

Les modules

Pourquoi utiliser des modules ?

Au cours des dix dernières années, les applications Web sont devenues de plus en plus complexes.

Les programmes javascript qui permettent de créer ces applications deviennent ainsi plus complexes et donc plus volumineux.

Pour pallier à ces difficultés, il a fallu créer un mécanisme permettant de diviser les programmes JavaScript en plusieurs parties que l'on appelle **modules** .

Avec les modules cela pourrait se reprendre à ça.



Les avantages sont notamment

- Structurer nos applications (plusieurs fichiers au lieu d'un seul).
- Réutiliser le code.
- Encapsuler le code et faciliter son contrôle.
- Gérer les dépendances plus facilement.

L'utilisation des modules dans JavaScript repose sur deux instructions

- **importer**
- **exporter**

En JavaScript, les modules sont de simples fichiers contenant du code JavaScript.
Pour activer un module s'il suffit d'écrire l'instruction import.

Cette syntaxe des modules ES6 correspond à un standard de construction préconisé dans les spécifications ECMAScript du langage javascript.

Un petit exemple de code ci dessous.

```
# Le code pour exporter un module
export class AppComponent { ... }

# Le code pour importer un module
import { AppComponent } from './app.component';
```



A quoi ressemble un module écrit en javascript ?

Prenons un exemple très simple de programme Javascript.

Pour le faire fonctionner nous avons besoin de deux choses

- Un **éditeur de code** (je vous conseille Visual Studio code)
- La plateforme javascript **Node.js**

Pour installer ces deux outils vous pouvez suivre ce guide

<https://www.ganatan.com/tutorials/demarrer-avec-angular>

Pour l'exercice il suffira de taper le code suivant dans un fichier **app.js**

app.js

```
function movieName(name) {  
  console.log('Movie name :' + name);  
}  
  
function movieDirector(director) {  
  console.log('Movie Director : ' + director);  
}  
  
function app() {  
  movieName('Gladiator');  
  movieDirector('Ridley Scott');  
}  
  
app();
```

Pour l'exécuter comme il s'agit d'un fichier javascript (**extension js**) tapez la commande node suivante

- *noeud app.js*

Maintenant quelques explications.

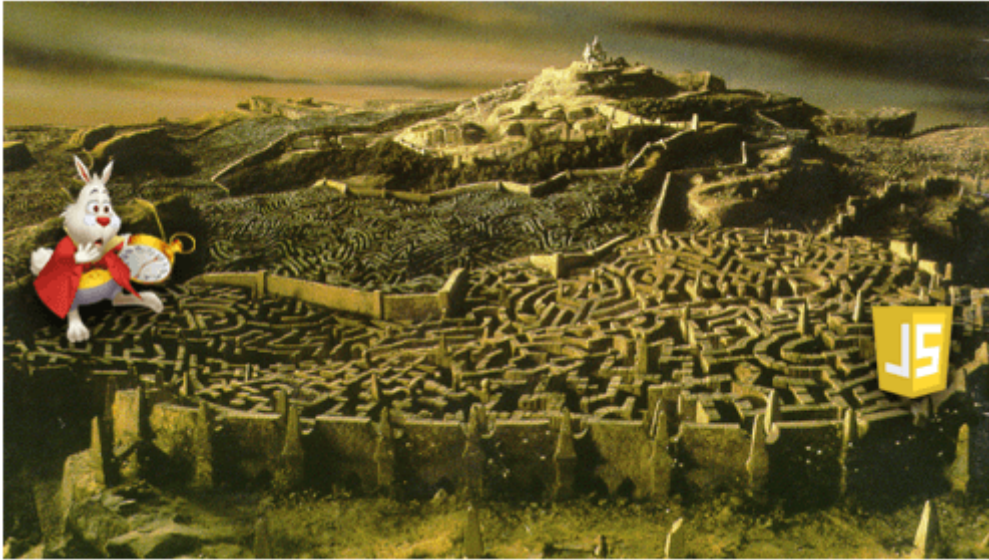
La fonction **app** est appelée lors de l'exécution du script **app.js** .

Cette fonction appelle à son tour les deux fonctions **movieName** et **movieDirector**.

Mais évidemment au fur et à mesure des modifications le nombre de fonctions va probablement augmenter.

De deux fonctions nous définirons nous retrouver à en appeler des centaines.

Et comme toujours le jour où l'on sera pressé ce sera moins facile de retrouver son chemin !



Un exemple javascript avec Modules

Adaptons maintenant notre code pour utiliser ce fameux système de modules.

Pour cela transformons notre fichier app.js en **3 fichiers distincts** .

- **app-modules.js**
- **nom-du-film.js**
- **film-director.js**

app-modules.js

```
import { movieName } from './movie-name.js';
import { movieDirector } from './movie-director.js';

function app() {
  movieName('Gladiator');
  movieDirector('Ridley Scott');
}
```


nom-du-film.js

```
function movieName(name) {  
  console.log('Movie name :' + name);  
}  
  
export { movieName };
```

film-director.js

```
function movieDirector(director) {  
  console.log('Movie Director : ' + director)  
}  
  
export { movieDirector };
```

Comment exécuter un exemple javascript avec Modules ?

Le programme est mieux structuré et il sera plus facile de faire des modifications.

Pour terminer cet exemple, l'idée serait d'exécuter la commande

- `noeud app-modules.js`

Mais comme trop souvent en informatique rien ne marche comme sur l'imaginaire.

La commande précédente donnera l'erreur suivante

- *SyntaxError : Impossible d'utiliser l'instruction d'importation en dehors d'un module*



Pour faire fonctionner notre test on va procéder comme suit

- Créer un fichier **index.html**
- Appeler le script avec l'option **type="module"**
- Et pour peaufiner le tout rajoutons aussi un fichier icon **favicon.ico**

Ce qui donnera le fichier suivant

index.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, user-scalable=no">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <title>Page Title</title>
</head>

<body>
  <h1>Page content</h1>
  <script type="module" src="app-modules.js"></script>
</body>

</html>
```

Pour terminer une dernière erreur.

Si vous lancez index.html dans votre navigateur, vous obtiendrez l'erreur suivante.

L'accès au script sur 'app-modules.js' depuis l'origine 'null' a été bloqué par la politique CORS.



Installons donc un **serveur http**.

Node.js nous permettra de le faire et de tester enfin notre programme javascript.

```
# Installer un serveur http
npm install -g httpserver

# Exécuter le serveur, le fichier index.html est utilisé par défaut
http-server -o

# Tester l'application
http://localhost:8080/
```

Si **Chrome** est votre navigateur utilisez **F12** et l'onglet **console** pour vérifier l'affichage.

- **Nom du film : Gladiateur**
- **Réalisateur : Ridley Scott**

Comme on dit "il faut le voir pour le croire".

Les modules ES6 ça marche.

Angular et ngModules

Que ce soit avec Javascript ou avec Angular la même question se posera à nous.

Plus le développement de notre application avancera, plus les fonctionnalités seront nombreuses, et plus il sera difficile de s'y retrouver dans le code.

Angular en tant que Framework majeur permet un découpage sous forme de modules. L'objectif étant de séparer les différentes fonctionnalités de l'application afin de mieux organiser notre code.

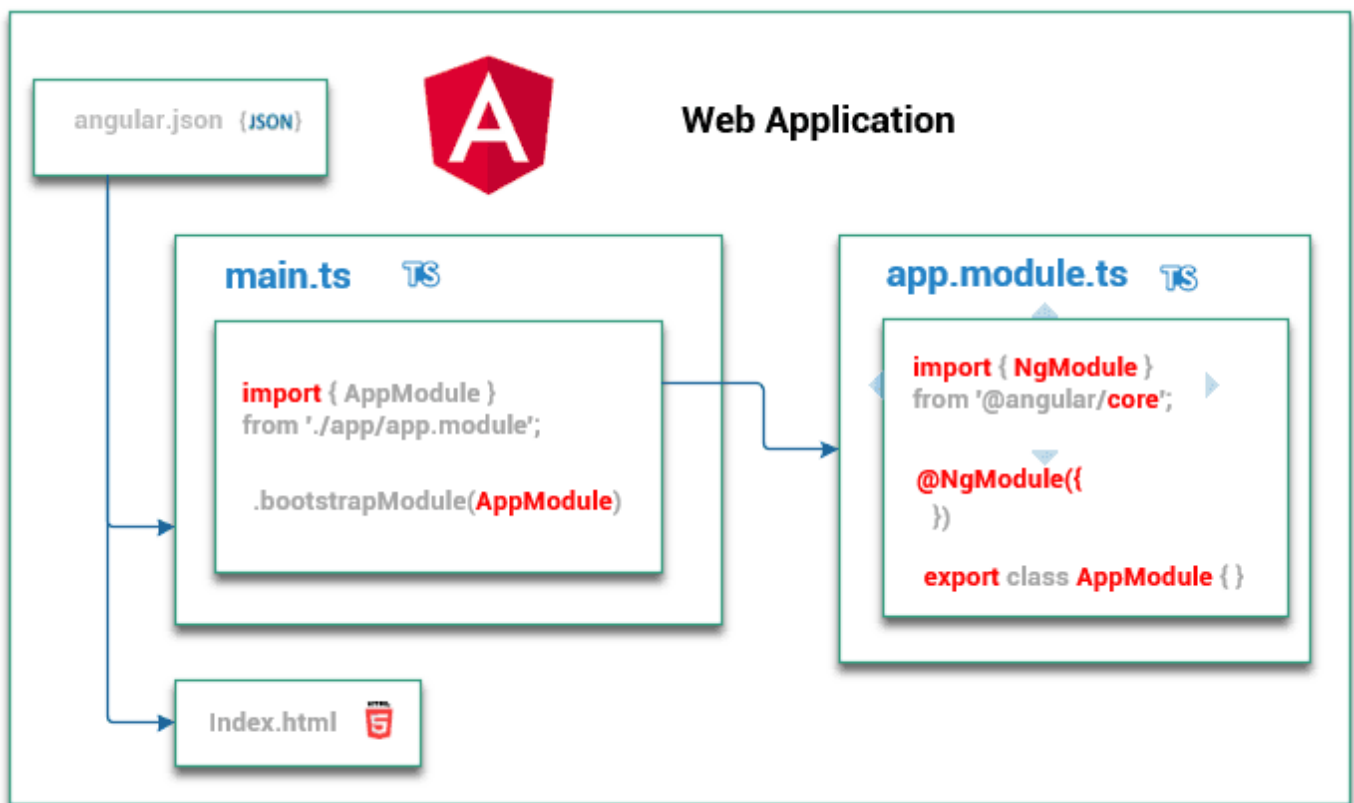
Angular dispose de son propre système de gestion des modules. Il porte le nom de **NgModules**.

Angular repose à la fois sur le système de **modules ES6** ainsi que sur son **propre système NgModules**.

- <https://angular.io/guide/architecture-modules>
- <https://angular.io/guide/ngmodules>
- <https://angular.io/guide/ngmodule-vs-jsmodule>

Angular dispose donc de son propre système de gestion des modules.
Il porte le nom de **NgModules**.

Avant d'aller plus loin une image va nous permettre de décrire une application Angular de base et la notion de modules.



Procédures par étape.

Notre application de base dispose d'un point d'entrée unique.

Le fichier **main.ts**

Ce fichier effectue le lancement ou **le bootstrapping** du module root.

Ce module **root** (traduit en français par racine) porte le nom **AppModule**.

Il est contenu dans le fichier **app.module.ts**

NgModule.

Quelques remarques pour parfaire tout cela

- Chaque application Angular dispose d'un **module au minimum** .
- Le "root module" est un module classique dont la particularité est de définir le "root composant" de l'application via la propriété **bootstrap** .
- NgModule est une interface dactylographiée.

Au final on se retrouve avec le code source suivant

- **app.module.ts**
- **main.ts**

src/app/app.module.ts

```
import { NgModule } from '@angular/core';
@NgModule({
  declarations: [ ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

src/main.ts

```
import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

Le concept des Modules Angulars

Maintenant que nous avons eu un aperçu d'Angular et de ses modules, entrons dans les détails pour mieux comprendre ce concept.

Les modules Angular représentent donc un **concept essentiel** dans le mode de fonctionnement de ce framework.

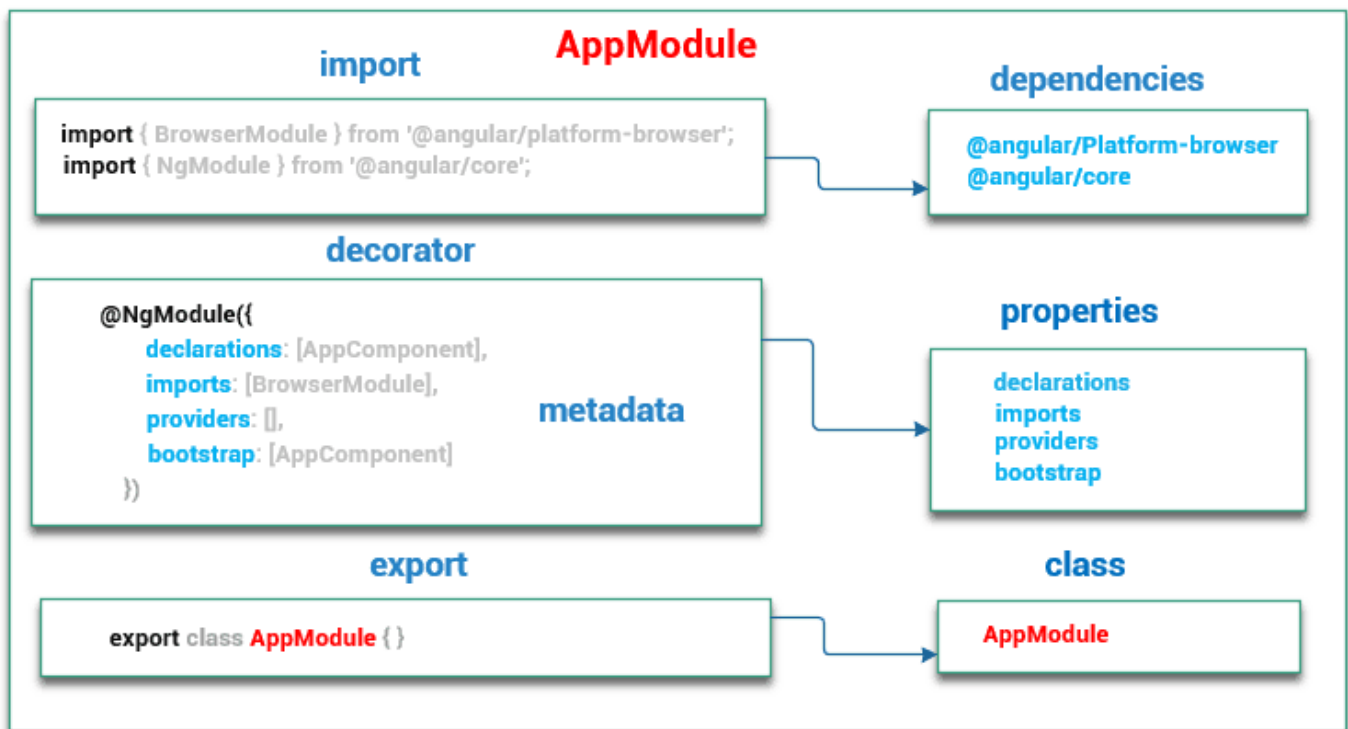
Ils jouent un rôle majeur dans la **structuration** de toutes les applications Angular.

En se lance en parlant **Module Angular**.

- Un Module Angular **définit les dépendances** à d'autres modules nécessaires à son propre fonctionnement.
- Un module Angular est simplement **défini** avec une **classe** et le décorateur **@NgModule** .

Pour s'éclaircir les idées une représentation de notre premier module.

app.module.ts



NgModule et décorateur

Un NgModule est une classe décrite par un décoré avec `@NgModule()`.

`@NgModule()` est une fonction qui dispose d'un objet métadonnées.

Ces propriétés possèdent le module

Le **décorateur** (ou **decorator**) **@NgModule** permettra d'attacher dynamiquement de nouvelles responsabilités à cette classe.

Il modifie la classe javascript en fournissant des **méta-données** (ou **métadonnées**) pour composer de nouvelles fonctionnalités.



Ce metadonnees dispose de **proprietes** (properties en anglais).

Ci-dessous une liste non exhaustive de ses propriétés.

- **déclarations**

Définit la liste des éléments appartenant à ce module.

Ce seront par exemple les directives, pipes, composants

- **exports**

Définit la liste des composants qui seront visibles et pourront donc être utilisés par les modules qui importent le module.

Les composants non déclarés seront donc non-exportés et ne pourront être utilisés que par les composants contenus dans le module.

- **importations**

Définit la liste des dépendances du module, c'est à dire les modules dont dépendent notre module.

- **Providers**

Cette propriété permet de déclarer les services que vous allez créer dans le cadre de ce module.

- **bootstrap**

Définit le composant root qui contient l'ensemble des autres composants de votre application.

Seul le module root peut déclarer cette propriété.

exports : le sous-ensemble de déclarations qui doivent être visibles et utilisables dans les modèles de composants d'autres NgModules.

importations : autres modules dont les classes exportées sont nécessaires aux modèles de composants déclarés dans ce NgModule.

fournisseurs : créateurs de services que ce NgModule contribue à la collection mondiale de services ; ils deviennent accessibles dans toutes les parties de l'application. (Vous pouvez également spécifier des fournisseurs au niveau des composants, ce qui est souvent préféré.)

bootstrap : la vue principale de l'application, appelée composant racine, qui héberge toutes les autres vues de l'application. Seule la racine

Module partagé



Elle permet de **rassembler** les suites d'instructions identiques dispersées dans un programme en une seule fonction.

Elle **améliore** ainsi la lisibilité du code et en **facilite** la correction et les modifications ultérieures.

Angular va nous permettre d'appliquer la factorisation en créant des **modules partagés** (modules partagés)

Les conseils de l'équipe Angular sont à l'adresse suivante

<https://angular.io/guide/sharing-ngmodules>

Les modules partagés vont nous permettre de mettre en commun des directives, des tuyaux et des composants dans un seul module.

Il suffira ensuite d'importer ce module dans n'importe quelle partie de notre application ou nous en aurons besoin.

Angular CLI et les modules

Angular CLI signifie **Angular Command Line Interface**.

Angular CLI est probablement l'outil le plus important fourni par le Framework angulaire.

Il vous permettra d'initialiser, de développer et de maintenir vos applications Angular.

Les commandes fournies par cet outil **gèrent automatiquement** les déclarations, les importations, ou le bootstrap.

Angular CLI rajoute donc implicitement la gestion des modules lors de la création de votre code.

Il nous reste maintenant à expérimenter les modules.

Comme on va pas réinventer la roue à chaque fois

Nous utiliserons évidemment Angular CLI il a été fait pour ça.

Au fur et à mesure des exercices nous verrons comment les principes des modules sont mis en œuvre.

Ce tutoriel vous propose deux cas de figure

- Une application de base créée à partir de rien (**from scratch**)
- Un prototype d'application basé sur **Angular CLI** , le **Routing** , le **Lazyloading** et le Framework **Bootstrap**

Choisissez vos armes et a vos postes de combat.



Candidature à partir de zéro

Nous allons donc créer notre application Angular de base.

From Scratch signifiant que nous la créerons à partir de rien sans code source extérieur.
On ne peut pas faire plus formateur !

Comme nous l'avons vu auparavant nous n'aurons besoin que de deux outils

- **Noeud.js**
- Un **éditeur de code**

Il ne nous reste plus qu'à exécuter les commandes suivantes.

```
# Installation d'angular-cli dernière version disponible
npm install -g @angular/cli

# Test de version installée
ng --version

# Générer un projet appelé arbitrairement angular-starter avec options par d
ng new angular-starter --defaults

# Se positionner dans le projet
```



```
# Executer  
ng serve
```

C'est pour cela que l'on peut qualifier Angular de **Framework** .

Quelques lignes de commande et l'application apparaissent par magie.
C'est du " **tout en un** ". Et de qualité de surcroît !

Analyses maintenant le code source généré par Angular CLI.

Nous essayons de reprendre tous les éléments liés aux modules que nous avons évoqués jusqu'ici.

Commençons par le fichier **app.module.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 { }
```

Décrivons étape par étape ce qu'a fait angular CLI.

Comment créer un module ?

- **export class AppModule**
 - Le module est une **classe**
 - Il porte le nom **AppModule**

Comment décrire les caractéristiques de ce module ?

- **@NgModule** ({ })

L'élément essentiel est le **décorateur** .

C'est le caractère **@**qui symbolise le décorateur.

Le décorateur est un **patron de conception** (design pattern en anglais).

Il **décrit** les grandes lignes d'une solution.

Prototype d'application

Utilisons maintenant une application plus étoffée contenant un certain nombre de fonctionnalités.

Pour pouvoir continuer ce tutoriel il nous faut rajouter un outil

- **Git** : Le logiciel de gestion de versions.

Nous allons utiliser un projet existant dont les caractéristiques sont

- Généré avec **Angular CLI**
- **Routage**
- **Chargement paresseux**
- Utilisation du Framework CSS **Bootstrap**

Les commandes à utiliser sont les suivantes

```
# Créez un répertoire demo (le nom est ici arbitraire)
mkdir demo

# Allez dans ce répertoire
cd demo

# Récupérez le code source sur votre poste de travail
git clone https://github.com/ganatan/angular-react-bootstrap

# Allez dans le répertoire qui a été créé
cd angular-react-bootstrap
cd angular

# Exécutez l'installation des dépendances (ou librairies)
npm install
```



```
# Vérifiez son fonctionnement en lançant dans votre navigateur la commande  
http://localhost:4200/
```

Essais

Il ne reste plus qu'à tester les différents scripts Angular.

```
# Développement  
npm run start  
http://localhost:4200/  
  
# Tests  
npm run lint  
npm run test  
  
# Compilation  
npm run build
```

Source du code

En suivant chacun des conseils que je vous ai donnés dans ce guide vous obtenez au final un code source Angular.

Le code source obtenu à **la fin de ce tutoriel** est disponible sur github

<https://github.com/ganatan/angular-react-modules>

Les étapes suivantes vous permettront **d'obtenir un prototype d'application** .

- [Etape 6 : Server Side Rendering avec angulaire](#)
- [Etape 7 : Progressive Web App avec Angular](#)
- [Etape 8 : Optimisation des moteurs de recherche avec Angular](#)



Les étapes suivantes vous permettront d'améliorer ce prototype

- [Composants avec Angular](#)
- [Services avec Angular](#)
- [Formulaires pilotés par modèles avec Angular](#)
- [Graphiques avec Angular](#)

Cette

dernière étape permet **d'obtenir un exemple d'application**

- [Créer une application Web complète avec Angular](#)

Le **code source de cette application finale** est disponible sur GitHub


<https://github.com/ganatan/angular-app>


Comment créer une application From scratch ?


Créez votre compte ganatan


Téléchargez gratuitement vos guides complets


Démarrez avec la CLI angulaire 

Gérez le routage 

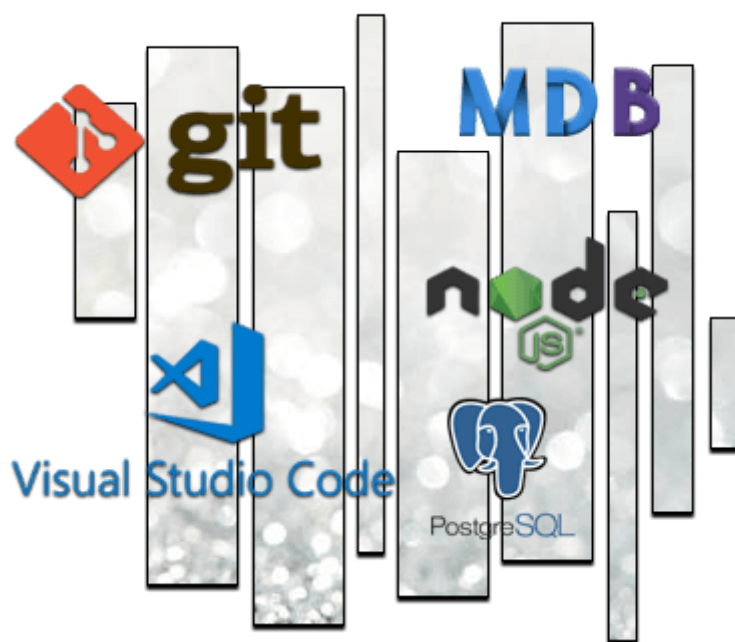
Appliquez le Lazy chargement 

Intégrez Bootstrap 

Utiliser Python avec Angular 

Utiliser Django avec Angular 

Utiliser Flask avec Angular 



Tutoriels Backend



[Python avec angulaire](#)



[Django avec angulaire](#)



[Flacon avec angulaire](#)

Tutoriels Frontend



[Démarrer avec React](#)



[Routage avec React](#)



Application Web : Angular 16, Bootstrap 5

Chargement paresseux, SSR, PWA, SEO

in



OUTILS

Angulaire

Réagir

Amorcer

Police géniale

Noeud.js

Exprimer

PostgreSQL

MySQL

MongoDB

nginx

LIENS UTILES

Tutoriels

À propos de

www.ganatan.com