

Formation Angular



Angular

- Framework basé sur Typescript, CSS et HTML, développé par Google
- Permet la création d'applications web monopage (SPA)
- Successeur de AngularJS, en 2016

Versions

- Angular 14 (02/06/2022)
 - Composants standalone
 - Liaison sur les attributs protected
 - inject()
- Angular 15 (16/11/2022)
 - host directive
 - Intercepteur fonctionnel
- Angular 16 (03/05/2023)
 - Signals
 - @Input requis
 - takeUntilDestroyed()
- Angular 17 (08/11/2023)
 - Blocs @if @for et @switch directement dans le template

Etude de l'application de démonstration d'Angular

- Installation d'angular

```
npm install -g @angular/cli
```

- Crédit du projet

```
ng new
```

- Le premier exemple est basé sur le projet qui était initialisé avec la version 16 d'Angular

Lancement de l'application

```
ng serve
```

- ng serve permet de tester l'application sur votre machine
- Le lancement de l'application se découpe en deux parties :
 - Le compilateur transforme votre projet Angular en fichiers HTML/CSS et javascript
 - Un serveur local est créé en sur la machine, qui héberge ces fichiers (avec Vite depuis la version 17, Webpack avant)

main.ts

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { InterpolationComponent } from './01-interpolation/interpolation.component';
import { AppModule } from './app/app.module';

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

/* ou */

// Composant standalone
bootstrapApplication(InterpolationComponent).catch(err => console.error(err));
```

- Point d'entrée de l'application
- Permet d'initialiser l'application avec un module (en haut) ou un composant standalone (en bas)
- Dans une application normale, ce fichier est très rarement modifié

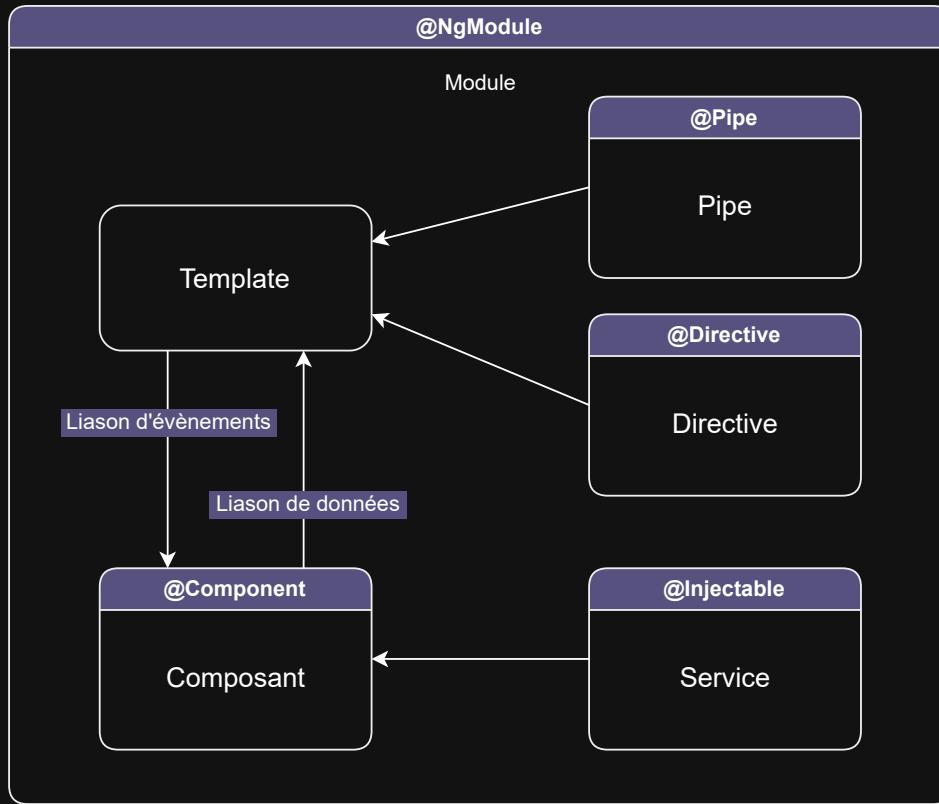
index.html

Comparaison du index.html du projet Angular et du code source du site web généré

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>FormationApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

```
<html lang="en">
  <head>
    <script type="module" src="/@vite/client"></script>
    <meta charset="utf-8">
    <title>FormationApp</title>
    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <link rel="stylesheet" href="styles.css">
    <style> @</style>
  </head>
  <body>
    <app-root _ngcontent-ng-c1593039801 ng-version="17.3.5">
      <div _ngcontent-ng-c1593039801 class="content" flex>
        <h1 _ngcontent-ng-c1593039801>Torticols</h1>
        <h3 _ngcontent-ng-c1593039801>Liste des colis</h3>
        <input _ngcontent-ng-c1593039801 type="hidden">
        <div _ngcontent-ng-c1593039801 class="card-container" flex> @</div>
        <h3 _ngcontent-ng-c1593039801>Details</h3>
        <div _ngcontent-ng-c1593039801 class="box" ng-reflect-ng-switch="true" flex> @</div>
      </div>
    </app-root>
    <script src="polyfills.js" type="module"></script>
    <script src="main.js" type="module"></script>
  </body>
</html>
```

Classes Angular



Modules

- Un module est un regroupement des autres classes Angular
 - Les composants, directives et pipes sont déclarés dans des modules
 - Les modules peuvent exporter une partie de leurs déclarations pour les autres modules
 - Chaque application Angular possède au moins un module, le module racine, utilisé par Angular pour lancer l'application
-
- Depuis Angular 14 et les composants standalones, il est possible de faire une application entièrement sans modules
 - A partir de la version 17, les composants sont standalone par défaut

app.module.ts

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

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

app.module.ts

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

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

Décorateur @NgModule

- Indique à Angular que la classe AppModule est un module
- Utilisation de métadonnées pour paramétrier le module

app.module.ts

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

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

Déclarations

- Déclaration de tous les composants, directives et pipes du module
- Les éléments déclarés sont ceux qui appartiennent à ce module, et non ceux exportés

app.module.ts

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

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

Modules importés

- Le module BrowserModule contient les services essentiels pour que l'application fonctionne sur navigateur
- Réexporte le module CommonModule, qui contient les directives et pipes basiques de Angular
- Il est également possible d'importer des composants standalone

app.module.ts

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

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

Composant racine de l'application

- Au démarrage de l'application, Angular recherche dans le DOM, le premier élément correspondant au sélecteur du composant racine et l'injecte à cet endroit

Composants

- Brique de base d'une application Angular
- A chaque composant est assigné un template

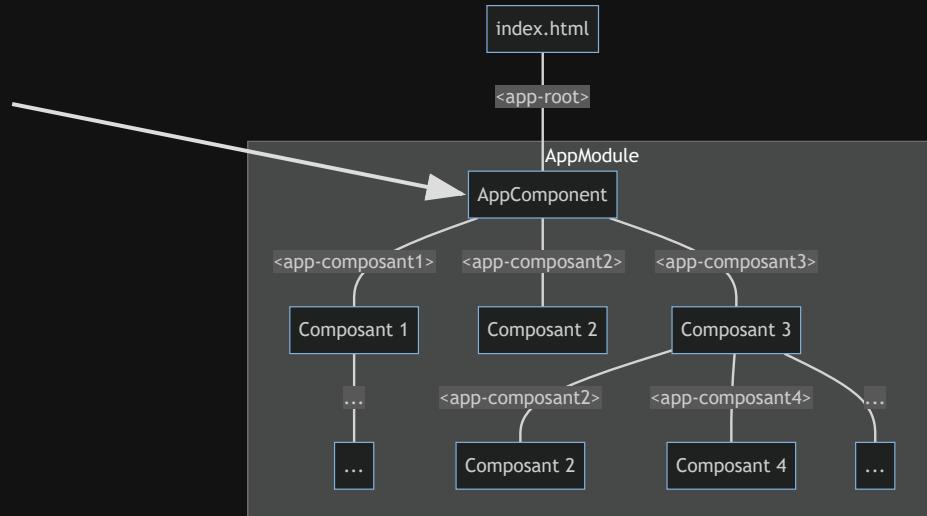
Composant

- Fichier Typescript
- Détermine le comportement du template

Template

- Fichier HTML avec des additions propres à Angular
- Peut avoir son propre fichier de style
- Informe le composant des actions utilisateur

Découpage en composants



- Une application Angular se présente sous la forme d'une arborescence de composants, et contient au moins un composant, le composant racine
- Chaque composant est assigné à un selecteur
- A chaque fois que le selecteur apparaît dans un template, Angular crée une nouvelle instance du composant et l'injecte à cet endroit

Exemple

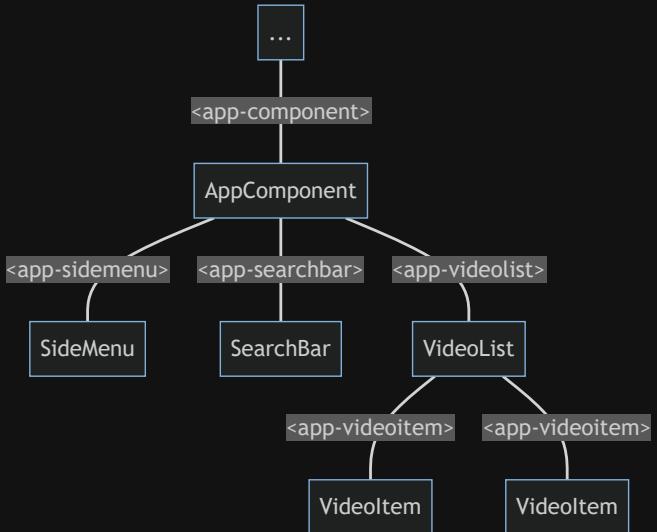
AppComponent

The screenshot shows the YouTube mobile application interface. On the left is a vertical sidebar with a red border containing a navigation menu. At the top right is a search bar with a magnifying glass icon. The main content area displays a grid of video thumbnails. Some thumbnails are highlighted with red boxes, indicating specific components:

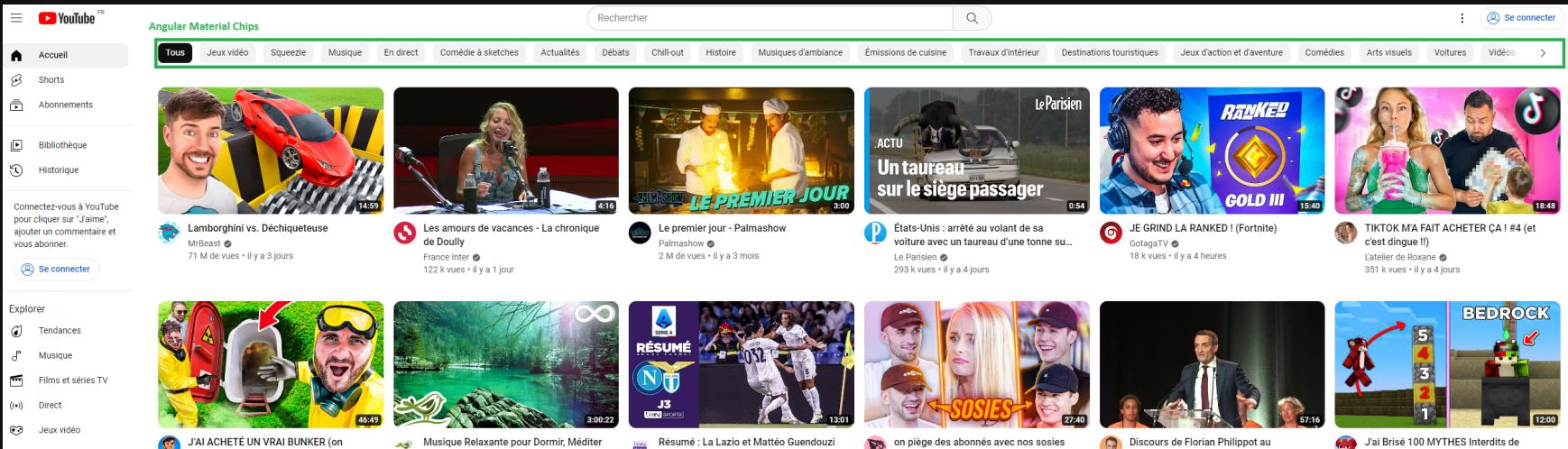
- SideMenu**: The sidebar on the left.
- SearchBar**: The search bar at the top right.
- VideoList**: A row of four video thumbnails at the top.
- Videoltem**: Two video thumbnails in the middle row.
- Videoitem**: Two video thumbnails in the middle row.
- VideoList**: A row of four video thumbnails at the bottom.
- Videoltem**: Two video thumbnails in the bottom row.
- Videoltem**: One video thumbnail in the bottom row.

Below the sidebar, there is a message: "Connectez-vous à YouTube pour cliquer sur 'J'aime', ajouter un commentaire et vous abonner." There is also a "Se connecter" button.

The sidebar menu includes sections like "Tendances", "Musique", "Films et séries TV", "Actualités", "Sport", "Savoirs & Cultures", "Mode et beauté", "Autres contenus YouTube", "YouTube Premium", "YouTube Music", "YouTube Kids", and "Paramètres".



Utilisation de bibliothèques



- La bibliothèque Angular Materials fournit des composants réutilisables
- La SNCF possède sa propre bibliothèque de composants : WCS

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Torticolis';
}
```

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Torticolis';
}
```

Décorateur @Component

- Indique à Angular que la classe AppComponent est un composant
- Utilisation de métadonnées pour paramétriser le composant

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Torticolis';
}
```

Sélecteur CSS du composant

Exemple dans index.html

```
...
<body>
  <app-root></app-root>
</body>
...
```

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Torticolis';
}
```

Template HTML associé au composant :

- Sous la forme d'un fichier (comme dans cet exemple)
- Sous forme de texte, dans ce cas utiliser l'attribut template à la place

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Torticolis';
}
```

Fichier de style utilisé :

- Comme pour le template, soit texte soit fichier
- style et styleUrls sont des tableaux, donc plusieurs entrées possibles
- Par défaut le style est propre au composant
- Un fichier styles.css est présent à la racine pour les styles globaux au projet

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Torticolis';
}
```

Code du composant

app.component.html

```
<div class="content">

<h1>{{ title }}</h1>

<!-- Next Steps -->
<h3>Liste des colis</h3>

<input type="hidden" #selection>

<div class="card-container">
  <button class="card" (click)="selection.value = 'D1AB7E3C-5D23-D646-4375-9B7872B52289'" tabindex="0">
    Lampe
  </button>

  <button class="card" (click)="selection.value = 'C9CE1525-59A7-577D-EE9E-B29065C0EB63'" tabindex="0">
    Chaise
  </button>

  ...

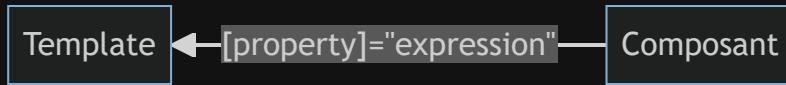
```

Interpolation



- Dans l'exemple précédent, `{{ title }}` est remplacé par Torticolis
- Il y a l'intérieur de l'interpolation une expression de template qui est évaluée et convertie en string
- Il est possible d'accéder aux propriétés et méthodes du composant dans l'expression
- **⚠ Les méthodes dans les expressions doivent être simples, prévisibles et sans effet de bord**
- **⚠ Aux interpolations avec les tableaux et objets**

Property binding



- La syntaxe [] indique à Angular de lier une expression avec une propriété d'un élément du template
- Le lien se fait uniquement du composant vers le template
- Si la propriété du composant change, le template sera modifié en conséquence
- ⚠ Mais pas l'inverse

Template

```
...  
<img [src]="image" />  
...
```



Composant

```
...  
image = 'https://t.ly/B0f_Z'  
...
```

Attribute binding

- **⚠️** Les attributs HTML sont différents des propriétés du DOM
- Il est possible de lier à un attribut, en préfixant avec "attr"
- Exemple :

```
<input [disabled]="condition">  
<input [attr.disabled]="condition ? 'disabled' : null">
```

- Les deux input ont le même comportement (disabled si condition est vraie)
- **⚠️** La propriété disabled attend un booléen, alors que l'attribut disabled attend un string
- **⚠️** Tous les attributs n'ont pas de propriété correspondante, et inversement
- **⚠️** Il peut y avoir des différences de comportement entre attribut et propriété (value d'un input)
- Privilégiez l'utilisation des propriétés