# Différence entre AngularJS et Angular

AngularJS (2010) : basé sur JavaScript, architecture MVC, adapté aux petits projets ou applications existantes.

Angular (2016+) : réécriture complète en TypeScript, architecture basée sur les composants, meilleures performances, conçu pour les applications modernes et mobiles, supporte l’injection de dépendances avancée et des outils modernes.

# Nouveautés depuis Angular 14, confirmées en Angular 19

Angular 14 : introduction des standalone components (composants autonomes), permettant de créer des composants sans module, simplifiant la structure des projets.

Angular 19 :

Les standalone components deviennent le standard.

Nouvelle API de ressources basée sur les signaux (signal-based resource API) pour une gestion plus réactive des données.

Amélioration du rendu côté serveur (SSR) avec configuration avancée des routes serveur.

Transitions de vues plus fluides entre les routes.

# Étapes pour installer et configurer Angular sur son PC

Installer Node.js (version LTS) et npm.

Installer Angular CLI globalement:

*npm install -g @angular/cli*

Créer un nouveau projet Angular :

*ng new mon-projet*

*cd mon-projet*

Lancer le serveur de développement :

*ng serve*

Accéder à l’application : ouvrir le navigateur sur http://localhost:4200.

# Qu’est-ce qu’un composant Angular ?

Un composant Angular est un bloc de base de l’application, représentant une partie de l’interface utilisateur (UI). Il regroupe le template HTML, le style CSS et la logique TypeScript pour gérer l’affichage et le comportement de cette partie de l’UI.

# Qu’est-ce qu’une directive Angular ? Exemples et utilité

Une directive Angular est une instruction ajoutée à un élément du DOM pour modifier son apparence ou son comportement.

Exemples :

*\*ngFor : répète un élément pour chaque item d’une liste.*

*\*ngIf : affiche ou masque un élément selon une condition.*

*[ngClass] : ajoute ou retire des classes CSS dynamiquement.*

*[ngStyle] : applique des styles CSS dynamiquement.*

Permet de rendre le HTML dynamique et interactif sans écrire de JavaScript pur.

# Qu’est-ce qu’un service Angular ?

Un service Angular est une classe TypeScript destinée à centraliser des fonctionnalités réutilisables (appels API, gestion de données, logique métier, etc.). Les services sont généralement des singletons et sont injectés dans les composants via l’injection de dépendances.

# Que fait la fonction ngOnInit ?

ngOnInit est une méthode de cycle de vie appelée automatiquement après l’initialisation d’un composant. Elle sert à effectuer les initialisations (chargement de données, abonnement à des observables, etc.) une fois que les propriétés d’entrée du composant sont prêtes.

# Fichiers principaux générés dans un projet Angular

src/ : dossier principal contenant le code source de l’application (composants, services, modules…).

app/ : sous-dossier de src/ qui contient les composants et la logique métier.

app.component.ts : logique du composant principal.

app.component.html : template HTML du composant principal.

app.component.css : styles CSS du composant principal.

app.module.ts : module principal de l’application.

app-routing.module.ts : configuration du routage (s’il existe).

assets/ : ressources statiques (images, etc.).

environments/ : fichiers de configuration par environnement.

angular.json : configuration du projet Angular.

package.json : dépendances npm et scripts.

tsconfig.json : configuration TypeScript.

# Le mécanisme de routage en Angular

Le routage Angular permet de naviguer entre différentes vues (composants) selon l’URL. Il simule le comportement d’une application multipage dans une SPA (Single Page Application).

Mise en œuvre :

1. Importer le module RouterModule dans app.module.ts :

*import { RouterModule, Routes } from '@angular/router';*

1. Déclarer les routes dans un tableau :

*const routes: Routes = [*

*{ path: 'accueil', component: AccueilComponent },*

*{ path: 'profil', component: ProfilComponent },*

*{ path: '', redirectTo: '/accueil', pathMatch: 'full' },*

*{ path: '\*\*', component: NotFoundComponent }*

*];*

1. Ajouter RouterModule dans les imports :

*imports: [RouterModule.forRoot(routes)]*

1. Utiliser <router-outlet> dans le template pour afficher les composants selon la route.
2. Utiliser [routerLink] dans le HTML pour créer des liens de navigation.

# C’est quoi RxJs ?

**RxJs** (Reactive Extensions for JavaScript) est une bibliothèque permettant de gérer des flux de données asynchrones et d’événements. Elle est utilisée dans Angular pour gérer les requêtes HTTP, les événements utilisateurs, les formulaires réactifs, etc. RxJs repose sur le concept d’**Observable** pour représenter et manipuler ces flux.

# Observable : notion et utilisation

Un **Observable** est un flux de données auquel on peut s’abonner pour recevoir des valeurs au fil du temps (ex : clics, réponses HTTP).

* **Créer un Observable** :

*import { Observable } from 'rxjs';*

*const obs = new Observable(observer => {*

*observer.next('valeur');*

*observer.complete();*

*});*

* **S’abonner à un Observable** :

*obs.subscribe({*

*next: (val) => console.log(val),*

*error: (err) => console.error(err),*

*complete: () => console.log('Terminé')*

*});*

* **Dans les templates Angular**, on utilise souvent le pipe async pour s’abonner automatiquement :

*<div>{{ monObservable$ | async }}</div>*

L’Observable permet de réagir à chaque nouvelle valeur émise, de gérer les erreurs et la fin du flux.

# Différence entre Subject et BehaviourSubject

* **Subject** : émet des valeurs aux abonnés. Si un abonné s’inscrit après une émission, il ne reçoit pas les anciennes valeurs.
* **BehaviourSubject** : nécessite une valeur initiale et émet toujours la dernière valeur à tout nouvel abonné, même s’il s’abonne après l’émission.

**Résumé :**

# À quoi sert le fichier angular.json ?

Le fichier angular.json configure le comportement global du projet Angular (chemins, options de build, scripts, assets, environnements, configuration des applications et librairies du workspace, etc.). Il permet d’ajuster la compilation, le déploiement et le test de l’application.

# Explication des commandes

|  |  |
| --- | --- |
| **Commande** | **Utilité** |
| ng serve | Lance le serveur de développement local sur le port 4200 par défaut. |
| ng serve --port 4500 | Lance le serveur sur le port 4500. |
| ng g c moncomponent | Génère un nouveau composant nommé moncomponent. |
| ng g class MaClasse | Génère une classe TypeScript nommée MaClasse. |
| ng g service MonService | Génère un service Angular nommé MonService. |
| ng g guard AuthGuard | Génère un guard (protection de route) nommé AuthGuard. |
| ng new GestionVols | Crée un nouveau projet Angular nommé GestionVols. |
| json-server --watch produit.json --port 3500 | Lance un serveur REST simulé basé sur le fichier produit.json sur le port 3500. |

# Comment appelle -t -on cet élément @component et quels sont ces attributs

L’élément **@Component** est appelé **décorateur de composant** dans Angular. Il sert à indiquer qu’une classe TypeScript est un composant Angular et à fournir les informations nécessaires à Angular pour l’utiliser dans l’application.

**Attributs principaux du décorateur @Component**

* **selector** : définit le nom de la balise HTML à utiliser pour intégrer ce composant dans un template (ex : 'app-smartphone').
* **templateUrl** : chemin vers le fichier HTML du composant.
* **styleUrls** : chemin(s) vers les fichiers CSS du composant.
* **template** : (optionnel) code HTML directement dans le décorateur (au lieu d’un fichier séparé).
* **styles** : (optionnel) styles CSS directement dans le décorateur.

Exemple :

*@Component({*

*selector: 'app-smartphone',*

*templateUrl: './smartphone.component.html',*

*styleUrls: ['./smartphone.component.css']*

*})*

***export******class*** *SmartphoneComponent { }*

# Mécanisme composant parent / composant enfant

Dans Angular, un composant parent peut contenir un ou plusieurs composants enfants. Cette relation permet de structurer l’interface en blocs réutilisables et d’organiser la communication entre différentes parties de l’application.

**Communication du parent vers l’enfant**

* Le parent transmet des données à l’enfant via des **propriétés décorées avec @Input** dans le composant enfant.
* Exemple :  
  Dans le parent :

*<app-enfant [donnee]="valeur"></app-enfant>*

Dans l’enfant :

*@Input() donnee: string;*

**Communication de l’enfant vers le parent**

* L’enfant peut notifier le parent d’un événement via des **propriétés décorées avec @Output** et la classe EventEmitter.
* Exemple :  
  Dans l’enfant :

*@Output() action =* ***new*** *EventEmitter<string>();*

Dans le parent :

*<app-enfant (action)="methodeParent($event)"></app-enfant>*

Ce mécanisme permet de faire circuler l’information dans les deux sens et d’organiser l’application de façon modulaire et claire