esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

Introduction à

- Introduction générale
- Introduction à Angular
- TypeScript
- Installation et mise en place de l’environnement
- Arborescence d’un projet Angular classique vs standalone
- Architecture d’un projet Angular classique (non-standalone)

Introduction générale

Introduction générale

- Le développement front-end concerne la partie visible et interactive d'une application web : ce que l'utilisateur voit et avec quoi il interagit. Il regroupe l'utilisation de technologies comme HTML, CSS et JavaScript, combinées à des outils modernes pour construire des interfaces riches, dynamiques et réactives.
- Avec l'avènement des Single Page Applications (SPA), le rôle du front-end s'est transformé : il ne s'agit plus seulement d'afficher du contenu, mais aussi de gérer la navigation, l'état de l'application, les interactions en temps réel, et la consommation d'API distantes.

Introduction générale

- Les frameworks front-end, comme Angular, ont été conçus pour répondre à ces nouveaux besoins, en proposant des architectures solides, des composants réutilisables, et des outils dédiés à la productivité des développeurs.

Du monolytique vers la séparation

- Migrer d'une architecture **monolithique** vers une séparation **frontend/backend** (ou vers une architecture en microservices) est souvent motivé par des besoins de **scalabilité, maintenabilité et agilité**.
- Une application **monolithique** se caractérise que tout est dans un seul bloc de code (UI, logique métier, accès DB).
- Une application **séparée** en frontend et backend se caractérise que chaque couche est indépendante (frontend + backend)

Monolytique vs séparation frontend/backend

	Monolitique	**Architecture séparée en frontend et backend**
Déploiement	Déploiement lent	Déploiement indépendant par couche
Scalabilité	Difficulté à scaler	Mise à l'échelle par composant
Maintenabilité	Code difficile à maintenir	Découplage des responsabilités
Régression	Risques de régressions	Moins d'impacts croisés

Développement front-end

Voici quelques-unes des principales technologies utilisées côté client :

- **HTML** : structure du contenu
- **CSS / SCSS / Tailwind** : mise en forme et design
- **JavaScript** : logique et interactions dynamiques
- **TypeScript** : sur-ensemble de JavaScript, typé et plus structuré
- **Frameworks et bibliothèques** :
  - **Angular**
  - **React**
  - **Vue.js**
  - **Svelte**
- **Outils de build** : Webpack, Vite, Parcel
- **Gestionnaires d'état** : NgRx (Angular), Redux (React), Pinia (Vue)

Développement back-end

Et côté serveur, pour gérer la logique métier, les bases de données et les API :

- **Langages** :
  - **JavaScript / TypeScript** avec Node.js
  - **Python** avec Django ou Flask
  - **PHP** avec Laravel ou Symfony
  - **Java** avec Spring Boot
  - **C#** avec ASP.NET Core
- **Bases de données** :
  - **Relationnelles** : MySQL, PostgreSQL
  - **NoSQL** : MongoDB, Firebase
- **APIs** : REST, GraphQL
- **Serveurs** : Express.js, Koa, FastAPI, etc.
- **Authentification** : JWT, OAuth2, Auth0

Introduction à Angular

Angular - Définition

- Angular est un **framework front-end,** open-source, développé par Google, conçu pour créer des applications web dynamiques, performantes et évolutives.

- Il permet aux développeurs de structurer, organiser et automatiser le développement d'applications complexes grâce à une architecture basée sur : les composants, les modules, l'injection de dépendances, le data binding (liaison de données), un système de routage intégré et un outillage CLI (Command Line Interface) puissant.

Angular - caractéristiques

Caractéristique	Détails
Langage principal	TypeScript (sur-ensemble de JavaScript)
Type d'application	SPA (Single Page Application)
Développé par	Google
Structure	Basée sur des composants et des modules
Support natif	Tests, formulaires, animations, SSR, PWA
Outils intégrés	Angular CLI, Angular Universal, Angular Material

Evolution du framework v2 – v18

- **Angular 2 (2016)** : réécriture complète d'AngularJS, introduction des composants, TypeScript, DI moderne
- **Angular 4-5 (2017)** : amélioration des performances, Angular Universal (SSR), animation module
- **Angular 6-7 (2018)** : introduction d'Angular CLI v6, RxJS 6, Angular Elements
- **Angular 8-9 (2019-2020)** : Ivy Compiler, differential loading
- **Angular 10-11 (2020)** : mise à jour des dépendances, amélioration du type-checking
- **Angular 12-13 (2021)** : dépréciation de View Engine, améliorations CSS et Webpack 5
- **Angular 14 (2022)** : typed forms, standalone components (en preview)
- **Angular 15 (2022)** : officialisation des composants standalone
- **Angular 16 (2023)** : introduction des Signals (nouveau modèle réactif), amélioration du SSR
- **Angular 17 (2023)** : routing plus rapide, améliorations des performances, hydration manuelle
- **Angular 18 (2024)** : **hydration automatique**, **Signals stabilisés**, **amélioration DX** (Developer Experience), **zone-less preview**

Pourquoi choisir Angular ?

- **Complet** : Angular est un framework "tout-en-un" qui inclut la gestion du routing, des formulaires, de l'internationalisation, des tests, etc.
- **TypeScript** : Typage statique, outils de développement avancés, meilleure lisibilité du code
- **Performant** : Détection des changements optimisée, rendu efficace, support du SSR
- **Soutenu par Google** : Développement continu, mises à jour fréquentes, documentation riche
- **CLI puissant** : Génération de code, tests, compilation, serveurs de développement

Cas d’usage typiques

Angular est particulièrement adapté aux applications complexes nécessitant une bonne structuration du code et une forte évolutivité, comme :

- **Applications d’entreprise** : tableaux de bord, outils internes, CRM, ERP
- **Applications web progressives (PWA)** : fonctionnalités offline, installation sur mobile
- **Portails administratifs** : gestion d’utilisateurs, permissions, rôles
- **Sites riches en données** : visualisations interactives, graphiques, gestion d’état

Nouveautés Angular 18

- Code plus clair (composants standalone, control flow natif)
- Réactivité simplifiée & typée via Signals
- Performances optimisées (hydratation partielle, absence de Zone.js, build plus rapide)
- Meilleure expérience développeur (outils, CLI, forms, state management, Angular DevTools)

SPA - Définition

- Une SPA (Single Page Application) est une application web qui ne recharge pas la page entière à chaque clic. Elle fonctionne dans une seule page HTML, et met à jour le contenu dynamique via JavaScript.
- Angular crée une SPA en fournissant une seule page HTML statique (index.html), puis en utilisant JavaScript (Angular) pour gérer toutes les interactions et la navigation dynamiquement, sans rechargement de page.
- Contrairement à SPA, on trouve **MPA** (Multi Page Application) qui est un site web où chaque clic de navigation recharge une nouvelle page complète depuis le serveur.

Définition de SSR (Server-Side Rendering)

Le SSR (Server-Side Rendering), c’est quand le HTML d'une page Angular est généré sur le serveur, avant d’être envoyé au navigateur.

- **Sans SSR** : Angular envoie juste du JavaScript, et le navigateur construit la page.
- **Avec SSR** : Le serveur crée la page HTML complète → le navigateur l’affiche immédiatement, puis Angular prend le relais.

RQ : Lors de la création d’un projet Angular, une question est posée si on veut activer le SSR ou non. Si vous répondez par oui des fichiers supplémentaires seront générés et qui sont coloré en marron ci après dans le cours.

```
C:\WINDOWS\system32>ng new mon-projet
? Which stylesheet format would you like to use? CSS             [ https://developer.mozilla.org/docs/Web/CSS
          ]
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? (y/N) y
```

SSR – Utiliser ou non

- Activer SSR si :

Situation	Pourquoi SSR est utile
SEO important (blog, e-commerce, contenu public)	Les moteurs de recherche voient une page HTML complète
Connexion lente ou vieux appareils	Le HTML est prêt côté serveur, donc affichage plus rapide
Améliorer le temps de chargement initial	L'utilisateur voit la page presque instantanément
Partage sur les réseaux sociaux (Open Graph)	Les balises HTML sont visibles sans JavaScript
Analytique critique dès le premier chargement	Les balises sont visibles sans attendre l'exécution JS

- Pas besoin de SSR si :

Situation	Pourquoi SSR est inutile
Application interne (intranet, dashboard, admin)	Pas de SEO requis, tous les utilisateurs sont connectés
Application SPA classique (type Gmail)	SSR ajoute de la complexité inutile
Performance backend critique	SSR augmente un peu la charge serveur
Tu veux du déploiement simple (SPA static)	Pas besoin de Node.js côté serveur

Standalone

- Le mot **"standalone"** ou **"autonome"** en français est un terme anglais souvent utilisé dans différents contextes (informatique, technologie, logiciels, etc.). Il signifie généralement que quelque chose peut **fonctionner de manière indépendante**, **sans dépendre d'autres systèmes, logiciels ou appareils**.
- Dans Angular, le terme "standalone" fait référence à une nouvelle façon de créer des composants, des directives ou des pipes sans avoir à les déclarer dans un module Angular (NgModule). C'est une fonctionnalité introduite à partir d'Angular 14, puis améliorée dans les versions suivantes.

C'est quoi un composant Standalone ?

Un **composant standalone** est un composant Angular qui :

- Peut **fonctionner sans être déclaré dans un NgModule**.
- Peut **importer directement d'autres composants, directives ou pipes standalone**.
- Est plus simple à utiliser dans des applications modernes, notamment avec des architectures modulaires ou micro-frontends.

C'est quoi un projet Standalone ?

Un **projet standalone** est un projet où :

- Tous les composants, directives et pipes sont créés avec standalone: true.
- Il n'y a **pas besoin de NgModules personnalisés** (AppModule, FeatureModule, etc.).
- L’application démarre depuis un **composant standalone de root**, via bootstrapApplication() (au lieu de bootstrapModule()).

Pourquoi Angular évolue vers standalone ?

- **Simplification** du développement.
- Réduction de la **courbe d'apprentissage** pour les débutants.
- Moins de **code boilerplate** (NgModule, declarations, etc.).
- Plus proche de **React / Vue** en termes de structure.
- Préparation pour des **architectures modernes** (micro-frontends, composants isolés, lazy loading par composant, etc.).

TypeScript

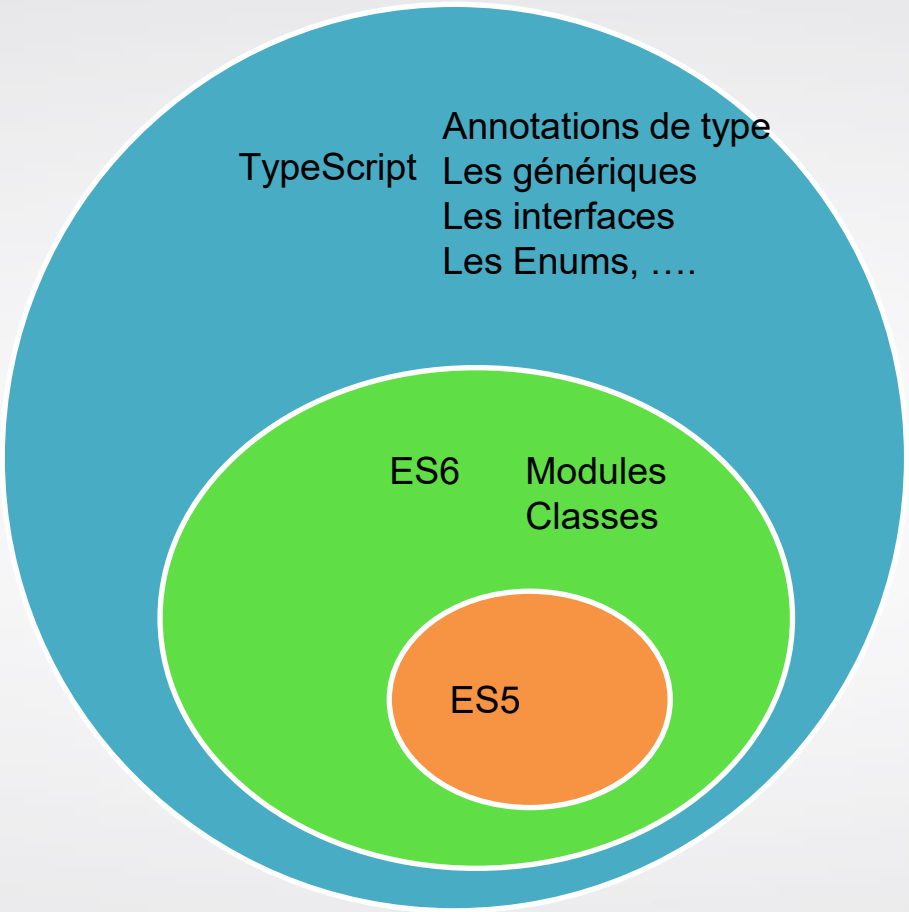
EcmaScript - Présentation

- C’est un est un ensemble de normes concernant les langages de programmation de type script comme JavaScript, ActionScript et Jscript : c’est un standard
- Normalisé par l’organisation ECMA International dans le cadre de la spécification ECMA-262.
- EcmaScript version/année est aussi le nom officiel de JavaScript
- Plusieurs versions existent pour Ecmascript. A partir de 2015 "version 6", chaque année en juin une nouvelle version majeur de EcmaScript est lancé. En juin 2024, la 15ème version de EcmaScript a vu le jour.

TypeScript - Présentation

- TypeScript est un langage gratuit, open-source, développé et maintenu par Microsoft depuis octobre 2012.
- C’est une sur-ensemble de JavaScript.
- TypeScript contient les spécifications de ES5 et ES6. Les fonctionnalités du langage TypeScript telles que les modules et l'orientation basée sur les classes font partie de la spécification ES6, tandis que les fonctionnalités telles que les génériques, les annotations de type, les inetrfaces, Enums, etc ne sont pas incluses dans les spécifications de l'ES6..
- Plusieurs verions de TypeScript existent (1.1 -> 5.x)

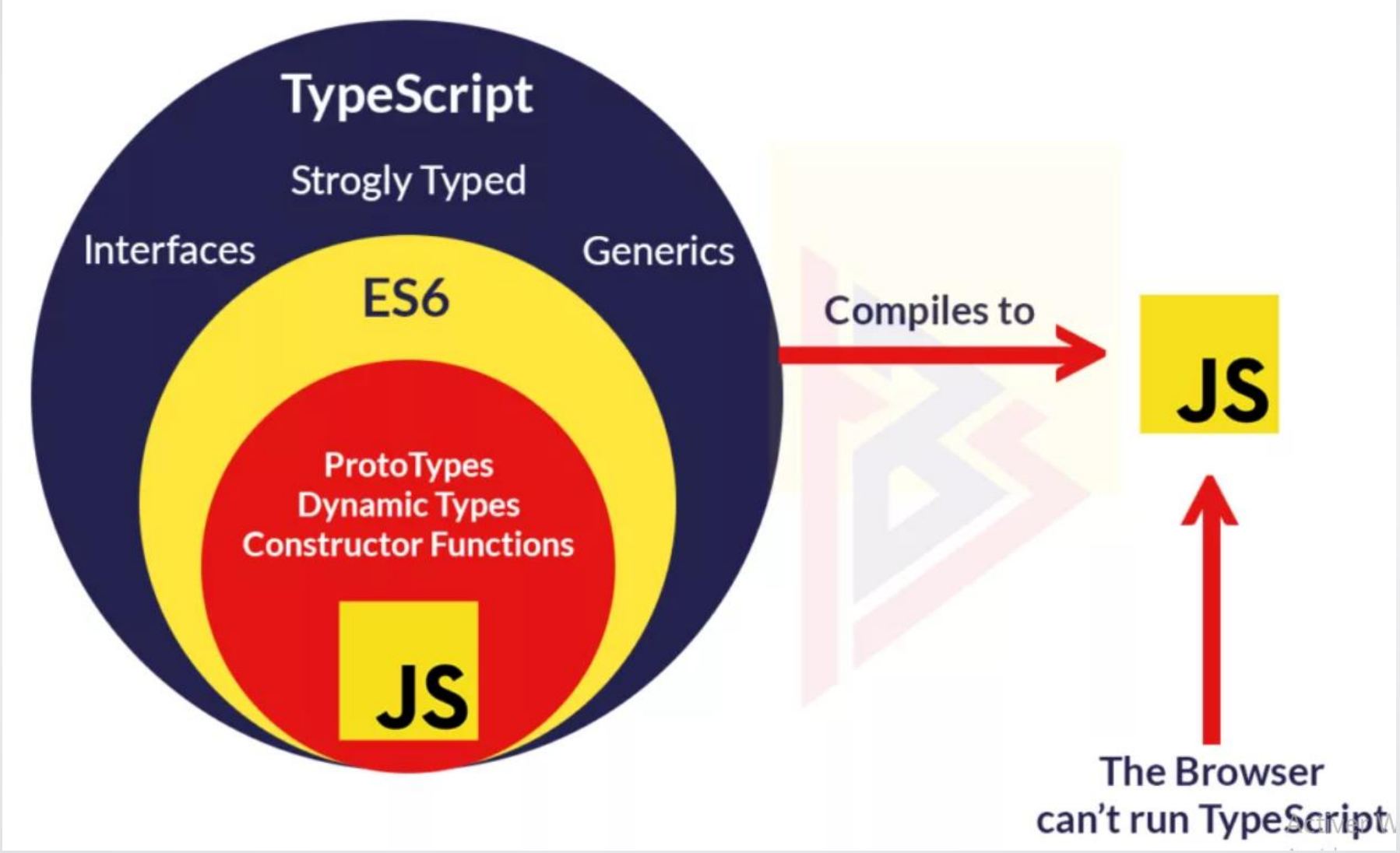
TypeScript - Présentation



TypeScript - Présentation

- Il n'existe pas actuellement pas un environnement d'exécution TypeScript et aucun support n'est prévu sur les navigateurs.
- Il faut alors utiliser un compilateur pour **transpiler** (C'est le fait de compiler le code source d'un langage en un autre langage) le code TypeScript en code JavaScript valide que l'on peut ensuite l'exécuter sur le runtime JavaScript de notre choix : Browser, NodeJS etc...
- Tout code valide en Javascript l'est également en TypeScript.
- TypeScript permet de simplifier le développement d'application JavaScript complexes.

TypeScript - Présentation



TypeScript - Caractéristiques

- TypeScript met toujours en évidence les erreurs au moment de la compilation pendant le développement, tandis que JavaScript signale les erreurs au moment de l'exécution.
- TypeScript prend en charge le typage fortement typé ou statique, alors que ce n'est pas en JavaScript.
- Il a un concept d'espace de noms en définissant un module.

Installation et mise en place de l’environnement

Outils nécessaires – Node.js

Node.js : Plateforme JavaScript côté serveur qui permet de

- Exécuter du JavaScript en dehors du navigateur (grâce au moteur V8 de Chrome)
- Créer des outils, serveurs, scripts, etc.

Angular utilise Node pour ses outils (compilation, tests, etc.)

Angular CLI, Webpack, ESLint, etc. tournent sur Node.

Outils nécessaires - NPM

npm (Node Package Manager) : Gestionnaire de paquets pour Node.js

- Permet d'installer, mettre à jour et gérer des bibliothèques JS
- Installe les dépendances d'un projet Angular (comme @angular/core, rxjs, etc.)
- Utilise le fichier package.json

Outils nécessaires - CLI

CLI (Command Line Interface) : Interface en ligne de commande

- utilisée via le terminal
- Angular a sa propre CLI : ng
- Permet de générer, tester, compiler des projets rapidement

Outils nécessaires - Version

A installer :

1) Version minimale de Node requise pour Angular 18 :

- Node.js 18.13.0
- Ou Node.js 20.x (recommandé)

A télécharger depuis https://nodejs.org/ et à installer

2) CLI pour la version 18 de Angular via la commande:

npm install @angular/cli@18

Outils nécessaires - Version

Autres prérequis associés : (sans installation explicite)

- npm : version compatible avec la version de Node utilisée (ex. npm 9+ avec Node 18, npm 10+ avec Node 20).

(installé par défaut avec Node)

- TypeScript : Angular 18 utilise TypeScript 5.4 par défaut.

Commandes utiles

Action	Commande
Vérifier la version de node installé	node -v
Vérifier la version de npm installé	npm -v
Vérifier la version de CLI installé	ng version
Installer un paquet js	npm install nompaquet

Création d’un projet Angular

Deux architectures de projets possibles à créer avec Angular 18:

- Projet standalone (par défaut)
- Projet non-standalone

Commande	**Résultat en Angular 18**
ng new mon-projet	Projet **standalone** par défaut
ng new mon-projet **--standalone false**	Projet **non-standalone** (avec modules)

Création d'un projet Angular

- Une fois la commande ng new est lancé, deux questions seront posées par la suite :

```
C:\WINDOWS\system32>ng new mon-projet
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS             [ https://developer.mozilla.org/docs/Web/CSS                    ]
  Sass (SCSS)     [ https://sass-lang.com/documentation/syntax#scss               ]
  Sass (Indented) [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less            [ http://lesscss.org                                            ]
```

```
C:\WINDOWS\system32>ng new mon-projet
? Which stylesheet format would you like to use? CSS             [ https://developer.mozilla.org/docs/Web/CSS
          ]
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? (y/N) y
```

Exécuter un projet Angular

Pour exécuter un projet Angular, on exécuter la commande **ng serve**

à partir de la racine du projet.

RQ : **ng serve –open** : Démarre et ouvre automatiquement le navigateur

- Cette commande compile le code source du projet pour transpiler le code TypeScript en Java Script et en même temps démarre un serveur Web local basé sur Node JS pour déployer l'application localement.
- Pour tester le projet généré, il suffit de langer le Browser et taper l'url : **http:// localhost:4200**

Etapes d’exécution d’un projet Angular

- Étape 1 : Chargement de la configuration (CLI lit les fichiers Angular angular.json, tsconfig.json, package.json (dépendances, scripts), éventuellement environment.ts)
- Étape 2 : Compilation avec Vite (ou Webpack) (.ts -> .js, .css, …)
- Étape 3 : Analyse des dépendances (Création d’un graph de dépendances, divise l’app en bundles (fichiers JS séparés pour main, vendor, etc.))
- Étape 4 : Démarrage du serveur de développement
- Étape 5 : Watch mode actif

Etapes d’exécution dans le navigateur

1. Le navigateur charge index.html
2. Il lit <script src="main.js"> → c’est le main.ts compilé
3. Le JS démarre Angular → charge module ou component
4. Angular crée et rend AppComponent

Etapes d’exécution dans le navigateur

Projet non-standalone	Projet standalone
index.html ↓ main.ts ↓ platformBrowserDynamic().bootstrapModule(AppModule) ↓ AppModule ↓ Angular analyse AppModule ↓ AppComponent (déclaré dans AppModule) ↓ Instanciation du composant ↓ ngOnInit(), affichage HTML	index.html ↓ main.ts ↓ bootstrapApplication(AppComponent) ↓ AppComponent (standalone: true) ↓ Instanciation du composant ↓ ngOnInit(), affichage HTML

Commandes CLI utiles

Commande	Description
ng generate component nom ou ng g c nom	Crée un composant
ng generate service nom ou ng g s nom	Crée un service
ng generate directive nom ou ng g d nom	Crée une directive
ng generate pipe nom ou ng g p nom	Crée un pipe
ng generate module nom ou ng g m nom	Crée un module
ng generate guard nom ou ng g g nom	Crée un guard

Commande	Description
ng build	Compile l'app pour la production
ng build --watch	Compile à chaque modification
ng build --configuration production	Build optimisé pour mise en ligne

Commande	Description
ng test	Lance les tests unitaires (Karma)
ng e2e	Lance les tests end-to-end (si configuré)
ng lint	Analyse statique du code (linting)

Arborescence d’un projet Angular classique vs standalone

Arboresence – Vue d’ensemble

Projet classique – Non standalone
ng new mon-projet-no-standalone --standalone false

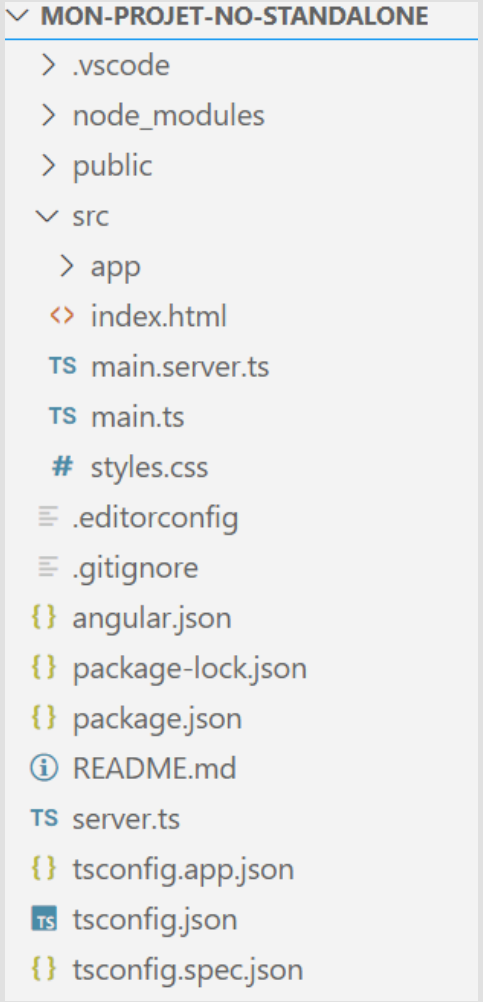
```
mon-projet/
│
├── src/
│   ├── app/
│   │   ├── app.module.ts          ← ✅ Module racine (NgModule)
│   │   ├── app.component.ts       ← 🧩 Composant racine
│   │   ├── app-routing.module.ts  ← 🔀 Module de routing
│   │
│   ├── main.ts                    ← 🚀 Point d’entrée
│   └── index.html                 ← HTML racine
│
├── angular.json                   ← Config Angular CLI
├── package.json                   ← Dépendances NPM
└── tsconfig.json                  ← Config TypeScript
```

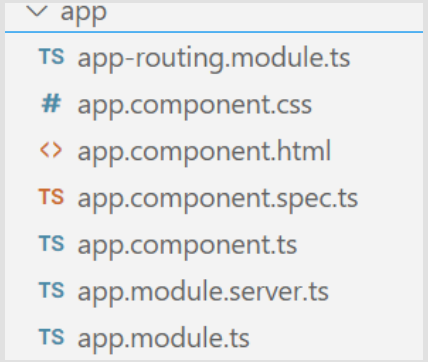
Projet standalone
ng new mon-projet

```
mon-projet/
│
├── src/
│   ├── app/
│   │   ├── app.component.ts       ← ✅ Composant racine standalone
│   │   ├── app.config.ts          ← ⚙️ Configuration de l'app (routing, providers)
│   │
│   ├── main.ts                    ← 🚀 Point d’entrée
│   └── index.html                 ← HTML racine
│
├── angular.json                   ← Config Angular CLI
├── package.json                   ← Dépendances NPM
└── tsconfig.json                  ← Config TypeScript
```

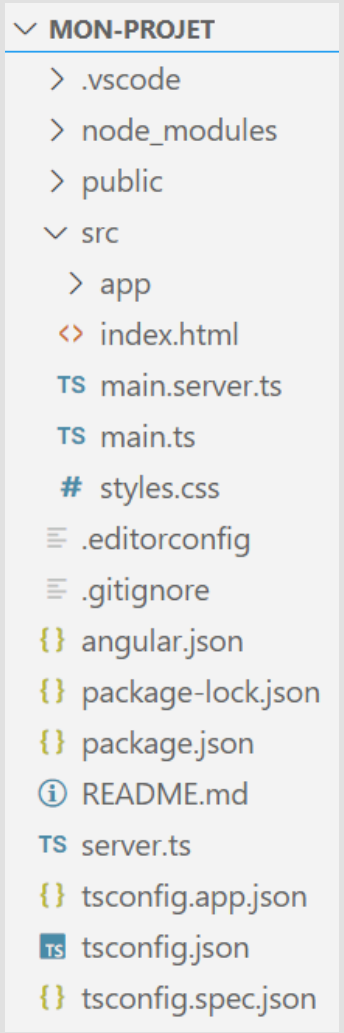
Arborescence - Détaillée

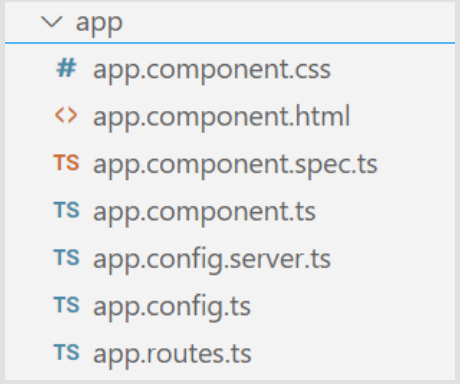
Projet classique – Non standalone
ng new mon-projet-no-standalone --standalone false





Projet standalone
ng new mon-projet





Arboresence – Socle en commun

node_modules : C'est un répertoire créé par npm pour stocker les dépendances d'un projet JavaScript.

public: Contient les fichiers statiques servis tels quels (images, favicon, HTML). N'est pas transformé par Angular.

index.html: Contient le fichier HTML de l'application.

main.server.ts: Est le point d'entrée côté serveur dans une application Angular avec SSR (Server-Side Rendering), grâce à Angular Universal.

main.ts: C'est le premier fichier exécuté lorsque l'application démarre. Il est responsable de charger et d'initialiser l'application Angular. Il charge les bibliothèques nécessaires pour l'application, telles que les bibliothèques de Angular et les fichiers de configuration.

styles.css: Ce fichier est utilisé pour définir les styles CSS globaux de l'application. Ces styles sont appliqués à tous les composants de l'application.

Arboresence – Socle en commun

angular.json: C'est un fichier de configuration permettant de définir les paramètres de base et les options de build de l'application.

package.json: Liste des dépendances déclarées avec des plages (^1.0.0, ~1.0.0) pour définir les besoins du projet

package-lock.json: Liste **verrouillée** de toutes les versions installées (y compris les sous-dépendances) pour une reproductibilité exacte des builds

tsconfig.app.json: Un fichier de configuration TypeScript utilisé spécifiquement pour compiler l'application Angular principale (hors tests et outils).

tsconfig.json: C'est un fichier de configuration utilisé par TypeScript pour définir les options de compilation (la version cible de JavaScript, le système de modules à utiliser, etc) et les paramètres de projet (le répertoire racine du projet, les fichiers à inclure ou exclure, etc..)

tsconfig.spec.json: Un fichier de configuration TypeScript spécifique aux tests unitaires dans un projet Angular**.**

Arboresence – dossier app

src/app: (fichiers en commun)

app.component.css : Contient le css du composant racine de l'application

app.component.html: Contient le code HTML du composant racine de l'application

app.component.ts: Contient la classe du composant racine de l'application

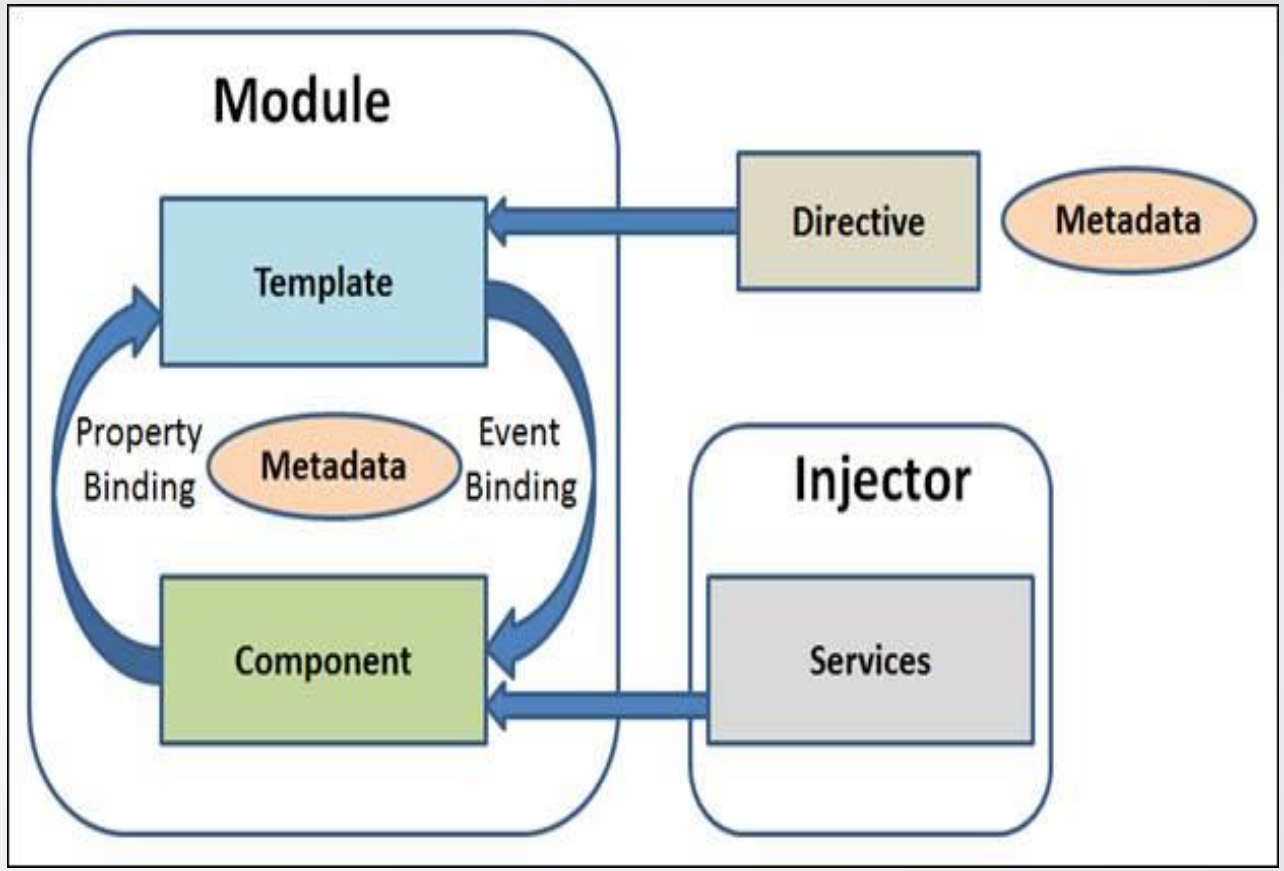
app.component.spec.ts: C'est un fichier de test unitaire pour le composant racine

Projet classique – Non standalone	Projet standalone
app.module.ts: Contient la configuration principale de l'application. **app.module.server.ts:** Module Angular **côté serveur** utilisé avec SSR. **app-routing.module.ts**: Contient la configuration de routage de l'application.	**app.config,ts:** Configuration de l'application (routes, providers, composants). Utilisé dans les applications standalone pour remplacer AppModule. **app.config.server.ts:** Configuration spécifique au serveur SSR. **app.routes.ts:** Contient les définitions de routes de l'application (Route[]).

Architecture d’un projet classique

Architecture d’un projet classique

1. Modules.
2. Components.
3. Templates.
4. Metadata.
5. Data binding.
6. Directives.
7. Services.
8. Dependency injection



Merci de votre attention