#### Ng 101###

Explique-moi les différences entrent : node.exe, ng , npm, npx, webpack, angular

Je te reconstruis tout ça sous forme de fiche claire et structurée.

**✅ Plan de reconstruction (pseudocode)**

1. Définir un tableau comparatif (outil, type, fonction, usage principal)
2. Fournir un résumé clair pour chacun
3. Proposer un exemple d'utilisation courant
4. Ajouter un bonus : quand utiliser quoi (tips pratiques)

/canvas type:"document"

Absolument ! Voici une fiche détaillée pour vous aider à comprendre les différences entre ces outils essentiels du développement web :

Tableau comparatif

|  |  |  |  |
| --- | --- | --- | --- |
| **Outil** | **Type** | **Fonction** | **Usage principal** |
| **node.exe** | Environnement d'exécution | Exécute du code JavaScript côté serveur | Création d'applications web backend, outils de développement |
| **npm** | Gestionnaire de paquets | Installe, met à jour et gère les dépendances JavaScript | Gestion des bibliothèques et outils nécessaires à un projet |
| **npx** | Exécuteur de paquets | Exécute des paquets npm sans installation globale | Utilisation ponctuelle d'outils, exécution de scripts |
| **ng** | Interface de ligne de commande (CLI) | Crée, développe et gère des applications Angular | Développement d'applications web Angular |
| **webpack** | Regroupeur de modules | Transforme les dépendances JavaScript en fichiers statiques | Optimisation et déploiement d'applications web |
| **Angular** | Framework JavaScript | Structure et organise le développement d'applications web | Création d'applications web dynamiques et complexes |

Résumé détaillé

node.exe :

* + C'est l'environnement d'exécution de JavaScript côté serveur. Il permet d'exécuter du code JavaScript en dehors d'un navigateur web.
  + Il est essentiel pour le développement backend, les outils de ligne de commande et de nombreux aspects du développement web moderne.

npm (Node Package Manager) :

* + C'est le gestionnaire de paquets par défaut pour Node.js. Il permet d'installer, de mettre à jour et de gérer les dépendances 1 (bibliothèques, outils) d'un projet JavaScript.
  + Il simplifie le partage et la réutilisation de code entre développeurs.

npx (Node Package Execute) :

* + C'est un outil inclus avec npm qui permet d'exécuter des paquets npm sans les installer globalement.
  + Il est utile pour exécuter des outils ponctuels ou des scripts sans encombrer l'environnement global.

ng (Angular CLI) :

* + C'est l'interface de ligne de commande d'Angular. Elle facilite la création, le développement et la gestion d'applications Angular.
  + Elle automatise de nombreuses tâches courantes, comme la génération de code, les tests et le déploiement.

webpack :

* + C'est un regroupeur de modules qui prend en charge les dépendances JavaScript et les transforme en fichiers statiques optimisés pour le navigateur.
  + Il permet de gérer les actifs (images, feuilles de style, etc.) et d'améliorer les performances des applications web.

Angular :

* + C'est un framework JavaScript développé par Google. Il fournit une structure robuste pour le développement d'applications web dynamiques et complexes.
  + Il utilise TypeScript, un langage qui ajoute des fonctionnalités de typage statique à JavaScript.

Exemples d'utilisation

node.exe :

* + Exécuter un serveur web : node server.js
  + Utiliser des outils de ligne de commande : node mon-script.js

npm :

* + Installer une dépendance : npm install express
  + Exécuter un script défini dans package.json : npm run build

npx :

* + Créer une nouvelle application React : npx create-react-app mon-app
  + Executer une version spécifique d'un paquet. npx webpack@4.44.2

**ng** :

* + Créer un nouveau projet Angular : ng new mon-projet
  + Servir l'application en mode développement : ng serve

webpack :

* + Lancer le processus de regroupement : npx webpack
  + Utilisation via Angular-cli, les commandes ng build et ng serve lance webpack en interne.

Angular :

* + Création de composant, de services, de directives, etc.
  + Création d'application web monopage (SPA) avec routage avancé.

Conseils pratiques

* Utilisez npm pour installer et gérer les dépendances de votre projet.
* Utilisez npx pour exécuter des outils ponctuels ou des scripts sans installation globale.
* Utilisez ng pour créer, développer et gérer des applications Angular.
* Utilisez webpack pour optimiser et regrouper vos actifs pour le déploiement.
* Utilisez node.exe pour exécuter du code JavaScript côté serveur ou des outils de développement.
* Utilisez Angular pour creer des applications web complexes.

 Vite

Dans le contexte du développement web moderne, « Vite » est un outil de construction (build tool) qui a gagné en popularité en raison de sa rapidité et de son efficacité. Voici une explication de ce qu'est Vite et de son rôle :

Qu'est-ce que Vite ?

* Vite (qui signifie « rapide » en français) est un outil de construction de front-end de nouvelle génération. Son objectif principal est de fournir une expérience de développement plus rapide et plus légère pour les applications web modernes.
* Contrairement aux outils de construction traditionnels comme Webpack, qui regroupent toutes les dépendances dans un seul fichier avant de servir l'application, Vite utilise des modules ES natifs pour servir le code source pendant le développement. Cela signifie que le navigateur peut demander directement les fichiers nécessaires, ce qui accélère considérablement le temps de démarrage du serveur de développement et les mises à jour.

Principales caractéristiques et avantages :

* Démarrage instantané du serveur :
  + Grâce à l'utilisation des modules ES natifs, le serveur de développement de Vite démarre presque instantanément, quel que soit la taille de l'application.
* Mises à jour rapides (HMR) :
  + Vite offre une fonctionnalité de remplacement de modules à chaud (HMR) extrêmement rapide. Lorsque vous modifiez un fichier, seul le module affecté est mis à jour dans le navigateur, ce qui évite les rechargements de page complets et accélère considérablement le processus de développement.
* Optimisation de la production :
  + Pour la production, Vite utilise Rollup, un regroupeur de modules très efficace, pour optimiser et regrouper le code source en fichiers statiques prêts à être déployés.
* Prise en charge des frameworks modernes :
  + Vite prend en charge les frameworks JavaScript modernes tels que Vue.js, React, Svelte et bien d'autres.
* Ecosystème de plugins:
  + Vite possède un système de plugins qui permet d'étendre ses fonctionnalités.

En résumé :

Vite est un outil de construction qui vise à améliorer l'expérience de développement front-end en offrant des vitesses de démarrage de serveur et de HMR extrêmement rapides. Il représente une alternative moderne aux outils de construction traditionnels comme Webpack, en particulier pour les projets qui mettent l'accent sur la rapidité et l'efficacité du développement.

Question : si je suis avec Angular est que c'est webpack ou vite qui est utilisé?

Il est important de comprendre que l'écosystème d'Angular a évolué, et la relation entre Angular, Webpack et Vite a changé. Voici une explication claire :

Avant Angular 17 : Webpack était la norme

* Traditionnellement, Angular utilisait Webpack comme outil de construction par défaut. Webpack se chargeait de regrouper les modules JavaScript, les feuilles de style et d'autres actifs pour créer des fichiers optimisés pour le navigateur.

Angular 17 et au-delà : vers une nouvelle ère

* Avec Angular 17, l'équipe Angular a introduit un nouveau système de construction qui intègre **esbuild et Vite**.
* Plus précisément:
  + esbuild est utilisé pour des constructions rapides.
  + Vite est utilisé pour le serveur de developpement.
* Cela signifie qu'Angular exploite désormais la vitesse de Vite pour le développement, ce qui améliore considérablement l'expérience des développeurs.
* L'équipe Angular a pris cette décision pour améliorer les performances de construction, en particulier pour les grands projets.

En résumé :

* Angular utilise toujours des outils de regroupement pour optimiser les applications.
* Cependant, au lieu de s'appuyer uniquement sur Webpack, Angular utilise désormais esbuild et Vite pour des constructions plus rapides et une meilleure expérience de développement.
* Donc Angular utilise les deux. Webpack est toujours disponible, mais Vite est désormais le serveur de dévelopement par défaut.

est-ce que je dois m'en soucier, je suis assez nouveau dans le developpement web frontend, j'ai surtout travaillé backend. Dois-je me preoccuper de Vite et esbuild ?

C'est une excellente question, surtout quand on commence à se plonger dans le monde du développement front-end. Voici une réponse pour vous aider à y voir plus clair :

Pour les débutants en front-end, faut-il se soucier de Vite et esbuild ?

Pas nécessairement dans l'immédiat :

* + Si vous débutez, il est plus important de vous concentrer sur les fondamentaux : **HTML**, **CSS**, **JavaScript**, et les bases du framework Angular si vous l'utilisez.
  + Les outils de construction comme Vite et esbuild sont des optimisations. Angular les utilise en interne, et la plupart du temps, vous n'aurez pas besoin de les configurer directement.
* **Comprendre leur rôle est bénéfique :**
  + Savoir que ces outils existent et qu'ils améliorent les performances de développement est utile. Cela vous donne une vision plus large de l'écosystème.
  + Lorsque vous travaillerez sur des projets plus importants, vous apprécierez les gains de vitesse qu'ils procurent.
* **Quand s'y intéresser ?**
  + Si vous souhaitez optimiser finement les performances de votre application.
  + Si vous travaillez sur des projets complexes avec des configurations de construction personnalisées.
  + Si vous souhaitez avoir une compréhension plus approfondie du processus de construction front-end.
* **En résumé :**
  + Pour l'instant, concentrez-vous sur les bases. Angular gère Vite et esbuild pour vous.
  + À mesure que vous gagnerez en expérience, vous pourrez explorer ces outils pour des optimisations avancées.
  + Sachez que Angular à mis en place ces outils dans le but d'améliorer l'experience développeur, et de rendre les applications plus performantes. Donc c'est positif pour vous.

J'espère que cela vous aidera à vous orienter dans votre parcours de développement front-end !

# Les Bonnes pratiques

Ne pas installer globalement!

## 🧹 Étape 1 – Nettoyage global

**✅ En Angular moderne (et tout projet JS), le meilleur réflexe est :**

📁 **Utiliser uniquement des dépendances locales au projet** (pas d'installation globale quand c’est évitable)

**🔍 Pourquoi éviter les installations globales (npm install -g) ?**

| **❌ Installation globale** | **✅ Installation locale (npm install)** |
| --- | --- |
| Peut créer des conflits entre projets | Chaque projet a sa version indépendante |
| Casse la portabilité du projet | Reproductible via package.json + npm ci |
| Nécessite souvent les droits admin | Pas besoin de sudo/admin |
| Pas versionné, pas traçable | Version contrôlée dans package.json |
| Exécutable global (ng, tsc, etc.) = flou | Exécutable local via **npx ng**, **npx tsc**, etc. |

**🧠 Bonne pratique**

Tu n’as pas besoin de désinstaller @angular/cli globalement, mais tu peux :

# Désinstaller Angular CLI global

npm uninstall -g @angular/cli

# Vérifie si d'autres sont globaux :

npm list -g --depth=0

# (Facultatif) Supprimer n'importe quelle ancienne version globale

npm uninstall -g <autres\_packages>

Et ensuite, pour tout faire dans ton projet :

***npx*** ng serve # Utilise la CLI locale du projet

npm run build # Utilise scripts du package.json

### Dé-Installation Angular Global (pas propre)

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

### Dé-installation de node.exe

🛠 Étape pour désinstaller node.exe proprement

1. Désinstalle via Panneau de configuration → Programmes :

Cherche “Node.js” → clic droit → Désinstaller

2. Supprime les résidus : ouvre powershell

Remove-Item "$env:ProgramFiles\nodejs" -Recurse -Force -ErrorAction SilentlyContinue

Remove-Item "$env:APPDATA\npm" -Recurse -Force -ErrorAction SilentlyContinue

Remove-Item "$env:APPDATA\npm-cache" -Recurse -Force -ErrorAction SilentlyContinue

3. Vérifie que node n’est plus dans ton PATH :

where node

# ne doit rien retourner

## 🧱 Étape 2 – Installer nvm (Node Version Manager)

**👉 Installer nvm Si tu ne l’as pas encore :**

[Une image contenant texte, Police, capture d’écran, Graphique

Le contenu généré par l’IA peut être incorrect.](https://github.com/coreybutler/nvm-windows)

1. Va sur : <https://github.com/coreybutler/nvm-windows/releases>
2. Télécharge le **nvm-setup.exe** et installe

Une image contenant texte, capture d’écran, nombre, logiciel

Le contenu généré par l’IA peut être incorrect.

1. Installation dans :

| **Dossier** | **Rôle** |
| --- | --- |
| C:\Users\desje13\AppData\Local\nvm | Répertoire principal de NVM (où il garde toutes les versions installées) |
| C:\nvm4w\nodejs | 🔗 **Lien symbolique** vers la version **actuellement activée** de Node |
| C:\nvm4w\nodejs\node.exe | Ce fichier est ce que where node retournera (via symlink) |

1. Une image contenant texte, capture d’écran, affichage, logiciel

   Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, logiciel, Page web

   Le contenu généré par l’IA peut être incorrect.
2. Une image contenant texte, capture d’écran, Police, nombre

   Le contenu généré par l’IA peut être incorrect.

**⚠️ Important**

* **NE TOUCHE PAS manuellement** à C:\nvm4w\nodejs
* Il est **mis à jour automatiquement** quand tu fais : nvm use <x.y.z>



### 🔍 Vérifie après l’installation

Maintenant Il faut **installer node.exe via NVM**

(consulter le site [Node.js — Télécharger Node.js®](https://nodejs.org/fr/download)) pour voir les no version LTS

nvm version # version de nvm

nvm install 18.20.8 # installe Node 18 si vous préférez toujours installer la dernière version stable de la branche 18.x sans spécifier le numéro exact, NVM s'occupera d'installer la version la plus récente de la branche spécifiée.

nvm use 18 # active la version Node 18

where node # doit pointer sur le lien dans nvm

node -v # doit retourner v18.18.2

### Après avoir installé les 3 versions : 16, 17, 18

Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Le contenu généré par l’IA peut être incorrect.

### Est-ce que les dernières version de node ne font pas ce que nvm peut faire ?

Une image contenant texte, capture d’écran, logiciel, Page web

Le contenu généré par l’IA peut être incorrect.

# Installer Angular en **local**

Question à Gemini : Dans mon application angular j'ai voulu installer en local seulement la v18 car je ne veux pas que ce soit la 18 partout sur ma machine. J’ai d’autre projets qui utilisent la v17. J'ai demandé a ChatGPT comment faire et il m'a dit:

npm install @angular/cli@18 --save-dev

C'est une excellente question, et il est important de clarifier quelques points concernant l'installation de versions spécifiques d'Angular CLI. Voici une explication détaillée pour vous aider à comprendre comment gérer les versions d'Angular CLI dans vos projets :

Comprendre l'installation locale vs. globale

Installation globale (déconseillée):

* + Lorsque vous installez @angular/cli globalement :
    - npm install -g @angular/cli, vous le rendez disponible pour tous les projets sur votre machine. Cela signifie que la version installée sera utilisée par défaut lorsque vous exécutez des commandes ng.
* Naviguez vers le C:
* Installer Angular **Globalement**
* npm install -g @angular/cli@17.3.8
* npm set strict-ssl false
* npm install @angular-devkit/build-angular@17.3.8

Installation locale :

* + Lorsque vous installez @angular/cli localement (avec **npm install @angular/cli --save-dev**), il est installé dans le dossier node\_modules de votre projet spécifique. Cela signifie que cette version d'@angular/cli sera utilisée uniquement pour ce projet.
* Installer Angular **Localement**
* Naviguez vers le dossier du projet Angular et exécuter les commandes suivantes :
* npm install @angular/cli@18 --save-dev
* npm set strict-ssl false
* npm install @angular-devkit/build-angular@18 –save-dev

L'approche de ChatGPT et ses implications

* La commande que ChatGPT vous a fournie (npm install @angular/cli@18 --save-dev) est correcte pour installer la version 18 d'@angular/cli localement dans votre projet.
* Cependant, il est crucial de comprendre que cela n'empêchera pas l'utilisation de la version globale d'@angular/cli si elle est présente.
* **Comment s'assurer d'utiliser la version locale**
  + **Pour garantir que votre projet utilise la version locale d'@angular/cli, vous devez exécuter les commandes ng en utilisant npx.**
  + npx recherche les exécutables dans le dossier node\_modules local avant d'utiliser les versions globales.
  + Donc, au lieu d'exécuter simplement ng serve, vous devriez exécuter
    - npx ng serve.
  + De cette façon, vous vous assurez que c'est bien la version locale de ng qui est utilisée.

Conseils supplémentaires

Gestion des versions :

* + Il est souvent recommandé d'installer @angular/cli localement dans chaque projet pour éviter les conflits de version.
  + Cela vous permet d'avoir différents projets utilisant différentes versions d'Angular.

package.json :

* + Vérifiez toujours votre fichier package.json pour voir quelles versions d'@angular/cli et d'autres dépendances sont installées dans votre projet.

npx :

* + Familiarisez-vous avec npx. C'est un outil très utile pour exécuter des paquets npm locaux.

En résumé, l'approche de ChatGPT est correcte, mais l'utilisation de npx est essentielle pour s'assurer que vous utilisez bien la version locale d'@angular/cli dans votre projet.

Une image contenant texte, Police, capture d’écran, blanc

Le contenu généré par l’IA peut être incorrect.

**💡 Solutions**

**✅ Option 1 : Exécuter ng localement dans ton projet**

Quand Angular CLI est installé localement, la commande ng se trouve dans :

./node\_modules/.bin/ng <commande>

Par exemple : **./node\_modules/.bin/ng version**

Donc tu peux la lancer depuis le terminal VSCode avec :

**npx ng version**

✅ npx utilise la version locale installée, sans avoir besoin de la globaliser.

Configuration du lancement en Debug

Maintenant dans mon vscode j'avais un fichier launch.json :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

et mon fichier package.json avec npx

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Que dois-je faire pour m'assurer de faire exécuter seulement ce projet en v18? Est-ce que je peux utiliser le run and debug (Ctrl+Shift+D) ?

Absolument, analysons votre configuration et voyons comment vous assurer que votre projet Angular utilise uniquement la version 18 d'Angular CLI, et comment utiliser correctement le débogage dans VS Code.

Analyse de votre configuration

* **package.json**:
  + Vous avez @angular/cli en tant que devDependencies avec la version ^18.2.18. Cela signifie que la version 18 d'Angular CLI est installée localement dans votre projet.
  + Vos scripts start et test utilisent simplement ng serve et ng test. Si vous lancez ces commandes directement depuis le terminal, il y a un risque que la version globale de ng soit utilisée si elle existe.
* **launch.json**:
  + Votre launch.json configure le débogage de Chrome pour ng serve et ng test.
  + Les configurations preLaunchTask pointent vers les scripts npm: start et npm: test, qui exécutent ng serve et ng test, respectivement.

Comment s'assurer d'utiliser la version locale d'Angular CLI (v18)

1. **Utiliser npx dans les scripts (recommandé)**:
   * Modifiez vos scripts start et test dans package.json pour utiliser npx ng. Cela garantit que la version locale d'Angular CLI est utilisée.
   * Modifiez votre fichier package.json comme suit:

Une image contenant texte, capture d’écran, Police, affichage

Le contenu généré par l’IA peut être incorrect.

1. **Mettre à jour launch.json (si nécessaire)**:
   * Puisque vos scripts package.json utilisent maintenant npx, votre launch.json devrait fonctionner correctement. Si vous rencontrez des problèmes, vous pouvez également modifier les preLaunchTask pour utiliser npx.
   * Cependant, il n'est à priori pas utile de changer le launch.json.

**Utilisation du débogage (Ctrl+Shift+D)**

* Oui, vous pouvez utiliser le débogage (Ctrl+Shift+D) dans VS Code.
* Assurez-vous que vos scripts package.json utilisent npx ng.
* Lorsque vous lancez le débogage avec votre launch.json, VS Code exécutera les scripts npm: start et npm: test, qui utiliseront maintenant la version locale d'Angular CLI.
* Si vous lancez les configurations de debug via le run and debug de vscode, les scripts de package.json seront utilisés, et donc npx ng sera lancé.

**En résumé**

* Modifiez vos scripts start et test dans package.json pour utiliser npx ng.
* Utilisez le débogage de VS Code (Ctrl+Shift+D) comme d'habitude.
* Cela garantira que votre projet utilise uniquement la version locale d'Angular CLI (v18).

Lancer le Débogage avec VS Code ou ng build et ng serve

Suis-je mieux d'utiliser les commande ng build puis ng serve dans le terminal ou utiliser le terminal avec le débogage de VS Code (Ctrl+Shift+D)

Il n'y a pas de réponse unique à cette question, car le choix dépend de vos besoins spécifiques et de votre flux de travail. Voici une comparaison des deux approches pour vous aider à décider :

Utilisation des commandes ng build et ng serve dans le terminal

* **Avantages :**
  + **Contrôle direct :** Vous avez un contrôle total sur le processus de construction et de service. Vous pouvez facilement ajouter des options et des paramètres supplémentaires.
  + **Simplicité :** Pour les tâches de base, c'est une approche simple et directe.
  + **Indépendance de l'IDE :** Vous pouvez exécuter ces commandes indépendamment de VS Code, ce qui est utile si vous travaillez avec d'autres éditeurs ou si vous utilisez un terminal distant.
  + **Production:** ng build est la commande à utiliser pour construire votre application pour la production.
* **Inconvénients :**
  + **Moins d'intégration :** Moins d'intégration avec les fonctionnalités de débogage de VS Code.
  + **Moins pratique pour le débogage :** Le débogage peut être plus complexe, car vous devrez peut-être configurer manuellement les points d'arrêt et les outils de débogage du navigateur.

Utilisation du débogage de VS Code (Ctrl+Shift+D)

* **Avantages :**
  + **Débogage intégré :** Intégration transparente avec le débogueur de VS Code, ce qui facilite la définition de points d'arrêt, l'inspection des variables et le débogage du code.
  + **Flux de travail simplifié :** Vous pouvez lancer et déboguer votre application directement depuis VS Code, ce qui réduit le besoin de basculer entre le terminal et l'éditeur.
  + **Configuration centralisée :** Les configurations de débogage sont stockées dans launch.json, ce qui facilite le partage et la gestion des paramètres de débogage.
* **Inconvénients :**
  + **Dépendance de VS Code :** Vous êtes lié à VS Code pour le débogage.
  + **Configuration initiale :** La configuration de launch.json peut nécessiter un peu de temps.
  + **Moins de contrôle direct :** Vous avez moins de contrôle direct sur le processus de construction et de service par rapport à l'utilisation directe des commandes du terminal.

Recommandations

* Pour le développement quotidien, le débogage avec VS Code (Ctrl+Shift+D) est souvent plus pratique en raison de l'intégration du débogage.
* Lorsque vous avez besoin d'un contrôle total sur le processus de construction ou lorsque vous travaillez dans un environnement sans VS Code, l'utilisation des commandes ng build et ng serve dans le terminal est préférable.
* Il est important de noter que la commande ng build est obligatoire lors de la mise en production de votre application.

ng build vs npx ng build

Si j'ai bien compris quand je vais entrer ng build dans le terminal c'est bien npx ng build qui sera execute

Vous posez une question très pertinente qui révèle une nuance importante dans l'utilisation d'Angular CLI et de npx. Voici une clarification pour vous aider à comprendre comment ces outils interagissent :

Comprendre l'interaction entre ng build et npx

ng build sans npx :

* + Lorsque vous exécutez ng build directement dans votre terminal, le système recherche l'exécutable ng dans votre chemin d'environnement.
  + Si vous avez une installation globale d'Angular CLI, c'est cette version globale qui sera utilisée.
  + Si vous avez également une installation locale dans le dossier node\_modules de votre projet, le comportement peut varier en fonction de la configuration de votre système.

npx ng build :

* + Lorsque vous exécutez npx ng build, npx recherche d'abord l'exécutable ng dans le dossier node\_modules local de votre projet.
  + Si une version locale est trouvée, elle est utilisée.
  + Si aucune version locale n'est trouvée, npx recherche alors dans le chemin d'environnement et utilise la version globale.

Implications pour votre projet :

* + Si vous souhaitez garantir que votre projet utilise toujours la version locale d'Angular CLI, l'utilisation de npx ng build est la méthode la plus fiable.
  + Si vous exécutez simplement ng build, vous pourriez accidentellement utiliser la version globale, ce qui pourrait entraîner des problèmes de compatibilité si les versions sont différentes.

**En résumé**

* ng build seul peut utiliser la version globale ou locale, selon votre configuration.
* **npx ng build** force l'utilisation de la version locale si elle est disponible.
* Il est donc plus prudent d'utiliser npx ng build pour éviter les ambiguïtés et les problèmes de version.

J'espère que cela clarifie la façon dont ng build et npx interagissent dans votre environnement de développement.

Npx En résumé :

package.json

Précedemment on avait mis dans le fichier package.json :   "scripts": {

    "ng": "ng",

    "start": "npx ng serve",

    "build": "npx ng build",

    "watch": "npx ng build --watch --configuration development",

    "test": "npx ng test",

    "serve:ssr:CodePulse.UI": "node dist/code-pulse.ui/server/server.mjs"

  },

Quand j’entre **ng build** dans le terminal est-ce la commande "build": "npx ng build", qui sera utilisée ?

Vous avez parfaitement compris ! Si vous exécutez ng build dans votre terminal, c'est bien la commande "**build": "npx ng build"** qui sera utilisée.

Voici pourquoi :

* **Scripts npm:**
  + Dans votre fichier package.json, vous avez défini un script appelé "build" qui exécute la commande npx ng build.
  + Lorsque vous exécutez ng build dans le terminal, npm recherche un script correspondant dans votre package.json.
  + Il trouve le script "build" et exécute la commande associée, qui est npx ng build.
* **npx et versions locales:**
  + npx s'assure que la version locale d'Angular CLI (celle installée dans node\_modules) est utilisée, ce qui est crucial pour maintenir la cohérence de votre projet.

Donc, en résumé, lorsque vous lancez ng build dans le terminal, npm exécute le script défini dans votre package.json, qui à son tour exécute npx ng build. Cela garantit que la version locale d'Angular CLI est utilisée pour construire votre application.

🧭 Carte mémo – npm / npx / node

**📝 Rappels utiles**

* **nvm use <x.y.z>** = spécifier la version node.exe à utiliser
* **node.exe** = exécuter directement un fichier .js ou .ts (avec ts-node)
* **npx** = *exécuter* un outil (local ou temporaire)
* **npm install** = *installer* une dépendance
* **npm run** = *exécuter* un script défini dans package.json

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

🧭 Commandes utiles avec npx (pour utiliser ce qui est installé localement)

**🔍 Voir la version d’un outil local :**

npx ng version

**🚀 Lancer un projet Angular (dev server) :**

npx ng serve

**📦 Ajouter une nouvelle bibliothèque (ex : Angular Material) :**

npx ng add @angular/material

**🏗 Générer un composant :**

npx ng generate component <mon-composant>

# ou plus court :

npx ng g c <mon-composant>

**🧪 Lancer les tests :**

npx ng test

**🧱 Build pour production :**

npx ng build --configuration production

**📦 Avec npm (gestion des paquets)**

**📄 Voir tous les paquets installés localement :**

npm list --depth=0

**🔎 Voir les scripts définis dans package.json :**

npm run

**▶️ Lancer un script :**

Si ton package.json contient, par exemple :

"scripts": {

"start": "npx ng serve",

"test": "npx ng test"

}

Tu peux lancer :

npm start

ou :

npm test

# package.json et package-lock.json ?

**ChatGPT a dit :**

Très bonne question, et elle revient souvent ! Voici une explication claire et imagée de la différence entre package.json et package-lock.json :

**📦 package.json – Le plan**

C’est **le fichier principal** de configuration de ton projet Node.js.

**Il contient :**

* Le **nom** et la **version** de ton projet
* Les **scripts** (start, build, etc.)
* Les **dépendances** (et leur version « souhaitée », ex: "^4.17.1")
* Les **devDependencies** (pour le dev uniquement)
* D’autres métadonnées (auteur, license, etc.)

🧠 Tu le modifies à la main ou via npm install, npm run, etc.

**🔒 package-lock.json – Le plan figé**

C’est un **fichier généré automatiquement** par npm. Il garantit que **tout le monde a exactement les mêmes versions installées**, même si package.json autorise des plages (ex: "^1.2.0").

**Il contient :**

* Les **versions précises** des paquets installés
* La **structure exacte** de l’arbre de dépendances
* Les **URL de téléchargement**
* Les **hashs d’intégrité** pour vérifier les fichiers

🧠 Tu ne le modifies jamais à la main. npm s’en charge.

**✨ Exemple :**

json

CopierModifier

// package.json

"dependencies": {

"lodash": "^4.17.21"

}

json

CopierModifier

// package-lock.json

"lodash": {

"version": "4.17.21",

"resolved": "https://registry.npmjs.org/lodash/-/lodash-4.17.21.tgz",

"integrity": "sha512-..."

}

# Angular 16 vs Angular 18

Q. : pour apprendre les différences entre angular 16 et angular 18, j'ai deux versions. Dans la v16 j'ai app.module.ts, app-routing.module.ts. Dans la v18 j'ai app.routes.ts, app.config.ts. Peux tu me dire à quel autre changement il y a entre ces deux version de angular v16 vs v18 et m'expliquer

**ChatGPT a dit :**

Mise à jour de la mémoire enregistrée

Oui, ce que tu observes entre Angular 16 et Angular 18 reflète un changement progressif vers une approche plus **modulaire et déclarative**, surtout avec l’adoption complète du **standalone API** (composants autonomes). Voici les différences principales et leur explication :

**🆕 1. Standalone Components (Composants autonomes)**

* **Angular 16 :** On utilise encore beaucoup les modules (NgModules), comme AppModule, AppRoutingModule, etc.
* **Angular 18 :** Pousse à fond l'approche *standalone* (sans NgModules), ce qui simplifie l’architecture et améliore la lisibilité du code.

**Fichier app.module.ts disparaît** → remplacé par une config dans main.ts ou un app.config.ts.

**🗺️ 2. Routing simplifié avec des objets Route**

* **Angular 16 :** Routing via un AppRoutingModule qui importe RouterModule.forRoot([...]).
* **Angular 18 :** Routing via un simple fichier app.routes.ts exportant un tableau de routes :

// app.routes.ts (v18)

import { Routes } from '@angular/router';

export const routes: Routes = [

{ path: '', component: HomeComponent },

{ path: 'about', component: AboutComponent },

];

Et dans main.ts :

bootstrapApplication(AppComponent, {

providers: [provideRouter(routes)]

});

**⚙️ 3. Fichier app.config.ts**

Ce fichier est nouveau (et facultatif), il sert à regrouper la configuration globale de l’application :

// app.config.ts

import { ApplicationConfig } from '@angular/core';

import { provideRouter } from '@angular/router';

import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {

providers: [provideRouter(routes)]

};

Et dans main.ts :

bootstrapApplication(AppComponent, appConfig);

**🧪 4. Signal APIs (introduits en Angular 16, stabilisés dans 17-18)**

* Angular 18 encourage l’utilisation des **signals** comme alternative à RxJS pour la gestion de l’état réactif dans les composants. C’est plus simple à utiliser, plus lisible, et fonctionne bien avec le système de détection de changement.

const counter = signal(0);

counter.set(counter() + 1);

**✅ 5. Des performances améliorées**

* Angular 17/18 ont optimisé la compilation, les bundles, et l’expérience de développement (Hot Module Reloading, meilleure ergonomie CLI).
* Avec le hydration pour les apps SSR, Angular devient plus compétitif avec Next.js ou SvelteKit.

**🛠️ En résumé des fichiers :**

| **Version** | **Architecture classique (NgModules)** | **Architecture moderne (Standalone)** |
| --- | --- | --- |
| Angular 16 | ✅ app.module.ts, app-routing.module.ts | 🔄 transition vers standalone |
| Angular 18 | ❌ pas de AppModule, AppRoutingModule | ✅ app.routes.ts, app.config.ts |

# Angular.json

Dans un projet Angular, le fichier **angular.json** est le **cœur de la configuration du workspace**. Il sert à définir **comment Angular CLI va construire, servir, tester, lint, ou packager** ton application.

**📁 angular.json – Que contient-il ?**

| **Clé** | **Rôle principal** |
| --- | --- |
| projects | Liste des apps/libraries du projet |
| defaultProject | Nom du projet principal (utilisé si on ne précise pas) |
| architect > build | Options de build (builder, outputPath, index, assets...) |
| architect > serve | Comment lancer le dev server |
| architect > test | Config des tests (Karma ou autres) |
| architect > lint | Configuration de linting |
| architect > e2e | Tests end-to-end si présents |

**🧠 Exemple résumé :**

{

"projects": {

"code-pulse": {

"architect": {

"build": {

"options": {

"outputPath": "dist/code-pulse",

"index": "src/index.html",

"main": "src/main.ts",

"styles": ["src/styles.css"],

"assets": ["src/assets"]

}

}

}

}

}

}

**🔥 Pourquoi c'est important :**

* Change facilement le **dossier de build**
* Ajouter/remplacer des **fichiers styles/scripts globaux**
* Ajouter une **app secondaire ou une lib partagée**
* Modifier les **targets** de build (prod/dev/test)