
Exemples de Configuration tsconfig.json pour l'Exclusion de Fichiers

Ces exemples illustrent comment utiliser les propriétés include, exclude et compilerOptions.skipLibCheck dans votre fichier tsconfig.json pour contrôler quels fichiers sont transpilés par TypeScript.
Pour chaque exemple, nous allons considérer la structure de projet suivante :

```
mon-projet/
├── src/
│   ├── app.ts
│   ├── utils.ts
│   ├── tests/
│   │   ├── unit.test.ts
│   │   └── mock.dev.ts
│   └── lib/
│       ├── common.ts
│       └── third-party-stub.dev.ts
├── scripts/
│   └── build.ts
├── node_modules/
│   └── ... (dépendances)
├── tsconfig.json
└── package.json
```

Exemple 1 : Exclure un Fichier Cible Spécifique

Scénario : Vous avez un fichier de test ou un fichier temporaire (src/tests/unit.test.ts) que vous ne voulez pas inclure dans le build final ou la vérification des types.
tsconfig.json :
JSON

```
{
  "compilerOptions": {
    "outDir": "./dist",          // Dossier de sortie pour les fichiers transpilés
    "rootDir": "./src",          // Dossier racine des fichiers TypeScript
    "strict": true,              // Active toutes les options de vérification de type strictes
    "module": "esnext",          // Spécifie le système de modules (ex: ES Modules)
    "target": "es2020",          // Cible la version JavaScript (ex: ES2020)
```

```
    "esModuleInterop": true,         // Permet l'interopérabilité des modules CommonJS/ES Modules
    "skipLibCheck": true             // Optionnel: Ignore la vérification de type des fichiers de bibliothèques
  },
  "include": [
    "src/**/*.ts"                    // Inclut tous les fichiers .ts sous le répertoire 'src'
  ],
  "exclude": [
    "src/tests/unit.test.ts",        // Exclut spécifiquement ce fichier
    "src/lib/old-legacy-code.ts"     // Autre exemple de fichier spécifique à exclure
  ]
}
```

Explication :

Dans cet exemple, tous les fichiers TypeScript (.ts) sous src/ seront inclus par défaut via l'include. Cependant, le fichier src/tests/unit.test.ts sera explicitement ignoré lors de la transpilation et de la vérification des types grâce à la propriété exclude.

Exemple 2 : Exclure Tous les Fichiers avec une Extension Spécifique (.dev.ts)

Scénario : Vous utilisez des fichiers avec l'extension .dev.ts pour des configurations de développement, des mocks ou des scripts spécifiques au développement, et vous voulez vous assurer qu'ils ne sont jamais inclus dans votre build de production.

tsconfig.json :

JSON

```
{
  "compilerOptions": {
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "module": "esnext",
    "target": "es2020",
    "esModuleInterop": true,
    "skipLibCheck": true
  },
  "include": [
    "src/**/*.ts"
  ],
  "exclude": [
```

```
    "**/*.dev.ts"              // Exclut tous les fichiers finissant par .dev.ts, quel que soit leur emplacement
  ]
}
```

Explication :

Le motif **/*.dev.ts est un motif glob puissant qui signifie "tout fichier avec l'extension .dev.ts, dans n'importe quel sous-répertoire du répertoire racine du projet". Ainsi, src/tests/mock.dev.ts et src/lib/third-party-stub.dev.ts seraient tous deux exclus.

Exemple 3 : Exclure un Répertoire Spécifique

Scénario : Votre projet contient un dossier scripts/ qui contient des scripts utilitaires pour le build ou le déploiement qui sont écrits en TypeScript mais ne font pas partie de l'application principale. Vous avez également un dossier dist/ (le dossier de sortie) que TypeScript n'a pas besoin de vérifier ou de traiter.

tsconfig.json :

JSON

```
{
  "compilerOptions": {
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "module": "esnext",
    "target": "es2020",
    "esModuleInterop": true,
    "skipLibCheck": true
  },
  "include": [
    "src/**/*.ts" // Inclut uniquement le code source de l'application
  ],
  "exclude": [
    "scripts",    // Exclut le répertoire 'scripts' et tout son contenu à la racine du projet
    "dist",       // Exclut le répertoire de sortie (souvent une bonne pratique)
    "src/tests"   // Exclut l'intégralité du répertoire de tests dans 'src'
  ]
}
```

Explication :

Les entrées "scripts", "dist" et "src/tests" dans exclude disent à TypeScript d'ignorer complètement ces répertoires et tous les fichiers qu'ils contiennent, même s'ils correspondent à d'autres règles d'inclusion.

Exemple 4 : Gestion des Dépendances (node_modules) et skipLibCheck

Scénario : Vous travaillez sur un projet Node.js avec de nombreuses dépendances installées via npm/yarn dans le dossier node_modules. Vous voulez vous assurer que TypeScript ne vérifie pas les types de ces bibliothèques tierces, ce qui peut accélérer la compilation et éviter les erreurs de type qui ne sont pas de votre ressort.

tsconfig.json :

JSON

```
{
  "compilerOptions": {
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "module": "commonjs",          // Typiquement pour les projets Node.js
    "target": "es2020",
    "esModuleInterop": true,
    "skipLibCheck": true,          // ✨ IMPORTANT : Ignore la vérification de type des fichiers de bibliothèques
    "forceConsistentCasingInFileNames": true // Bonne pratique pour éviter les problèmes de casse
  },
  "include": [
    "src/**/*.ts"                // Inclut uniquement les fichiers TypeScript de votre code source
  ],
  "exclude": [
    // "node_modules"            // Pas nécessaire d'exclure explicitement si 'include' ne l'englobe pas
                                 // et si 'skipLibCheck' est activé.
                                 // Par défaut, 'node_modules' est déjà exclu.
    "dist"                       // Exclure le dossier de sortie est toujours une bonne pratique
  ]
}
```

Explication :

- L'entrée include: ["src/**/*.ts"] est essentielle ici. Elle limite la portée de TypeScript à votre propre code source dans le dossier src/. Par conséquent, le dossier node_modules (qui n'est pas sous src/) n'est pas automatiquement balayé par TypeScript.
- "exclude": ["node_modules"] n'est généralement **pas nécessaire** lorsque include est défini de manière restrictive. TypeScript exclut node_modules par défaut des dossiers balayés s'il n'est pas explicitement inclus.

- La ligne **"skipLibCheck": true** est la clé ici. Elle indique au compilateur TypeScript de ne pas effectuer une vérification de type complète sur les fichiers de définition de type (.d.ts) trouvés dans node_modules. Cela résout la plupart des problèmes de performances et des "fausses" erreurs provenant des bibliothèques tierces, sans empêcher TypeScript de connaître les types déclarés par ces bibliothèques pour votre propre code.
