

- 2. Angular CLI / Webpack / Debug
 - Angular CLI
 - Présentation
 - Installation / Première utilisation
 - Description du contenu
 - Définition d'un projet Angular
 - package.json
 - angular.json
 - Webpack
 - Principe
 - Entry
 - Output
 - Loaders
 - Plugins
 - Installation de Webpack
 - Configuration Webpack
 - Lancement de Webpack
 - Exemple de projet webpack
 - Débuguer une application Angular

2. Angular CLI / Webpack / Debug

Angular CLI

Présentation

Angular CLI met à disposition la majorité des outils permettant de construire une application basée sur Angular.

- Création d'un squelette d'application,
- Génération de squelette de composants (TS/HTML/SCSS/SPEC)
- Construction de l'application,
- Lancer les Test Unitaires et les Tests End To End,
- Proxifier le backend du projet.

L'atout principale de Angular CLI est sa simplicité d'utilisation, et de configuration, puisque tout est défini dans un fichier JSON : angular.json (anciennement angular-cli.json)

Dans la plus part des cas, Angular CLI est la solution à utiliser pour gérer un projet basé sur Angular.

Vous pouvez voir toutes les commandes de génération [ici](#)

Installation / Première utilisation

Pour installer Angular CLI :

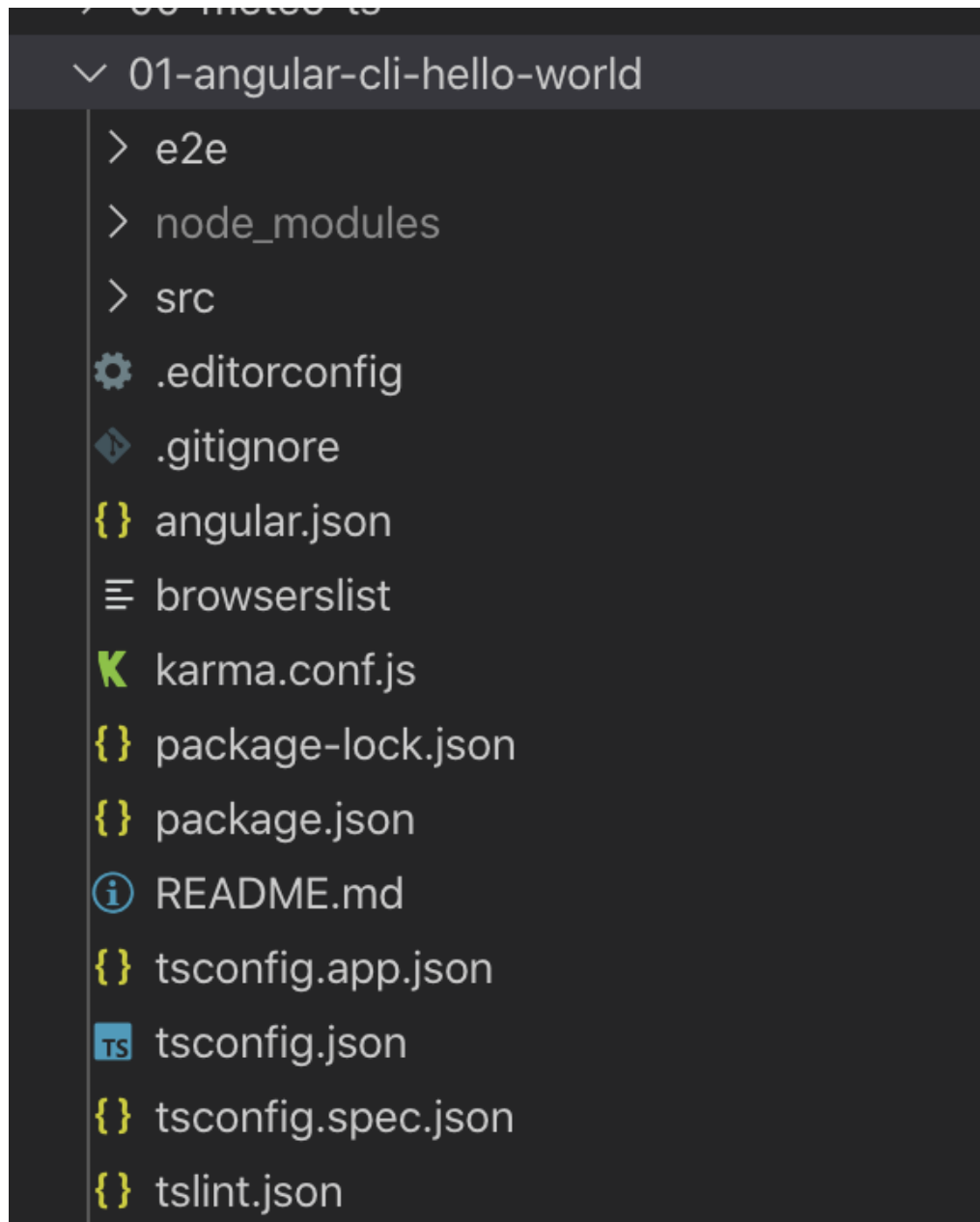
```
npm i -g @angular/cli
```

Pour initialiser un nouveau projet Angular :

```
ng new hello-world
```

NB : dans le cas où `ng` n'est pas reconnue en tant que commande interne, veuillez vérifier la variable `path` de l'environnement, notamment le chemin vers NPM.

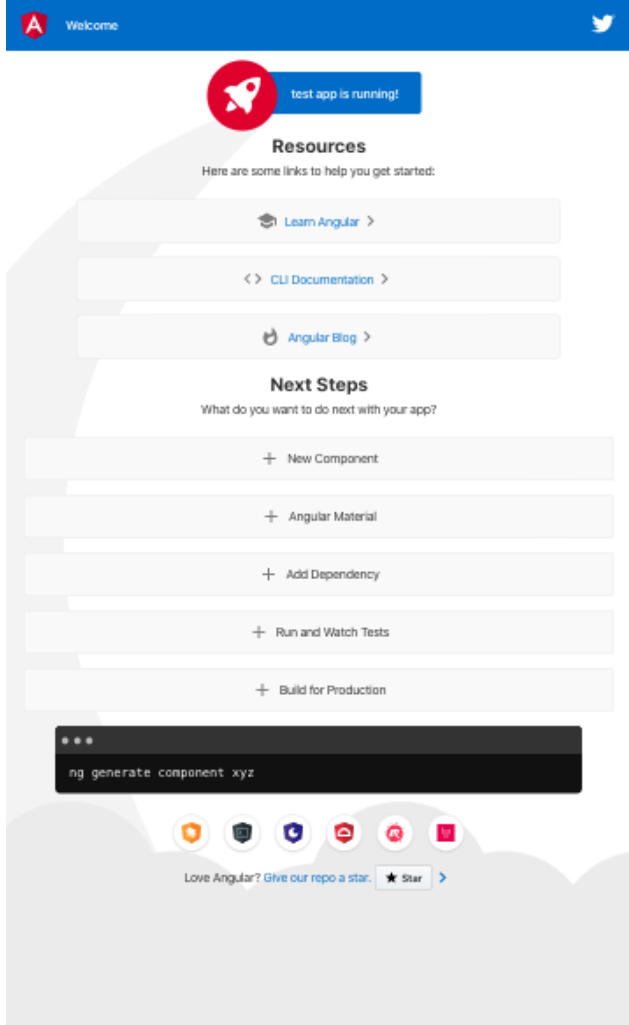
Angular génère une template de projet :



Positionner vous dans le dossier généré, puis lancer le projet :

```
ng serve
```

Le projet est déployé sur le serveur local à l'adresse <http://localhost:4200>



Le projet est disponible sur [Gitlab](#)

Description du contenu

La template de projet contient l'ensemble des fichiers nécessaires au démarrage d'un projet industrialisé.

- `/e2e/` : spécification des tests d'intégration,
- `/nodes_modules/` : dépendances npm,
- `/src/` : sources du projet,
- `angular.json` : définition du projet @ang/cli (voir chapitre suivant),
- `.editorconfig` : configuration de formatage du code (à ignorer, remplacé par Prettier),
- `.gitignore` : liste des fichiers / dossier à ne pas pousser sur le repository,
- `karma.conf.js` : configuration des Tests Unitaires (voir chapitre sur les tests unitaire),
- `package.json` : définition du projet, des dépendances npm, des tâches npm,
- `protractor.conf.js` : configuration des Tests d'intégration (E2E) (voir chapitre sur les tests d'intégration),
- `README.md` : fichier de présentation du projet au format Markdown,
- `tsconfig.json` : configuration du transpiler TSC (voir chapitre [Typescript](#)),
- `tslint.json` : configuration de l'analyseur de code (voir chapitre [TSLint](#))

Définition d'un projet Angular

package.json

En préambule, il est nécessaire de faire un focus sur ce fichier, qui n'est en rien lié aux spécificités d'un projet Angular, mais est plutôt aux projets javascript en général, intégrant des dépendances NPM.

Le fichier `package.json` décrit :

- `name` : Nom principale du package, utilisé pour le chargement de la dépendance, et comme nom dans le repertoire `node_modules`.
- `version` : Version du package,
- `description` : Description succincte du package,
- `keywords` : mot clés associés au package,
- `licence` : droits / restriction d'utilisation du package,
- `private` : défini si le package peut être publié sur un gestion de package,
- `publishConfig` : défini les paramètre de publication sur un repository (attentions, le fichier `.npmrc` de votre poste doit être correctement configuré pour pouvoir accéder au repository configuré si vous souhaitez publié depuis votre poste.),
- `dependencies` : Dépendances nécessaires à la fois pour le développement et l'exécution en production de votre package.
- `devDependencies` : Dépendances uniquement nécessaires pour le développement, mais ne seront pas installés en production.
- `peerDependencies` : Dépendances qui seront installées automatiquement si besoin. C'est souvent le cas lorsqu'un paquet nécessite une version spécifique pour fonctionner.
- `os` : OS compatibles/incompatibles avec le package (ex: win32, linux, !win64)
- `cpu` : Architectures compatibles/incompatibles avec le package (ex : arm, x64, !ia32)

Exemple de fichier `package.json` :

```
{
  "name": "hello-world",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~8.2.4",
    "@angular/common": "~8.2.4",
    "@angular/compiler": "~8.2.4",
    "@angular/core": "~8.2.4",
    "@angular/forms": "~8.2.4",
    "@angular/platform-browser": "~8.2.4",
    "@angular/platform-browser-dynamic": "~8.2.4",
    "@angular/router": "~8.2.4",
    "rxjs": "~6.4.0",
    "tslib": "^1.10.0",
    "zone.js": "~0.9.1"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~0.803.3",
    "@angular/cli": "~8.3.3",
    "@angular/compiler-cli": "~8.2.4",
    "@angular/language-service": "~8.2.4",
    "@types/node": "~8.9.4",
    "@types/jasmine": "~3.3.8",
    "@types/jasminewd2": "~2.0.3",
    "code-lyzer": "^5.0.0",
    "jasmine-core": "~3.4.0",
    "jasmine-spec-reporter": "~4.2.1",
    "karma": "~4.1.0",
    "karma-chrome-launcher": "~2.2.0",
```

```

    "karma-coverage-istanbul-reporter": "~2.0.1",
    "karma-jasmine": "~2.0.1",
    "karma-jasmine-html-reporter": "^1.4.0",
    "protractor": "~5.4.0",
    "ts-node": "~7.0.0",
    "tslint": "~5.15.0",
    "typescript": "~3.5.3"
  }
}

```

Plus d'informatione sur le [site NPM](#)

angular.json

Le fichier `angular.json` (pour Ang 6+) est le point d'entrée de toute la configuration du projet Angular.

Plus d'information sur [Angular CLI](#)

Webpack

Angular-CLI est basé sur le builder Webpack. Angular-CLI simplifie la mise en oeuvre du build, en inhibant la complexité de configuration de build de Webpack, au travers du fichier `angular.json`.

```

"build": {
  "builder": "@angular-devkit/build-angular:browser",
  "options": {
    "outputPath": "dist/hello-world",
    "index": "src/index.html",
    "main": "src/main.ts",
    "polyfills": "src/polyfills.ts",
    "tsConfig": "tsconfig.app.json",
    "aot": false,
    "assets": ["src/favicon.ico", "src/assets"],
    "styles": ["src/styles.css"],
    "scripts": []
  },

```

Principe

Webpack est un module bundler, c'est à dire qu'il est capable de générer un module (ou plusieurs) à partir d'un ensemble de fichier en point d'entrée (**entry**). Les **loaders** permettent d'interpréter un grand nombre de type de fichier pour les packager dans un module. Différents **plugins** existent et rentrent dans la phase de construction du build. Les sources seront packagées et construite dans un dossier de sortie (**output**), sous forme d'un bundle JS, et un ensemble d'assets définient comme devant être disponible dans la distribution.

[Documentation officielle de Webpack](#)

Entry

Les **entry** sont les sources, et assets en entrée de la constuction WebPack.

Output

Les **output** représentent le bundle de sortie, matérialisée en générale, par un bundle JS, et les assets configurées comme devant faire partie de la distribution.

Loaders

Par défaut, Webpack ne gère que du JavaScript, mais il possède un système de loaders qui permettent transformer d'autres types de ressources en Javascript. Ainsi, chaque ressource chargée par Webpack devient un module qu'il est possible d'importer dans le code.

On peut citer les loaders principaux :

- Typescript,
- Javascript,
- Sass,
- CSS,
- Images,
- JSON

[Liste des loaders Webpack](#)

Plugins

Les plugins permettent de réaliser des traitements spécifique lors de la construction d'un bundle :

- Injection de variables,
- Partage de dépendances communes,
- Minification du code,
- etc.

[Liste des plugins Webpack](#)

Installation de Webpack

L'installation de Webpack se fait au travers des dépendances NPM (ou dans dans vos `devDependencies` du fichier `package.json`):

```
npm install -g webpack@latest
npm install -g webpack-cli@latest
npm install -g webpack-dev-server@latest
```

Configuration Webpack

La configuration de webpack est décrite au travers d'un fichier Javascript.

Exemple de configuration intégrant les notion de loaders et de plugins exploité pour la construction d'une application :

```
var webpack = require("webpack");
var path = require("path");
const CopyWebpackPlugin = require("copy-webpack-plugin");

module.exports = {
  entry: "./src/index.js",
  output: {
    filename: "bundle.js",
    path: __dirname + "/dist",
  },

  module: {
    rules: [
      {
        test: /\.scss$/,
        loader: "style-loader!css-loader!sass-loader",
```

```

    },
    {
      test: /\.js$/,
      loader: "babel-loader",
    },
    {
      test: /\.html$/,
      loader: "raw-loader",
    },
  ],
},
plugins: [
  new CopyWebpackPlugin([
    {
      from: path.join(__dirname, "src/public"),
    },
  ]),
],
];
};

```

Noter que les loaders & plugins sont très souvent des dépendances NPM également, et qu'il est donc nécessaire de les ajouter dans les `devDependencies` du `package.json`.

Exemple :

```

npm install babel-loader style-loader css-loader sass-loader --save-dev
npm install html-webpack-plugin webpack --save-dev

```

L'option `--save-dev` rajoute automatiquement ces dépendances dans la liste des `devDependencies`.

Lancement de Webpack

Webpack est lancé au travers de ligne de commande dans le dossier contenant la définition webpack.

```

webpack --config webpack.config.js

```

[Voir la liste des commandes disponibles](#)

Exemple de projet webpack

Le projet `02-webpack-hello-world` basé sur WebPack est disponible [ici](#).

Cet exemple basique démontre l'exploitation d'un build de projet basé sur webpack :

- un fichier d'entrée pour webpack intégrant des dépendances vers des modules JS et SCSS,
- un bundle de sortie dans un fichier Javascript,
- l'exploitation de loaders permettant d'inclure différents type de fichier : JS, CSS, SCSS, HTML,
- l'exploitation de plugins pour copier des assets dans le dossier de desination du build.

Pour installer le projet :

```

npm install

```

Pour lancer le build webpack :

```

webpack --config webpack.config.js

```

Vous pouvez constater que le buidl webpack génère un dossier `dist` intégrant le bundle Javascript, incluant l'arbre de dépendances résolu, ainsi qu'une copie de la page `index.html`.

Débugger une application Angular

Pour débbuger une application au travers de VSCode, vous devez :

- Télécharger l'addon "Debugger for Chrome"
- Configurer le launcher au travers du fichier 'launch.json' (Debug > Add Configuration)

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "chrome",
      "request": "launch",
      "name": "Launch Chrome against localhost",
      "url": "http://localhost:4200",
      "webRoot": "${workspaceFolder}"
    },
    {
      "type": "chrome",
      "request": "attach",
      "name": "Attach to Chrome",
      "port": 9222,
      "webRoot": "${workspaceFolder}"
    }
  ]
}
```

- Lancer l'application de manière habituelle (npm run start)
- Lancer le debug (Debug > Run)
- Positionner un point d'arrêt dans le code Typescript
- Lancer l'application

Paused in Visual Studio Code



Resources

Here are some links to help you get started:

 [Learn Angular >](#)

[<> CLI Documentation >](#)

 [Angular Blog >](#)

Next Steps

What do you want to do next with your app?

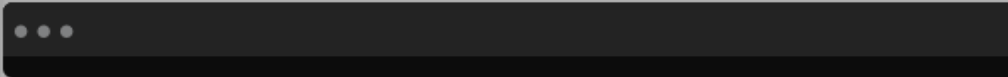
+ New Component

+ Angular Material

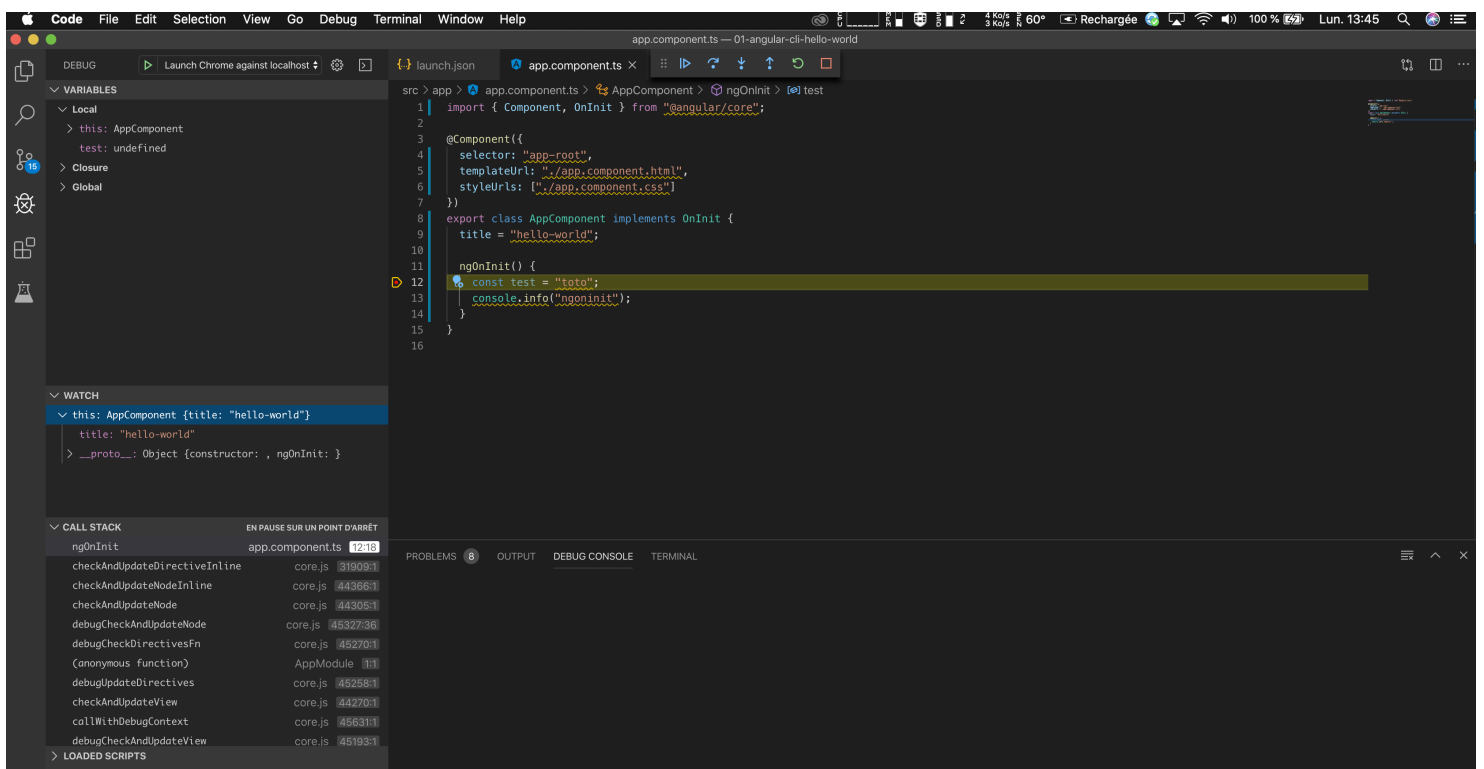
+ Add Dependency

+ Run and Watch Tests

+ Build for Production



Love Angular? [Give our repo a star.](#) [★ Star >](#)



The screenshot shows the Visual Studio Code editor with the Angular CLI application code open. The code is in the file `app.component.ts` and is as follows:

```
src > app > app.component.ts > AppComponent > ngOnInit > test
1 | import { Component, OnInit } from '@angular/core';
2 |
3 | @Component({
4 |   selector: 'app-root',
5 |   templateUrl: './app.component.html',
6 |   styleUrls: ['./app.component.css']
7 | })
8 | export class AppComponent implements OnInit {
9 |   title = 'hello-world';
10 |
11 |   ngOnInit() {
12 |     const test = 'koto';
13 |     console.info('ngOnInit');
14 |   }
15 | }
16 |
```

The debug console on the left shows the following output:

```
DEBUG
Launch Chrome against localhost
app.component.ts x
launch.json
app.component.ts x
src > app > app.component.ts > AppComponent > ngOnInit > test
1 | import { Component, OnInit } from '@angular/core';
2 |
3 | @Component({
4 |   selector: 'app-root',
5 |   templateUrl: './app.component.html',
6 |   styleUrls: ['./app.component.css']
7 | })
8 | export class AppComponent implements OnInit {
9 |   title = 'hello-world';
10 |
11 |   ngOnInit() {
12 |     const test = 'koto';
13 |     console.info('ngOnInit');
14 |   }
15 | }
16 |
```

The call stack on the left shows the following stack trace:

```
CALL STACK
ngOnInit
app.component.ts 12:18
checkAndUpdateDirectiveInline core.js 31909:1
checkAndUpdateNodeInline core.js 44366:1
checkAndUpdateNode core.js 44305:1
debugCheckAndUpdateNode core.js 45327:36
debugCheckDirectivesFn core.js 45270:1
(Anonymous function) AppModule 1:1
debugUpdateDirectives core.js 45258:1
checkAndUpdateView core.js 44270:1
callWithDebugContext core.js 45631:1
debugCheckAndUpdateView core.js 45193:1
```