



*Institut Galilée Université Paris 13
Villetaneuse
2018-2019*

Projet LEE

Tutoriel d'introduction à angular schematics

Réalisé par : Louiza AOUCHETA

Tutoriel d'introduction à angular schematics

Avant de commencer le tutoriel sur l'utilisation de **angular.schematics**, on va débiter par un bref aperçu sur ce qu'est **schematics** (schémas) et pourquoi ils sont utilisés.

I. Présentation de angular schematics

Schematics en français schémas est un outil de workflow pour le Web moderne.

Dans Angular on entend souvent parler de Schematics et Collection

schematics - est une "recette" qui peut être exécutée en utilisant `ng generate <schematic-name>` pour générer et ajuster des fichiers de projet

collection - (qui est une collection de schémas c'est à dire une liste de schémas) est un package (package npm) qui contient au moins un schéma.

Si on utilise Angular CLI, nous pouvons exécuter beaucoup de schémas par défaut car ils sont inclus dans la `@schematics/angular` collection. Cela permet de générer des éléments tels que des composants ou des services.

Nous pouvons également installer une collection supplémentaire à partir de **npm** et exécuter ses schémas en transmettant un **--collection** comme indicateur supplémentaire à `ng generate` tel que si on fait `ng generate <schematic-name> --collection <collection-name>` .

En général, les schémas permettent de:

- Ajouter des bibliothèques à un projet angular,
- Mettre à jour des bibliothèques dans un projet angular
- Générer du code.
- Créer un nouveau composant ou mettre à jour du code pour corriger des modifications importantes d'une dépendance
- Ajouter une nouvelle option de configuration ou une nouvelle structure à un projet existant.

Exemples d'avantages pour créer une collection de schémas:

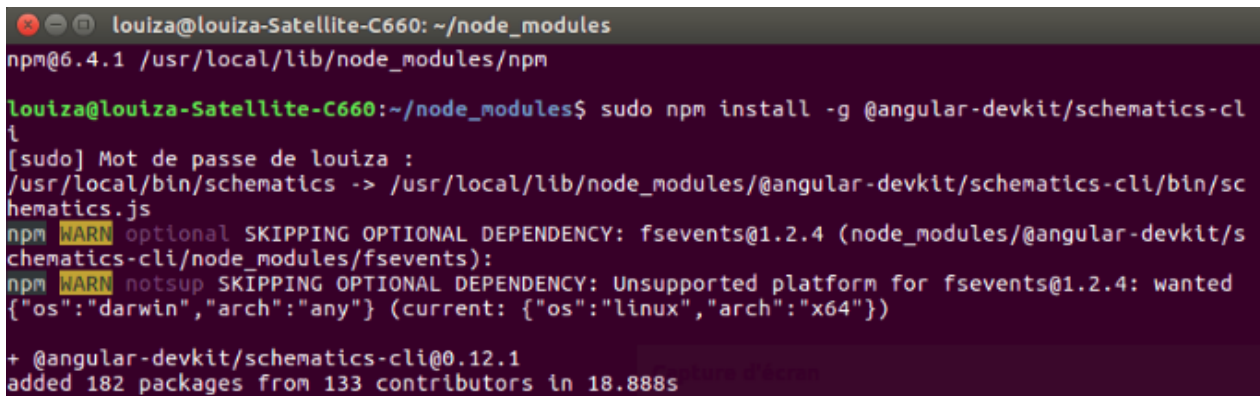
- Génération de modèles d'interface utilisateur couramment utilisés dans une application.
- Génération de composants spécifiques à l'organisation à l'aide de modèles ou de présentations prédéfinis.
- Mise en application de l'architecture organisationnelle.

II. Création d'un premier schéma

Avant de créer un schéma angular il est nécessaire de s'assurer que Node 6.9 ou une version supérieur est bien installé sur la machine sur lequel le travail doit être effectué.

Ensuite, installez Schematics via une de commande:

```
sudo npm install -g @angular-devkit / schematics-cli
```



```
louiza@louiza-Satellite-C660: ~/node_modules
npm@6.4.1 /usr/local/lib/node_modules/npm

louiza@louiza-Satellite-C660:~/node_modules$ sudo npm install -g @angular-devkit/schematics-cl
i
[sudo] Mot de passe de louiza :
/usr/local/bin/schematics -> /usr/local/lib/node_modules/@angular-devkit/schematics-cli/bin/sc
hematics.js
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules/@angular-devkit/s
chematics-cli/node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted
{"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

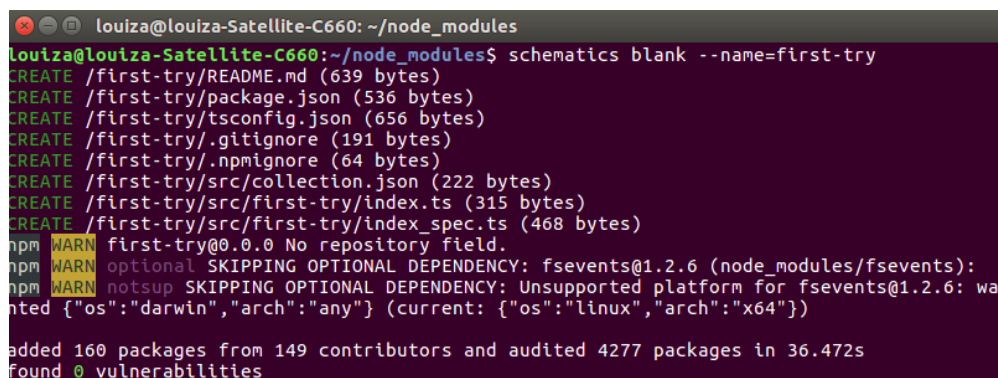
+ @angular-devkit/schematics-cli@0.12.1
added 182 packages from 133 contributors in 18.888s
```

Cela installera un exécutable de schematics.

Maintenant que nous avons installé l'outil via une ligne de commande, nous avons accès à cet exécutable, cependant, nous avons la possibilité de créer un nouveau projet de schémas vierge à l'aide de Schematics CLI:

```
louiza@louiza-Satellite-C660:~/node_modules$ schematics blank --name=first-try
```

En exécutant cette ligne de commande nous observons que tous les fichiers liés au schéma sont créés



```
louiza@louiza-Satellite-C660:~/node_modules
louiza@louiza-Satellite-C660:~/node_modules$ schematics blank --name=first-try
CREATE /first-try/README.md (639 bytes)
CREATE /first-try/package.json (536 bytes)
CREATE /first-try/tsconfig.json (656 bytes)
CREATE /first-try/.gitignore (191 bytes)
CREATE /first-try/.npmignore (64 bytes)
CREATE /first-try/src/collection.json (222 bytes)
CREATE /first-try/src/first-try/index.ts (315 bytes)
CREATE /first-try/src/first-try/index_spec.ts (468 bytes)
npm WARN first-try@0.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.6 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.6: wa
nted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

added 160 packages from 149 contributors and audited 4277 packages in 36.472s
found 0 vulnerabilities
```

Une fois le projet créé nous pouvons l'ouvrir dans un environnement de travail qui prend en charge angular tel que WebStorm et nous allons voir qu'un schéma dispose réellement de 4 principaux fichiers .

- package.json : contient le nom du projet, sa version, sa description ainsi que certaines dépendances pour déboguer et tester des scripts et la propriété la plus importante est schematics qui dispose d'un chemin vers le fichier collection.json

- collection.json: contient la définition du schéma

- index.ts: représente le code schematics réel

- ts.config : représente la configuration du compilateur TypeScript

A l'intérieur du dossier **src** nous allons trouver le fichier **collection.json**, en ouvrant ce fichier nous observons que le bout de code suivant est généré automatiquement



```
1 {
2   "$schema": "../../node_modules/@angular-devkit/schematics/collection-schema.json",
3   "schematics": {
4     "first-try": {
5       "description": "A blank schematic.",
6       "factory": "./first-try/index#firstTry"
7     }
8   }
9 }
```

La clé importante est "schematics" , qui décrit les schémas inclus dans cette collection.

Dans notre exemple, nous décrivons un schéma: first-try , il a une description simple avec un champ factory.

Le champ factory est une chaîne de caractères (string) qui spécifie l'emplacement du fichier du point d'entrée de notre schéma, suivi d'un symbole de hachage, suivi de la fonction qui sera appelée. Dans ce cas, nous invoquerons la fonction firstTry() dans le fichier first-try / index.ts qui fait référence à une chaîne de caractères pour pointer sur une fonction JavaScript. Il représente le RuleFactory .

Une Rule (règle) est une fonction qui prend un Tree (arbre) et retourne un autre Tree . Les règles sont au cœur des schémas; ce sont eux qui apportent des modifications aux projets, appellent des outils externes et implémentent une logique. Comme son nom l'indique une RuleFactory est une fonction qui crée une règle dite Rule.

L'arbre Tree est une structure de données qui contient une base (un ensemble de fichiers déjà existants) et une zone de stockage intermédiaire (liste des modifications à appliquer à la base). Lorsque des modifications sont apportées, il n'y a pas de changement réel de la base, mais nous ajoutons ces modifications à la zone intermédiaire.

Le Tree est utilisé par angular CLI et représente le projet sur le lecteur jusqu'au premier schéma appelé, mais les schémas composés peuvent recevoir n'importe quel Tree.

Voici le RuleFactory vierge créé jusqu'à présent dans index.ts

```
package.json × collection.json × index.ts ×
Compile TypeScript to JavaScript?
1 import { Rule, SchematicContext, Tree } from '@angular-devkit/schematics';
2
3
4 // You don't have to export the function as default. You can also have more than one rule factory
5 // per file.
6 export function firstTry(_options: any): Rule {
7   return (tree: Tree, _context: SchematicContext) => {
8     return tree;
9   };
10 }
```

La fonction `firstTry` prend en argument un ensemble d'options et retourne une `Rule` qui prend un `Tree` et le retourne inchangé.

L'argument `_options` est un objet qui peut être considéré comme un input. Dans la CLI, il s'agit des arguments de ligne de commande que l'utilisateur a transmis. Du point de vue d'un autre schéma, ce sont les options qui ont été transmises par ce schéma.

Dans la plupart des cas, il s'agit souvent d'un objet quelconque et peut être saisi par `any`. Il peut également être validé avec un schéma JSON pour assurer que les entrées ont les types et valeurs par défaut appropriés.

Il est aussi possible de créer un fichier à la racine du `Tree` du schéma de la `RuleFactory`

```
package.json × collection.json × index.ts ×
Compile TypeScript to JavaScript?
1 import { Rule, SchematicContext, Tree } from '@angular-devkit/schematics';
2
3
4 // You don't have to export the function as default. You can also have more than one rule factory
5 // per file.
6 export function firstTry(_options: any): Rule {
7   return (tree: Tree, _context: SchematicContext) => {
8     tree.create({ path: 'hello', content: "Hello World" });
9     return tree;
10   };
11 }
```

Pour résumer, un `Tree` contient les fichiers sur lesquels les schémas doivent être appliqués. Il contient une liste de fichiers et contient des métadonnées associées aux modifications souhaitées. Dans notre cas, la seule modification apportée consiste à créer un nouveau fichier et sont considérés comme une collection de fichiers et de modifications.

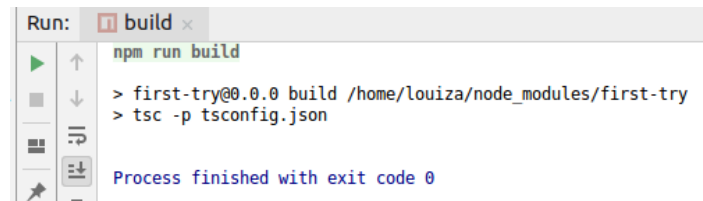
Par défaut, angular CLI transmet la racine du projet angular en tant que `Tree`, mais tout schéma peut être transmis à un `Tree` différent des autres. De plus il existe quatre méthodes qui créent directement un changement dans un `Tree` et qui sont : `create`, `delete`, `rename` et `overwrite`.

III. Exécution d'un schéma angular

Pour exécuter notre exemple, il est impératif d'utiliser l'outil de ligne de commande "schematics" avec le chemin d'accès au répertoire de notre projet schématique en tant que "collection".

```
louiza@louiza-Satellite-C660:~/node_modules$ npm run build
```

... attend la fin de la construction et on obtient



Cette dernière ligne veut dire que l'exécution ne retourne aucune erreur

```
louiza@louiza-Satellite-C660:~/node_modules$ schematics.: first-try
```

... voir qu'un fichier est créé à la racine.

```
louiza@louiza-Satellite-C660:~/node_modules/first-try$ schematics .:first-try
CREATE /hello (5 bytes)
```

Lors du débogage (qui peut également être utilisé avec --debug=true), le --debug=true par défaut consiste également à s'exécuter en mode d'exécution à sec, ce qui empêche l'outil de créer des fichiers.

En utilisant l'argument --dry-run=false cela pourrait être changé. Par conséquent, cela signifie que les changements se produiront réellement sur le système de fichiers. Si un fichier a été supprimé ou écrasé, il y a un risque de perte de contenu. L'idéal c'est d'être dans un répertoire temporaire séparé lors du débogage des schémas et de désactiver les essais à blanc uniquement lorsque cela est nécessaire.

Nous pouvons également démarrer npm run build -- -w dans un terminal séparé afin de reconstruire automatiquement notre projet schématique lorsqu'un fichier est modifié.

IV. Exemple de collection de schémas

Il est possible de consulter des exemples de schémas et de lire le code et les commentaires associés via la commande suivante:

```
louiza@louiza-Satellite-C660:~/Documents/M2/LEE/projet/schematics/first-try$
schematics schematic --name demo
```

Cela va nous créer une collection de schémas

```
louiza@louiza-Satellite-C660: ~/Documents/M2/LEE/projet/schematics/demo/src
louiza@louiza-Satellite-C660:~/Documents/M2/LEE/projet/schematics$ schematics schematic --name demo
CREATE /demo/README.md (639 bytes)
CREATE /demo/package.json (525 bytes)
CREATE /demo/tsconfig.json (631 bytes)
CREATE /demo/.gitignore (191 bytes)
CREATE /demo/src/collection.json (1698 bytes)
CREATE /demo/src/my-full-schematic/index.ts (2455 bytes)
CREATE /demo/src/my-full-schematic/index_spec.ts (864 bytes)
CREATE /demo/src/my-full-schematic/schema.json (306 bytes)
CREATE /demo/src/my-full-schematic/files/test2 (127 bytes)
CREATE /demo/src/my-full-schematic/files/test__INDEX__ (29 bytes)
CREATE /demo/src/my-other-schematic/index.ts (1346 bytes)
CREATE /demo/src/my-other-schematic/index_spec.ts (509 bytes)
CREATE /demo/src/my-schematic/index.ts (1458 bytes)
CREATE /demo/src/my-schematic/index_spec.ts (482 bytes)
```

Si on modifie dans le package.json le chemin de collection.json qui est dans la ligne

"schematics": "./src/collection.json", par

"schematics": "./collection.json", notre fichier collection.json prendra les collections de tous les schémas.



Cette collection de schémas peut être utilisée avec angular CLI, pour ce faire il faut commencer par créer un projet vide avec la CLI via la commande
ng new my-project

Ensuite, dans le nouveau projet, lier la collection de schémas que nous venons de construire:
npm link \$PATH_TO_SCHEMATIC_PROJECT

Il faut remplacer \$PATH_TO_SCHEMATIC_PROJECT par le chemin d'accès à la racine du projet.

Une fois le projet de schematics est lié, nous pouvons utiliser ng generate pour appeler les schémas:

ng generate first-try: first-try

Lien github : https://github.com/Louizaaaa/angular_schematics