INFORMATIQUE
TECHNIQUE

Expert

Angular

Développez vos applications web avec le framework JavaScript de Google

Daniel DJORDJEVIC
William KLEIN
Sébastien OLLIVIER

Microsoft® Most Valuable Professional

eni

Avant-propos

Chapitre 1
Introduction

Chapitre 2
Ma première application

Chapitre 3
Fondamentaux d'Angular

Chapitre 4
TypeScript

Chapitre 5
Angular CLI

Chapitre 6
Les composants

Chapitre 7
Les services

Chapitre 8
L'injection de dépendances

Chapitre 9
Le requêtage HTTP

Chapitre 10
Les interactions utilisateur

Chapitre 11
Les formulaires

Chapitre 12
Le routage

Chapitre 13
Les directives

Chapitre 14
Tester son application

Chapitre 15
Le cross-platform avec Angular

Chapitre 16
Pour aller plus loin

Chapitre 5
Angular CLI

1. Introduction

L'objectif de ce chapitre est de présenter la CLI développée par les équipes d'Angular. En passant par la définition de celle-ci et les différentes commandes qu'Angular CLI apporte, le but est de pouvoir se lancer dans un projet Angular et de voir les bénéfices que la CLI peut apporter.

2. Qu'est-ce qu'Angular CLI ?

2.1 La définition

Angular CLI est une *Command Line Interface* (interface en ligne de commande, en français) développée par les équipes d'Angular même. Cette CLI permet de créer des projets dans lesquels la CLI pourra ajouter des fichiers et plus exactement des entités Angular. Il sera possible d'ajouter des modules, des composants, des services ou bien des directives en une ligne de commande.

Mais ce n'est pas tout, un projet créé avec Angular CLI est configuré par défaut pour fonctionner avec de nombreuses tâches transverses qu'implique une application web TypeScript. Nous parlons ici de bundling des sources, de minification, mais aussi des outils qui permettent de tester son application ou même de la déployer.

Remarque

Le bundling est l'opération qui consiste à mettre en commun les différentes parties de l'application afin de constituer un seul paquet final : un bundle. La minification quant à elle, a pour objectif de réduire la taille du code source en renommant les variables et fonctions par des noms plus courts, sans pour autant avoir un impact sur le fonctionnement du code.

Remarque

Historiquement, Angular CLI a migré depuis SystemJS vers Webpack pour la partie module loader, tout en bénéficiant de tout ce que permet de faire Webpack en tant que Task Runner, cela veut dire que les tâches telles que le bundling sont faites avec Webpack.

Pour les développeurs web débutants, ou du moins n'ayant pas une bonne maîtrise de Webpack, il est assez intéressant d'avoir une build déjà prête qui fait tout le travail, et qui permet ainsi au développeur de se concentrer sur l'essentiel : son code et son application. D'un autre point de vue, cela implique que la configuration Webpack est une boîte noire pour le développeur, avec tout de même des points d'extensions, limités, qui sont dans le fichier `.angular-cli.json`. Nous allons voir en détail, dans la suite du chapitre, ce que contient ce fichier.

Cependant, pour les développeurs un peu plus chevronnés, qui veulent ou ont besoin d'avoir la maîtrise sur la configuration Webpack, il est important de savoir que depuis la version 1.0.0-beta.3 de la CLI, il est possible d'éjecter cette configuration Webpack, pour ensuite pouvoir en faire ce que l'on veut.

S'il fallait résumer Angular CLI en une phrase : c'est un outil qui permet de générer et de piloter un projet Angular sans se soucier des problématiques transverses du développement web, avec tout l'outillage pour développer et déployer son application. Un projet peut alors être lancé très rapidement, en quelques secondes, avec une infrastructure toute prête, c'est un énorme gain de temps.

2.2 Les commandes principales

2.2.1 Créer un nouveau projet Angular CLI : ng new

Pour créer un nouveau projet Angular CLI, il suffit d'utiliser la commande `ng new`, en lui spécifiant le nom de l'application. Par défaut, l'application sera créée dans un répertoire du même nom.

Syntaxe de la création d'un nouveau projet Angular CLI

```
ng new <nom de l'application>
```

Remarque

Ce chapitre est conçu pour mettre en avant les commandes principales qui seront les plus utilisées par les développeurs pendant les phases de développement. Angular CLI possède une grande quantité de commandes avec chacune de nombreux paramètres. Afin de ne pas être trop redondant avec la documentation d'Angular CLI, pour avoir plus d'informations ou découvrir toutes les commandes, il est intéressant d'explorer la documentation sur Github : https://github.com/angular/angular-cli/

2.2.2 Compiler l'application : ng build

Pour compiler l'application, la simple commande de base `ng build` suffit.

Syntaxe pour compiler l'application

```
ng build
```

Les fichiers générés, appelés *build artifacts* de manière générale, sont placés dans le répertoire `dist/` à la racine du projet.

Angular CLI embarque un système de gestion de *build targets* et d'environnements. Cela permet au développeur de compiler en mode développement ou production (*build targets*) puis de choisir quel environnement il veut cibler. Par exemple, si l'application front Angular utilise une API, il est courant d'avoir une API de développement et une API de production.

Compiler en développement ou production

Par défaut, l'application est toujours compilée en mode développement. Par opposition, lorsque l'application est compilée en mode production, Angular CLI va optimiser les packages de l'application en utilisant des concepts tels que l'*uglifying* et le *tree-shaking*.

Le principe de l'*uglifying* vient principalement de la librairie **UglifyJS**, qui est une librairie JavaScript qui permet de minifier les fichiers. C'est un principe très courant dans le développement web qu'il ne faut pas négliger, l'objectif étant d'enlever les caractères qui sont inutiles au bon fonctionnement du code. Cela permet de réduire la taille du bundle final et d'offrir aux utilisateurs des temps de chargement plus rapides.

Quant au *tree-shaking*, l'objectif est d'enlever le code qui n'est pas utilisé. Pour cela, lors de la compilation, les exports qui ne sont pas utilisés dans le reste de l'application seront retirés du bundle final. Plus précisément, c'est grâce aux modules ES6 et les imports/exports, que le *tree-shaking* peut détecter les modules non utilisés, via les exports qui ne sont tout simplement pas référencés.

Pour compiler en mode développement ou production, il suffit de renseigner le paramètre `target`.

Syntaxes pour compiler dans un mode particulier

```
ng build –target=<mode>
ng build --<mode>
```

Exemples de commandes pour compiler en production

```
ng build –target=production
ng build --prod
```

Les fichiers d'environnements

À la création du projet, deux fichiers d'environnements sont créés : `environment.prod.ts` et `environment.ts`.

L'objectif est de définir les paramètres de l'application Angular qui sont spécifiques à chaque environnement. Le cas d'école est l'URL d'une API qui alimente l'application en données.

Les paramètres de base sont ceux qui sont présents dans le fichier `environment.ts`. Ensuite, en fonction de l'environnement ciblé par la compilation, le fichier sera remplacé par celui correspondant.

En regardant le contenu du premier fichier `environment.ts`, on peut voir une première propriété, un booléen `production` qui est à `false`, le tout dans une constante qui est exposée.

Contenu du fichier `environment.ts`

```
export const environment = {
  production: false
};
```

Quant au fichier `environment.prod.ts`, on retrouve la même structure, mais avec le booléen à `true`.

Ainsi, on peut ajouter des propriétés dans ces fichiers et les utiliser dans son application. Pour l'utiliser, il suffit d'importer la constante `environment`.

```
import { environment } from '../environments/environment';
```

Puis les propriétés sont accessibles tout naturellement.

```
    if(environment.production)
    {
      // Code spécifique à la production
    }
    if(!environment.production)
    {
      // Code à ne pas exécuter en production
    }
```

Pour choisir l'environnement visé, il suffit de compléter la commande `ng build` avec le paramètre `environment`.

Remarque

Il ne faut surtout pas mélanger le concept de target et d'environnement. La target permet à Angular CLI de savoir comment compiler l'application, c'est-à-dire d'activer ou non les optimisations de production par exemple. Tandis que l'environnement est uniquement une question de configuration, de paramétrage, au sein de l'application.

Syntaxe pour compiler en visant un environnement particulier

```
ng build -env=<environment>
```

Exemples de commandes pour viser l'environnement "prod"

```
ng build -env=prod
ng build -e=prod
```

Remarque

Si aucun environnement n'est spécifié, l'environnement choisi sera "dev" si l'application est compilée en développement (`--dev`) et "prod" si elle est compilée en production (`--prod`).

Extraire le style dans des feuilles de style séparées

Par défaut, la compilation injecte les feuilles de style globales de l'application dans le bundle JavaScript final. Il est possible de les extraire dans un fichier CSS dédié : `styles.bundle.css`.

Pour cela, il suffit d'utiliser le paramètre `extract-css`.

Exemples de commandes pour extraire le CSS global

```
ng build --extract-css
ng build --ec
```

Avoir plus d'informations sur la compilation

Par défaut, Angular CLI n'affiche qu'un résumé de ce qu'il se passe lors de la compilation, il est possible d'augmenter la verbosité de celle-ci grâce au paramètre `verbose`.

Exemples de commandes pour activer la verbosité

```
ng build --verbose
ng build -v
```

Remarque

Pour les développeurs plus expérimentés avec Webpack, ou tout simplement pour ceux qui sont désireux de prendre la main sur la configuration Webpack, afin d'avoir une maîtrise complète de la compilation et d'avoir toutes les informations nécessaires, il est possible d'éjecter la configuration Webpack. Cf. section Éjecter la configuration Webpack dans ce chapitre.