



Expert
EXPERT

Angular

Développez vos applications web
avec le framework JavaScript
de Google

Daniel DJORDJEVIC
William KLEIN



Microsoft®
Most Valuable
Professional

Sébastien
OLLIVIER

Avant-propos

1. Pourquoi cet ouvrage ?	13
2. À qui s'adresse cet ouvrage ?	14
3. Structure de l'ouvrage	14

Chapitre 1 Introduction

1. État des lieux du développement web	17
2. Angular	18
2.1 Organisation par composants	18
2.2 TypeScript	19
2.3 Les spécifications ES6	19
2.4 DOM Virtuel	19
2.5 Rendu côté serveur possible	20
3. AngularJS vs Angular	21

Chapitre 2 Ma première application

1. Introduction	23
2. Le setup	23
2.1 Installation de npm	23
2.2 Installation de TypeScript et initialisation du projet	25
2.3 Installation des dépendances Angular	26
3. Mon premier composant	28
4. Mon premier module	31
5. Lancement de l'application	32
6. Angular CLI à la rescousse	34

Chapitre 3 Fondamentaux d'Angular

1. Introduction	37
2. Les composants	37
3. Les templates	39
3.1 Binding	40
3.2 Événements JavaScript	42
3.3 Listes	43
3.4 Afficher/cacher un élément	45
3.5 Liens	48
3.6 Styles	49
3.7 Directives	53
4. Les décorations	55
5. Les pipes	56
5.1 Utiliser un pipe	56
5.2 Les pipes du framework	57
5.3 Créer un pipe	58
5.3.1 Les pipes de transformation	58
5.3.2 Les pipes filtres	60
5.4 Les pipes purs et impurs	61
6. Les modules	63
7. Observable et promise	64
8. Préparer l'application pour la production	66

Chapitre 4
TypeScript

- 1. JavaScript 67
- 2. TypeScript..... 69
 - 2.1 Syntaxe 70
 - 2.1.1 Variables 70
 - 2.1.2 Fonctions..... 71
 - 2.1.3 Classes..... 71
 - 2.1.4 Arrow Function 73
 - 2.1.5 Union type 74
 - 2.1.6 Interface 75
 - 2.1.7 Générique 76
 - 2.1.8 Les décorateurs 76
 - 2.2 Typings 77
 - 2.3 Compilation 78
 - 2.4 EcmaScript 6 et 7 dès maintenant..... 79
 - 2.5 Angular et TypeScript 80

Chapitre 5
Angular CLI

- 1. Introduction 81
- 2. Qu'est-ce qu'Angular CLI ? 81
 - 2.1 La définition 81
 - 2.2 Les commandes principales 83
 - 2.2.1 Créer un nouveau projet Angular CLI : ng new..... 83
 - 2.2.2 Compiler l'application : ng build 83
 - 2.2.3 Servir l'application avec ng serve..... 87
 - 2.2.4 Exécuter les tests avec Angular CLI..... 88
 - 2.3 Les commandes de scaffolding..... 91
 - 2.4 La génération d'un composant..... 92

2.5	De angular-cli à @angular/cli	99
2.5.1	Migrer	99
2.5.2	Les changements à effectuer	99
3.	Créer une application avec Angular CLI	102
3.1	Installer Angular CLI	102
3.2	Créer le socle initial	102
4.	Configurer un projet Angular CLI	106
4.1	Schéma du fichier de configuration d'Angular CLI	106
4.2	Gérer les assets lors de la compilation	110
4.3	Rajouter des environnements	113
4.4	Intégrer une librairie externe	114
4.5	Intégrer un préprocesseur CSS	115
4.6	Ajouter des entités de manière globale	116
4.6.1	Ajouter un script globalement	116
4.6.2	Ajouter une feuille de style globalement	117
4.7	Éjecter la configuration Webpack	119
5.	Compilation Ahead-Of-Time (AOT)	119

Chapitre 6

Les composants

1.	Introduction	123
2.	Qu'est-ce qu'un composant ?	123
2.1	Une première définition	123
2.2	Créer un premier composant	125
2.2.1	Syntaxe inline	125
2.2.2	Une réelle syntaxe, découpée en plusieurs fichiers	126
2.2.3	Utiliser un composant dans son application	127
2.3	Afficher les propriétés d'un composant	129

- 3. Les inputs et outputs 131
 - 3.1 Les inputs d'un composant 131
 - 3.1.1 Déclarer une variable en tant qu'Input 131
 - 3.1.2 Un exemple concret avec une liste 132
 - 3.1.3 Donner un nom personnalisé à son input..... 134
 - 3.2 Les outputs d'un composant 135
 - 3.3 Donner un nom personnalisé à son output 138
- 4. Interaction entre composants 138
 - 4.1 Passer une donnée du parent vers l'enfant à l'aide d'un input . 139
 - 4.2 Intercepter un changement de valeur à l'aide d'un setter 141
 - 4.3 Notifier le parent à l'aide d'un EventEmitter en output 143
 - 4.4 Observer les changements d'input avec ngOnChanges 145
 - 4.5 Utiliser une variable locale 148
- 5. Les décorateurs @ViewChild et @ViewChildren 150
 - 5.1 Récupérer une référence avec @ViewChild 150
 - 5.2 Récupérer plusieurs références avec @ViewChildren 152
- 6. Les composants Angular et la View Encapsulation 154
 - 6.1 Le Shadow DOM 154
 - 6.2 Spécifier une View Encapsulation 155
 - 6.3 Les types de View Encapsulation 156
 - 6.3.1 La View Encapsulation émulée 156
 - 6.3.2 La View Encapsulation native 159
 - 6.3.3 Aucune View Encapsulation 160

Chapitre 7
Les services

- 1. Introduction 163
- 2. Qu'est-ce qu'un service ? 163
- 3. Déclarer son service 164
- 4. Utiliser son service 164

5. Rendre son service asynchrone	166
5.1 Les promesses	166
5.2 Les observables	169
5.3 Que choisir ?	174
6. Notifier lorsque les données changent	175

Chapitre 8

L'injection de dépendances

1. Principe de base	181
2. Injection de dépendances dans Angular	182
2.1 Enregistrement global	183
2.2 Enregistrement dans un composant	184
2.3 Injecter une dépendance	185
3. Provider	186
3.1 useClass	186
3.2 useExisting	187
3.3 useFactory	189
3.4 useValue	192
3.5 OpaqueToken	192
3.6 Dépendance optionnelle	196
3.7 Injection restreinte	197
3.8 Restriction de l'enregistrement d'une dépendance	199

Chapitre 9

Le requêtage HTTP

1. Introduction	201
2. Obtenir et envoyer des données	201
3. Transformer des données	204

- 4. Communiquer de manière sécurisée 205
 - 4.1 Authentification basique 205
 - 4.2 Authentification Oauth 208
- 5. Simuler le requête HTTP 212

Chapitre 10

Les interactions utilisateur

- 1. Qu'est-ce que l'event binding ? 219
- 2. S'abonner à un événement. 220
- 3. Récupérer une entrée utilisateur 221
 - 3.1 Comment manipuler l'objet \$event ? 221
 - 3.2 Utiliser un typage fort pour \$event. 222
 - 3.3 Une alternative grâce à un template reference variable 223
 - 3.4 Utiliser un template reference variable avec un événement .. 224
 - 3.5 Filtrer les entrées utilisateur. 225

Chapitre 11

Les formulaires

- 1. Les formulaires basés sur un template 229
- 2. Créer un composant formulaire. 230
 - 2.1 Le composant 230
 - 2.2 La vue et le data binding 231
 - 2.2.1 La syntaxe ngModel 232
 - 2.2.2 ngModel en détail. 232
 - 2.2.3 L'utilisation de ngModel dans un cas concret 232
 - 2.3 Intégrer le formulaire dans l'application. 233
- 3. Les états et la validité d'un champ 234
 - 3.1 Les états d'un input 234
 - 3.2 Styliser selon la validité 236
- 4. Soumettre le formulaire 237

5. Les formulaires et les FormControls	238
5.1 Les contrôles et les groupes de contrôles.	238
5.2 Les validateurs intégrés	240
5.3 Créer un validateur personnalisé.	241
5.4 Les validateurs asynchrones.	242

Chapitre 12

Le routage

1. Introduction	245
2. Définir les routes d'une application	245
3. Le rendu de composant	247
4. Naviguer dans son application.	249
5. Récupération des données de routage	250
6. Outlet nommé	254
6.1 Définir des outlets nommés.	254
6.2 Naviguer avec des outlets nommés	256
7. Resolver	257

Chapitre 13

Les directives

1. Introduction	263
2. Qu'est-ce qu'une directive?	263
2.1 Introduction	263
2.2 Directives communes.	264
2.2.1 NgIf	264
2.2.2 NgFor.	264
2.2.3 NgStyle	265
2.2.4 NgClass	265

3. Les directives d'attribut	266
3.1 Créer une directive d'attribut	266
3.2 Interagir avec les événements du DOM	268
3.3 Passer des valeurs aux directives d'attribut.	269
4. Les directives structurelles	272
4.1 La balise <template> et l'astérisque.	272
4.2 Créer une directive structurelle	273

Chapitre 14

Tester son application

1. Introduction	279
2. Les tests unitaires	279
2.1 Introduction aux tests avec Jasmine	280
2.2 Exécuter du code avant ou après chaque test	283
2.3 Les matchers mis à disposition	284
2.3.1 Comment utiliser un matcher ?	285
2.3.2 Des exemples de matchers	285
2.3.3 Négation d'un matcher	288
2.4 Créer un matcher personnalisé	289
2.4.1 Créer une librairie de matchers personnalisés.	289
2.4.2 Utiliser un matcher personnalisé.	291
2.5 Les composants	292
2.5.1 Le composant à tester.	292
2.5.2 Le TestBed	293
2.5.3 Vérifier que le composant est bien instancié	294
2.5.4 Contrôler les propriétés du composant	295
2.5.5 S'assurer que le rendu du composant soit cohérent	295
2.6 Les services	297
2.6.1 Le service à tester	297
2.6.2 Tester le service.	297
2.6.3 S'assurer que le service est bien injecté	298

2.7	Les directives	299
2.7.1	La directive à tester	299
2.7.2	Tester la directive	300
2.8	Injecter un faux service	302
3.	Les tests e2e	303
3.1	Lancer les tests e2e	304
3.2	Écrire un test e2e	305
3.2.1	Interagir avec le navigateur	306
3.2.2	Récupérer un élément du DOM et interagir avec	307
3.2.3	Interagir avec les éléments DOM	308
3.2.4	Tester le composant	310
3.3	Écrire un test e2e, en mieux	311

Chapitre 15

Le cross-platform avec Angular

1.	Apache Cordova	315
1.1	Créer un projet Apache Cordova	316
1.2	Angular dans Apache Cordova	318
1.3	Utiliser un plug-in Apache Cordova	324
2.	Ionic 2	329
2.1	Créer un projet Ionic	329
2.2	Utiliser les composants Ionic	330
2.3	Amorcer une application Ionic	334
2.4	Naviguer dans une application Ionic	337
2.5	Utiliser un plug-in avec Ionic	339
3.	NativeScript	342
3.1	Créer une application avec NativeScript	343
3.2	Utiliser les composants NativeScript	344
3.3	Amorcer une application NativeScript	348
3.4	Naviguer dans une application NativeScript	349
3.5	NativeScript et Web	352

Chapitre 16
Pour aller plus loin

- 1. Introduction 355
- 2. Rendu côté serveur 355
 - 2.1 Principe de la mise en place 356
 - 2.2 Création du module serveur 357
 - 2.3 Compilation AOT 359
 - 2.4 Serveur Node.js 361
 - 2.5 Compilation et exécution 369
- 3. La détection de changement 371
 - 3.1 Pourquoi la détection de changement ? 371
 - 3.2 Les Zones, ou comment notifier Angular ? 374
 - 3.2.1 Le comportement de la détection de changement 376
 - 3.2.2 L’immuabilité et la stratégie
de détection de changement OnPush 377
 - 3.3 Encore plus de contrôle sur la détection de changement 380
- 4. Le cycle de vie d’un composant 381
 - 4.1 La présentation des lifecycle hooks 382
 - 4.1.1 Les différents hooks 382
 - 4.1.2 Utiliser un lifecycle hook 383
 - 4.2 Le cycle de vie d’un composant 383
 - 4.2.1 Le constructeur 386
 - 4.2.2 ngOnInit 387
 - 4.2.3 ngOnChanges 387
 - 4.2.4 ngDoCheck 388
 - 4.2.5 ngAfterContentInit 389
 - 4.2.6 ngAfterContentChecked 390
 - 4.2.7 ngAfterViewInit 390
 - 4.2.8 ngAfterViewChecked 391
 - 4.2.9 ngOnDestroy 391
- Index 393

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.