



Art
EXPLORE

Angular et Node.js

Optimisez le développement
de vos applications web
avec une architecture **MEAN**

Fichiers complémentaires
à télécharger



Pierre POMPIDOR

Les éléments à télécharger sont disponibles à l'adresse suivante :

<http://www.editions-eni.fr>

Saisissez la référence ENI de l'ouvrage **EIANGNOD** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Avant-propos

Chapitre 1 Introduction

- 1. Introduction 13
- 2. L'architecture MEAN pour une application web 15
 - 2.1 Le principe des applications monopages
(single page applications) 16
 - 2.2 Le paradigme de conception modèle-vue-contrôleur 16
- 3. Angular au centre de l'architecture MEAN 19
- 4. Présentation du fil rouge : une application d'e-commerce 21

Chapitre 2 Le langage JavaScript

- 1. Introduction à JavaScript 25
 - 1.1 Bref historique 25
 - 1.2 Panorama de l'utilisation de JavaScript 26
 - 1.3 Les bibliothèques et les frameworks applicatifs JavaScript ... 27
- 2. Où coder du code JavaScript ? 28
- 3. Les outils du navigateur et le débogage 29
- 4. Les éléments de programmation de base 30
 - 4.1 Les variables 30
 - 4.1.1 Les types internes 30
 - 4.1.2 Le transtypage 31

2 ————— Angular et Node.js

Optimisez le développement de vos applications web

4.2	Les structures de données usuelles	32
4.3	Application d'expressions régulières	34
4.4	Les blocs d'instructions	35
4.5	Les structures conditionnelles	35
4.5.1	La structure if ... else	35
4.5.2	La structure switch d'aiguillage multiple	36
4.6	Les structures itératives	36
4.6.1	Les structures itératives avec indices de boucle	36
4.6.2	Les structures itératives sans indices de boucle	37
5.	La programmation fonctionnelle en JavaScript	39
5.1	Une fonction passée en paramètre (fonction de callback)	39
5.1.1	Exemple avec la méthode forEach()	39
5.1.2	Exemple avec la méthode map()	40
5.2	Une fonction retourne une fonction (factory)	41
6.	La programmation objet avec JavaScript	42
6.1	Les principes de la programmation objet avec JavaScript	42
6.2	Les objets littéraux	43
6.3	L'héritage par chaînage de prototypes	44
6.3.1	La propriété __proto__ de l'objet héritant	44
6.3.2	La propriété prototype	45
6.4	La création d'un objet par l'appel d'une fonction constructrice	45
6.5	Exemples d'implémentations d'une méthode	47
6.6	La problématique de l'objet courant (this)	47
6.7	L'héritage	50
6.8	Le chaînage de méthodes	51
7.	Les principaux apports de la norme ECMAScript 6	52
7.1	La norme ECMAScript	52
7.2	Le mot réservé let	52
7.3	L'interpolation de variables dans les chaînes	53
7.4	Les paramètres par défaut	53

7.5	Une manipulation plus confortable des listes.	53
7.5.1	La structure for (... of ...)	53
7.5.2	La méthode includes()	54
7.6	L'opérateur « fat arrow » (= >)	54
7.7	Les classes	55
8.	La programmation réactive, les observables et les promises	55
8.1	Premier exemple : un observable sur un bouton.	57
8.2	Deuxième exemple : un observable sur un entier.	58
8.3	Troisième exemple : un observable sur un timer	59
8.4	Les promises	59

Chapitre 3

Les extensions à JavaScript pour utiliser des classes

1.	Présentation des extensions à JavaScript	61
2.	Le langage TypeScript	62
2.1	Le transpiler tsc.	62
2.2	Typage statique des variables	63
2.2.1	Les types de base	63
2.2.2	Typage de variables non scalaires.	63
2.2.3	Le type enum	64
2.2.4	Le type générique any	64
2.2.5	Typage des fonctions.	64
2.3	Les classes	65
2.3.1	L'héritage	66
2.3.2	Les interfaces	67
2.3.3	La généricité	67
3.	Le langage Dart	68
3.1	Installation et test de Dart.	68
3.2	Génération du code JavaScript avec Dart	69
3.3	Les classes	69
3.3.1	L'héritage	70
3.3.2	Les interfaces	71

4 _____ Angular et Node.js

Optimisez le développement de vos applications web

Chapitre 4

La plateforme Node.js

1. Présentation de Node.js	73
2. Installation et test de Node.js	74
2.1 Création du fichier de test	74
2.2 Installation et test de Node.js sous Ubuntu	75
2.3 Installation et test de Node.js sous Windows	75
2.4 Installation et test de Node.js sous macOS	76
3. La modularité de Node.js	76
3.1 Les modules et les packages	76
3.1.1 Le gestionnaire de modules de Node.js : npm	77
3.1.2 Spécification des dépendances : le fichier package.json	77
3.2 Création d'un premier serveur Node.js de test	78
3.3 Création et réutilisation d'un module	80
3.4 Création d'un serveur renvoyant des données	82
3.5 Le module express	83
3.5.1 Gestion de routes REST avec le module express	83
3.5.2 Gestion des templates avec le module express	86
3.5.3 Spécification des dépendances dans un fichier package.json	88
3.5.4 Installation du module express	88
3.6 Le module fs (FileSystem)	88
3.7 Test d'un serveur Node.js	90
3.7.1 Création d'un fichier de données JSON	90
3.7.2 La problématique du contrôle d'accès HTTP	91
3.7.3 Renvoi au client d'un fichier JSON	92
3.7.4 Paramétrage des routes	93
3.7.5 Gestion des paramètres	98
4. Sécurisation d'un serveur Node.js (protocole HTTPS)	100
4.1 Création de l'autorité de certification	100
4.2 Création du certificat	101
4.3 Création du serveur	101

5. Bilan des acquis de ce chapitre.	102
--	-----

Chapitre 5

Le SGBD NoSQL MongoDB

1. Introduction	105
2. Pourquoi utiliser une base de données NoSQL ?	106
3. Présentation de MongoDB.	107
3.1 Les collections et les documents	107
3.2 Les index	108
4. Mise en œuvre de MongoDB	108
4.1 Installation de MongoDB.	108
4.1.1 Installation de MongoDB sous Linux.	108
4.1.2 Installation de MongoDB sous Windows ou sous macOS	109
4.1.3 Utilisation de MongoDB en lignes de commande.	109
4.2 Affichage de la liste des bases de données.	110
4.3 Création d'une base de données.	110
4.4 Affichage de la liste des collections	110
4.5 Création d'une collection	110
4.5.1 Insertion des documents dans une collection	111
4.5.2 Importation de documents à partir d'un fichier	111
4.5.3 Exportation des documents d'une collection dans un fichier JSON.	112
4.6 Interrogation d'une collection	112
4.6.1 Interrogation via un objet « filtre ».	112
4.6.2 Les opérateurs de comparaison, les opérateurs ensemblistes et logiques	114
4.6.3 L'opérateur \$exists.	114
4.6.4 L'opérateur \$in.	115
4.6.5 L'opérateur \$nin.	115
4.6.6 L'opérateur \$or.	115
4.6.7 L'opérateur \$not	115

6 ————— Angular et Node.js

Optimisez le développement de vos applications web

4.6.8	L'opérateur \$nor	116
4.7	Application d'expressions régulières	116
4.8	Les projections et la méthode distinct()	116
4.8.1	Les projections	116
4.8.2	La méthode distinct()	117
4.9	Référencement des documents et jointures	118
4.9.1	Les objets imbriqués (nested objects)	119
4.9.2	Les objets référencés	119
4.9.3	Les jointures	120
4.10	Mise à jour et suppression d'un document	124
4.10.1	Mise à jour d'un document	124
4.10.2	Suppression d'un document	125
4.11	Suppression d'une collection	125
5.	Utilisation de MongoDB via Node.js	125
5.1	Installation du module MongoDB pour Node.js	125
5.2	Connexion au serveur MongoDB	127
5.3	Insertion de données à partir d'un serveur Node.js	128
5.4	Interrogation de données à partir d'un serveur Node.js	129
5.4.1	Exploitation du résultat de la méthode find()	129
5.4.2	Utilisation de la méthode toArray()	131
5.5	Synchronisation des requêtes	132
5.5.1	Utilisation des fonctions de callback	133
5.5.2	Utilisation du module async	135
5.5.3	La méthode async.series()	137
5.5.4	La méthode async.waterfall()	138
6.	Interrogation de MongoDB via les routes gérées par express	139
6.1	La structure d'un serveur Node.js interrogeant MongoDB ...	139
6.2	La problématique du cross-origin resource sharing	140
6.3	Exemples de gestion de routes	141
6.3.1	Gestion d'une route pour lister les marques	142
6.3.2	Gestion d'une route pour filtrer les produits	142
6.3.3	Recherche d'un produit à partir de son identifiant interne	144

7. Le fil rouge du côté serveur	145
7.1 Création de la collection.	145
7.2 Mise en place de deux recherches sur les produits	147
7.2.1 La superstructure du serveur.	148
7.2.2 Gestion de la route qui filtre les documents sur différents critères.	149
7.2.3 Gestion de la route qui renvoie un document via son identifiant interne.	150
7.2.4 Exemples de requête sur le serveur.	150
8. Bilan des acquis de ce chapitre.	151

Chapitre 6

Introduction au framework applicatif Angular

1. Présentation d'Angular.	153
1.1 Une évolution radicale d'AngularJS.	153
1.2 La modularité de l'application : les modules et les composants	154
1.2.1 Les modules	155
1.2.2 Les composants et les services	156
1.3 Manipulation des composants comme des balises.	157
1.4 Utilisation d'une extension à JavaScript (TypeScript ou Dart)	158
2. Angular par rapport au framework MVC (voire MVVM).	158
3. Mise en place d'une application Angular	160
3.1 Présentation d'Angular CLI	161
3.2 Installation d'Angular CLI	162
3.3 Création d'un projet Angular avec Angular CLI.	162
3.4 Structure des dossiers d'un projet Angular CLI	164
3.5 Un premier composant créé par Angular CLI.	165
3.6 Le root module créé par Angular CLI	167
4. Les décorateurs	168

8 _____ Angular et Node.js

Optimisez le développement de vos applications web

5. Création d'un nouveau composant qui affiche un message	170
5.1 Création du composant	170
5.1.1 Le template HTML	171
5.1.2 La feuille de style	172
5.2 Interfaçage du composant dans le composant racine	172
5.3 Spécification des composants dans le module	172
5.4 Activation du module.	173
5.5 La page web frontale	173
6. Le cycle de vie d'un composant	174
6.1 Le hook ngOnChanges()	175
6.2 Le hook ngOnInit()	176
6.3 Le hook ngDoCheck()	177
6.4 Le hook ngOnDestroy()	178
7. Bilan des acquis de ce chapitre.	178

Chapitre 7

Angular : les templates, bindings et directives

1. Les templates	181
1.1 Imbrication des templates	182
1.2 Les templates insérés (embedded templates)	188
1.3 Les templates externalisés	189
2. Data bindings entre le composant et le template.	189
2.1 Accès aux éléments du DOM	190
2.2 Interpolation d'une variable dans un template.	190
2.3 Property binding	192
2.4 Event binding	194
2.5 Two-way data binding.	196
3. Les directives	198
3.1 Les directives structurales	199
3.1.1 La directive *ngFor	200
3.1.2 La directive *ngIf	202

3.2	Les directives attributs	203
3.3	Émission d'information d'un composant vers son père (@Output()).	208
4.	Les pipes	211
5.	Exemple de synthèse : un formulaire d'authentification	215
6.	Le fil rouge : création d'un composant qui affiche les produits	218
6.1	Le composant	219
6.1.1	La classe implémentant le composant	220
6.1.2	Le template HTML	221
6.1.3	La feuille de style	223
6.2	Le module spécifiant le composant	223
6.3	Activation du module.	224
6.4	La page web frontale	224
6.5	Lancement de l'application	225
7.	Bilan des acquis de ce chapitre.	225

Chapitre 8

Angular et la connexion à Node.js : les services

1.	Introduction	227
2.	Injection de dépendances	228
3.	Utilisation des services pour le transfert de données	229
3.1	Récupération de données formatées en JSON	229
3.2	Envoi de données JSON au serveur	230
3.3	Envoi de données via la querystring	231
4.	Mise en œuvre des services dans le fil rouge.	232
4.1	Déclaration des routes du côté serveur	232
4.2	Gestion des produits	235
4.2.1	Affichage des sélecteurs.	235
4.2.2	Affichage des produits suivant des critères de recherche	238
4.2.3	Affichage des produits associés à des mots-clés.	240

10 _____ Angular et Node.js

Optimisez le développement de vos applications web

4.2.4	Accès à un produit par son identifiant	243
4.3	Gestion du panier	244
4.3.1	Affichage des identifiants des produits du panier	244
4.3.2	Affichage de tous les produits du panier	245
4.3.3	Ajout d'un produit au panier	247
4.3.4	Suppression d'un produit du panier	250
4.3.5	Réinitialisation du panier	252
5.	Bilan des acquis de ce chapitre	252

Chapitre 9

Angular et la gestion des routes internes

1.	Principe général du routage	255
1.1	Pourquoi mettre en place un routage ?	255
1.2	Les routes, le routeur, les tables de routage	257
1.3	Les vues activées par les routes	259
1.4	Exemple de routage	260
1.5	Définition d'un arbre de vues	264
1.6	Utilisation des outlets nommées	269
2.	La syntaxe des routes	273
2.1	Les deux syntaxes d'une route : chaîne ou link parameters array	274
2.2	Les routes absolues et les routes relatives	275
2.3	Paramétrage des routes	276
2.4	Association d'une route à une auxiliary outlet	277
3.	Sélection des routes	278
3.1	La directive routerLink	278
3.2	La méthode navigate()	279
3.3	Exemple de route	280
4.	Gestion des routes du contrôleur vers le composant cible	281
4.1	Configuration de la table de routage	281
4.2	Les propriétés d'une route	283

4.3	Prise en charge d'une route par plusieurs modules/tables de routage	284
4.4	Contrôle des routes : les guards	287
4.5	Invocation d'un composant	295
4.6	Capture d'une route lors de l'invocation d'un composant	297
5.	Gestion de routes dans le fil rouge	300
5.1	Le module de routage associé au root module	301
5.2	Le module de routage associé au feature module research	302
5.3	Le module de routage associé au feature module cart	303
6.	Bilan des acquis de ce chapitre	304

Chapitre 10

Angular et la visualisation d'informations

1.	Introduction	307
2.	Création de charts avec D3.js et dc.js	308
2.1	Installation de D3.js	308
2.2	Le langage SVG	309
2.3	Génération d'éléments graphiques associés aux objets d'une collection	312
2.4	Sélection et modification d'éléments du DOM	313
2.4.1	Ajout d'éléments graphiques	313
2.4.2	Remplacement d'éléments graphiques	315
2.5	Mise en œuvre d'écouteurs d'événements	316
2.6	Intégration de D3.js dans Angular	316
2.6.1	Un serveur virtuel de données commerciales	317
2.6.2	Le service accédant aux données du serveur	318
2.6.3	Le template du composant qui affiche un histogramme	319
2.6.4	Implémentation du composant qui affiche un histogramme	319

12 _____ Angular et Node.js

Optimisez le développement de vos applications web

2.7	Les bibliothèques dc.js et Crossfilter	324
2.7.1	Installation de dc.js et de Crossfilter	325
2.7.2	Implémentation du composant affichant l'histogramme	325
3.	Intégration de cartes Google Map dans un projet Angular	327
3.1	Installation des prérequis techniques	328
3.1.1	Installation des types TypeScript	328
3.1.2	Installation de la bibliothèque PrimeNG	329
3.2	Présentation d'une carte Google Map statique	330
3.3	Un composant PrimeNG pour gérer une carte Google Map . .	332
3.4	Gestion d'un chart associé à une Google Map	336
3.5	Autre exemple de composant PrimeNG (Calendar)	341
4.	Bilan des acquis de ce chapitre	343

Chapitre 11

Test et déploiement

1.	Test	345
2.	Déploiement	349
2.1	Déploiement avec Apache	349
2.2	Déploiement avec Node.js	350
2.3	Déploiement avec http-server (raccourci sur Node.js)	351
3.	Pour aller plus loin	352

Index	353
-----------------	-----

Chapitre 4

La plateforme Node.js

1. Présentation de Node.js

Node.js est un environnement permettant d'exécuter du code JavaScript hors d'un navigateur. À l'heure de la rédaction de cet ouvrage, il repose sur le moteur JavaScript V8 développé par Google pour ses navigateurs Chrome et Chromium.

Son architecture est modulaire et événementielle. Il est fortement orienté réseau en possédant pour les principaux systèmes d'exploitation (Unix/Linux, Windows, macOS) de nombreux modules réseau (dont voici les principaux par ordre alphabétique : DNS, HTTP, TCP, TLS/SSL, UDP). De ce fait, il remplace avantageusement, dans le cadre qui nous intéresse ici (c'est-à-dire la création et la gestion d'applications web), un serveur web tel qu'Apache.

Créé par Ryan Lienhart Dahl en 2009, cet environnement est devenu rapidement très populaire pour ses deux qualités principales :

- Sa légèreté (en corollaire de sa modularité).
- Son efficacité induite par son architecture monothread (en corollaire de la gestion événementielle que propose nativement l'environnement JavaScript).

Intégrer Node.js dans le développement d'applications web participe donc à la logique actuelle de rendre les opérations d'accès aux données les moins bloquantes possible (pour dépasser la problématique dite du « bound I/O » selon laquelle, avant toute autre cause, la latence globale d'une application est due au temps de latence des accès aux données).

Node.js permet donc, pour les applications web, de créer des serveurs extrêmement réactifs.

Dans ce qui suit, vous allez :

- Installer et tester Node.js sous Linux, Windows ou macOS.
- Créer un serveur HTTP renvoyant une chaîne de caractères.
- Mettre en œuvre un module.
- Créer un serveur HTTP utilisant le module express invoqué sur une route REST et renvoyant des données formatées en JSON, d'abord en totalité, puis filtrées sur une propriété.

Dans ce chapitre, nous n'introduirons que quelques modules (et fonctions) de Node.js.

La documentation complète des modules est disponible à cette adresse : <https://nodejs.org/api/>

2. Installation et test de Node.js

2.1 Création du fichier de test

Pour tester Node.js, vous allez dans un premier temps créer un code JavaScript qui va être le plus simple possible, et le faire exécuter par Node.js.

■ Créez donc le fichier `testDeNode.js` qui ne comprend qu'une ligne de code :

```
■ console.log("Test de Node");
```

2.2 Installation et test de Node.js sous Ubuntu

- Pour installer Node.js sous Ubuntu, le plus simple est d'utiliser la commande `curl` et le gestionnaire de paquets en ligne de commande (`apt-get`) :

```
curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -  
sudo apt-get install nodejs
```

- Un lien symbolique nommé `node` peut être créé pour lancer plus naturellement vos serveurs :

```
sudo ln -s /usr/bin/nodejs /usr/bin/node
```

- Ouvrez un terminal (shell) et exécutez le fichier de test :

```
node testDeNode.js
```

La chaîne de caractères « Test de Node » s'affiche !

Une procédure complète est en ligne sur <http://doc.ubuntu-fr.org/nodejs>.

2.3 Installation et test de Node.js sous Windows

L'installation de Node.js et son test sous Windows vont se dérouler en quatre étapes :

- Téléchargez l'installateur Windows Installer en vous rendant sur le site officiel de Node.js : <https://nodejs.org/en/download>
- Exécutez l'installateur (le fichier .msi précédemment téléchargé) en acceptant les conditions d'utilisation et le paramétrage par défaut.
- Redémarrez votre ordinateur.
- Ouvrez l'invite de commandes et exécutez le fichier de test :

```
node testDeNode.js
```

La chaîne de caractères « Test de Node » s'affiche !

2.4 Installation et test de Node.js sous macOS

L'installation de Node.js et son test sous macOS vont se faire en trois étapes :

▣ Téléchargez le package d'installation pour macOS (Macintosh Installer) en vous rendant sur le site officiel de Node.js : <https://nodejs.org/en/download/>

▣ Ouvrez un terminal et installez le package :

```
■ pkg install nomPackage.pkg
```

▣ Exécutez le fichier de test :

```
■ node testDeNode.js
```

La chaîne de caractères « Test de Node » s'affiche !

3. La modularité de Node.js

3.1 Les modules et les packages

Une des principales forces de Node.js est d'être modulaire (et notamment de proposer de nombreux modules réseau). Si certains de ces modules sont installés directement en même temps que Node.js, la plupart doivent être installés à la demande.

Lors de la création d'une application qui exige l'installation de modules, deux méthodes sont possibles pour effectuer celle-ci :

- Directement avec le gestionnaire de modules npm (et son option `install`).
- Indirectement (mais toujours avec npm) via la spécification des dépendances de l'application (c'est-à-dire des modules nécessaires à celle-ci) dans un fichier nommé `package.json`.

Un module est utilisé dans une application avec la fonction `require()` :

```
■ var moduleDansVotreApplication = require('<nomDuModule>');
```

Votre application Node.js peut être elle-même réutilisée comme module sous certaines conditions qui seront présentées ultérieurement.

Attardons-nous sur un point un peu subtil : la distinction entre modules et packages.

Les modules sont les briques conceptuelles d'une application Node.js. Un module peut être organisé en plusieurs codes JavaScript et dépendre d'autres modules. Ainsi, toutes les ressources nécessaires à un module (les codes, le fichier `package.json` spécifiant ses dépendances...) sont regroupées dans un package qui, de fait, est un dossier.

Donc, si les deux termes sont quasiment interchangeables, le terme « module » renvoie plus à la fonctionnalité globale, et « package » au dossier et à l'organisation des fichiers de code qui se trouvent dans celui-ci.

3.1.1 Le gestionnaire de modules de Node.js : npm

npm (*Node.js Package Manager*) est le gestionnaire de modules de Node.js (il est installé avec celui-ci).

Les modules sont installés globalement dans le dossier `node_modules`, situé au niveau des répertoires système si l'option `-g` est utilisée :

```
■ npm install -g <module>
```

ou sinon (sans l'option `g`) dans le répertoire courant (mais également dans un dossier nommé `node_modules`).

3.1.2 Spécification des dépendances : le fichier `package.json`

Pour spécifier les dépendances nécessaires à la création d'une application Node.js (c'est-à-dire les modules associés aux packages nécessaires à celle-ci), il est recommandé de créer un fichier nommé `package.json`.

Dans le contexte d'un fichier `package.json`, nous ne parlerons plus que de packages (et non de modules).

Voici un schéma minimal de ce fichier :

```
■ {  
  "name":      "<nom de l'application>",  
  "version":   "<version de l'application>",  
  "description": "<description de l'application>",  
  "author":    "<nom de l'auteur de l'application>",  
  "main":      "<code à exécuter comme point d'entrée>",  
}
```

```
"scripts": {  
  "start": "node <code à exécuter>"  
},  
"dependencies": {  
  "<nom du package>": "<version minimale du package à installer>",  
  ...  
}  
}
```

Pour installer les modules nécessaires, la commande suivante doit être exécutée :

```
■ npm install
```

Et voici celle qui va lancer le serveur :

```
■ npm start
```

Pour créer un squelette de fichier *package.json*, utilisez la commande suivante :

```
■ npm init --yes
```

Dans ce cas, la valeur de la propriété *main* est initialisée à *index.js*. Expliquons un peu l'intérêt de cette propriété :

Si votre application devient un package (comprenant un ou plusieurs codes réutilisables), la propriété *main* désigne le code qui est le point d'entrée dans le package lors de l'exécution de l'instruction `require()`.

3.2 Création d'un premier serveur Node.js de test

Vous allez écrire votre premier serveur (le bien classique « Hello World ») en utilisant le module HTTP qu'offre Node.js.

■ Saisissez le code de ce serveur dans le fichier *helloAvecNode.js* :

```
var http = require('http');  
  
var server = http.createServer(function(request, response) {  
  response.end('Hello World de Node.js');  
});  
  
server.listen(8888);
```