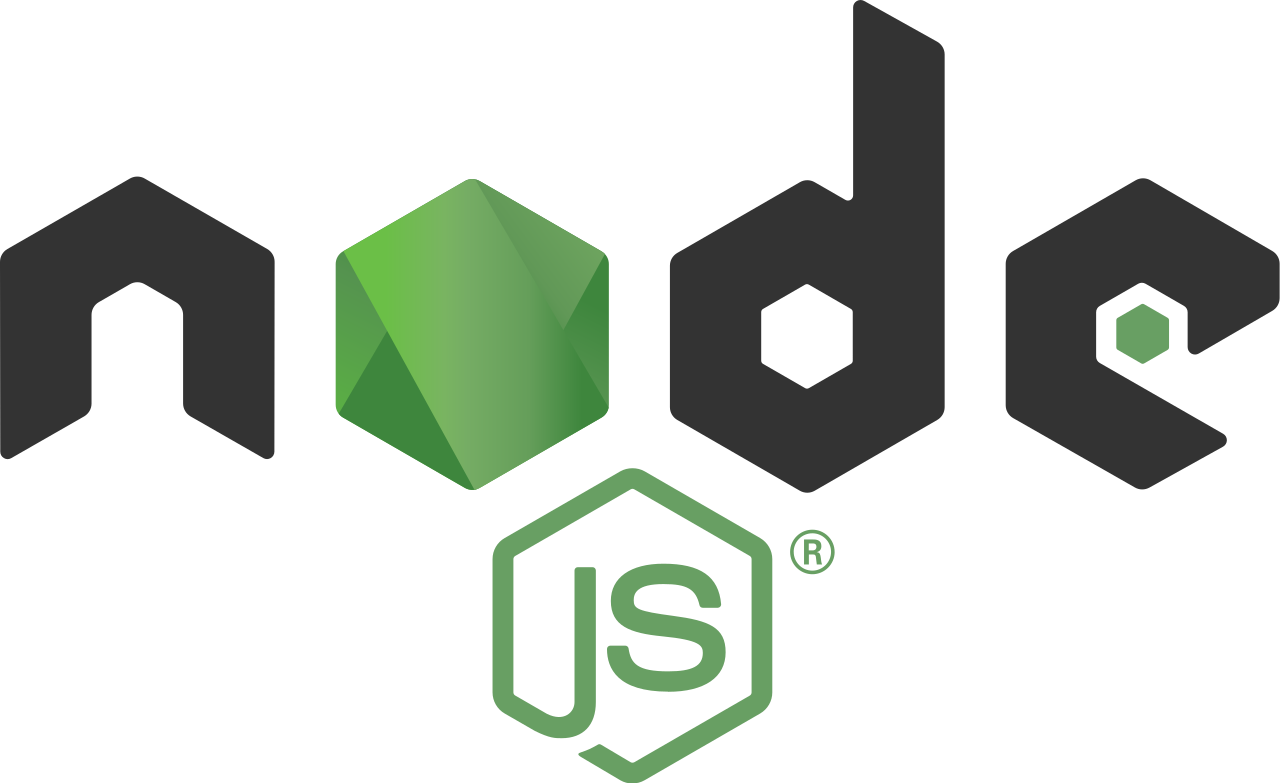
Devenir un(e) développeur(se) Node.js 🐢🚀

[Thomas GENTILHOMME](https://www.linkedin.com/in/thomas-gentilhomme/)

Ce document a pour objectif de livrer mon expérience **personnelle** et mes ressources à celles et ceux qui souhaiteraient devenir des développeurs back-end [Node.js](https://nodejs.org/en/) 😎.

Mon expérience et mes opinions vont fortement influencer l’écriture du document 👿 (je vous invite donc à conserver un fort esprit critique envers moi pour vous construire votre propre chemin 🐤).

Ce document n’est pas figé et il évoluera autant que je l’estime nécessaire à travers le temps.



[**👋 Introduction**](#_skfzaqm4bpcq) **6**

[👀 Qui suis-je ?](#_s74isbmdq54i) 6

[👊 Pour qui ?](#_jvqvaxy9igdv) 7

[💬 Pourquoi ?](#_vgjrbqp4o5m1) 7

[😬 L’investissement nécessaire](#_mhnhxy9x4co) 8

[📌 Légendes](#_o57wl26w1zjr) 9

[**JavaScript**](#_85cycxx6soor) **10**

[🌞 Introduction](#_ip5btyao4tt2) 10

[📕 Dédier un temps quotidien à la lecture](#_nngex2tunbh8) 11

[📖📐 La pratique avec un livre](#_2hqphqsg548u) 11

[💪 Mettez-vous au défi régulièrement](#_5o19o69xm15h) 12

[🏫 Nodeschool](#_70tu7zxaybuj) 13

[🔎 Les autres sites](#_idbczuen5sg8) 14

[🏆 Competitive programming](#_85pn8wl66159) 14

[🌌 Cours en ligne, talks et articles](#_bggntvb32asd) 15

[Udemy vs Frontend Masters](#_d2svs7iw0cq1) 15

[📄 Articles](#_b68eul3t1768) 16

[Axel Rauschmayer](#_9wppjvt7akbx) 16

[Jake Archibald](#_2568phvoygkt) 17

[Surma](#_1i5f2llnk77y) 17

[Lydia Hallie](#_p7rnqk1sd9am) 18

[Autres articles](#_pe4cakr0904a) 18

[📺 Talks](#_affq6lkoxklr) 20

[🔧 Engine & VM JavaScript (V8 etc)](#_c4fb08fvwed4) 21

[😡 La dangereuse mode des benchmarks](#_57yg6ym1riv6) 22

[**⚡ ECMAscript**](#_5mpvi0tkahty) **23**

[JavaScript ou ECMAScript ?](#_jkoo94l63ofs) 23

[ECMAScript 6 (ES2015)](#_x6bfrn1qa9lu) 23

[JavaScript: 20 ans 🎉](#_aib3h4icx00p) 23

[TC39](#_qtn5509xwnz3) 24

[**TypeScript**](#_9a7wczk5sc5q) **25**

[Introduction](#_7hvj8g6jk2a9) 25

[🙊 TypeScript est secondaire](#_mqyk3nhqrgps) 26

[🐲 Ressources sur l’écriture d’annotation de types](#_ma0j1snp4il6) 27

[**🐢 Node.js**](#_owtn4aw1yf1k) **28**

[🌝 Introduction](#_ph3gq8hsu2zo) 28

[🐥 Ressources pour les débutants](#_xp0ewd8zy7ny) 29

[🌐 Découverte de l’écosystème HTTP Node.js](#_lwczmo2gb70t) 30

[🚣 Introduction](#_8pknryfmheg2) 30

[💡 Serveur HTTP natif avec Node.js et un peu de liens sur le protocole.](#_tr1dfrwuya88) 31

[🚁 Framework Small scope](#_rfvphjd3w5tj) 32

[✈️ Framework Big scope](#_amb00me3d4hw) 33

[🌠 Framework Full-stack](#_9nsuoazvs9m) 34

[🎯 Comment faire un choix](#_254hq12nbz81) 35

[📰 Conférences et Articles](#_2mww06on9f01) 36

[🎡 Event-loop](#_7frihhv83k6b) 38

[Introduction](#_y4xty7tdrmbf) 38

[Reactor pattern](#_w6wfdpq9i8l6) 39

[🎥 Un peu d’histoire](#_i7j5s89nbit9) 40

[Libuv](#_bfbu8zmav0wd) 41

[📜 Articles et talks](#_erpvdobxu515) 42

[👽 Native API (création d’addon natif en C, C++ et Rust)](#_m0s1tyl1659p) 44

[Rust](#_tmdqflnj00ym) 45

[📟 CLI](#_ypao1b4pixay) 46

[📦 Packages](#_v2vrm5ypi37f) 47

[Articles et talks](#_n8wgc070p19s) 48

[NEW ! Workshops](#_qcw1jh7u3y94) 48

[Monorepo & workspaces](#_wmoalvj9ztri) 49

[🔍 Debugging & Profiling](#_69zlsvmasdr7) 50

[Modules (CJS & ESM)](#_wkcro1dceunh) 51

[WebSocket](#_2vpz74fm2f12) 53

[📊 Test unitaire et coverage](#_v8lqc6lydezn) 54

[💃 Méthodologies](#_s7dhhvp8e7q2) 55

[🌟 Les différents modules core](#_pssgi2xyt3tk) 56

[Console](#_gvhf36ed1cyr) 56

[Events](#_n1imhcdrizb6) 57

[Path](#_5neyxs80zt5o) 59

[FileSystem (fs)](#_woj9d0dm2lqh) 60

[Timers](#_yieq07fragdk) 61

[URL (API WHATWG)](#_w5hedtugepyp) 62

[OS](#_t7qh53qkueag) 63

[Child process](#_6ld9bmfa0f6h) 63

[Utilities](#_2uox34winnxt) 64

[VM](#_x3n43atq0q1) 65

[Assert](#_ysx7a6viwkug) 65

[Zlib](#_1ik6r6bczdte) 66

[Readline](#_yzp9jkmjmc8i) 67

[perf\_hooks](#_gox2hp4awne5) 68

[async\_hooks](#_k2rmaif75h90) 69

[Crypto](#_eeawbcomq31c) 70

[📦 Archive de packages npm](#_aayksn3rj2tj) 71

[**💾 Database**](#_h0giikdxqhq) **72**

[Introduction](#_3p9nrw8qghoe) 72

[Influence de Node.js sur le chapitre](#_r1xi6ylp9pyx) 72

[L’importance d’apprendre SQL](#_on5osya5jpeo) 73

[PostgreSQL](#_4kx21agt4zut) 74

[NEW ! Vidéo d’introduction](#_8yss58h5rk9v) 74

[MySQL et MariaDB](#_tucex5h2fycv) 75

[MongoDB](#_4fo3dswwqbth) 76

[Redis](#_lijgqbld12dm) 78

[SQLite](#_o970biuls39g) 79

[Autres bases de données](#_4mhchzpco00q) 79

[ORM ou non ?](#_t99li0xcsg88) 80

[**📡 MQTT (broker)**](#_in9l1ywbwwgn) **81**

[Introduction](#_s1ckpb7xgww5) 81

[Redis](#_vhpntsamx38i) 82

[RabbitMQ, ZeroMQ](#_s8wq28xse2dn) 82

[Autres](#_70nko5r9s9t0) 82

[**🔐 Sécurité**](#_cvc3uw7m92eh) **83**

[Introduction](#_7muhhkon9xn3) 83

[Audit](#_wpi5ndc9ilrl) 84

[Garder sous contrôle votre Environnement](#_a8q8if43hgmr) 86

[Dépendances directes et indirectes](#_al6my3xbd66m) 87

[Faille de sécurité courante](#_sfii6xls2qdj) 88

[Analyse statique & AST](#_u0jma6y1wtjr) 88

[Liens et ressources complémentaires à lire](#_oq5umw1mrtjw) 89

[Si la sécurité est un domaine qui vous intéressent](#_9r4mwlis5czz) 89

[**GIT**](#_88y67w56e7ja) **90**

[**🌇 Architecture & Paradigmes**](#_lew1f04xkh) **91**

[SemVer](#_gol9x0w1uhyh) 91

[CalVer](#_s2k3bcws0307) 91

[JSON Schema](#_ko53nc30kntt) 92

[Documentation](#_23oeigdhda3x) 93

[Clean code](#_ymxzsxmi2mpy) 94

[Programmation Orienté Objet](#_g8h2onidal4q) 95

[Ressources complémentaires](#_xsb3nc59rj6l) 96

[Design patterns](#_4yu8fuqeywks) 97

[Compilateur et Interpréteur](#_1gm9ududf0u) 98

[NEW ! DDD (Domain Driven Design)](#_4q4ooofdxlq5) 99

[Autres](#_ql0i3q5kyamn) 99

[**🔬 Monitoring**](#_uxx8j9x8i3ix) **100**

[Introduction](#_n1wwmlounmxq) 100

[API](#_gv56ox26kxk0) 100

[**💻 Système**](#_jkwjfxa2jb2z) **101**

[Introduction](#_am1h4n9zyh0l) 101

[Scripting: Shell, Bash et Powershell](#_iofv8mxv9k8i) 102

[**💖 Remerciements**](#_loxaqrl715c1) **103**

# 

# 👋 Introduction

## **👀 Qui suis-je ?**

Moi c’est Thomas. J’ai 28 ans et je suis indépendant (freelance) depuis le début de ma carrière en 2015. À titre personnel je développe depuis l’âge de dix ans (principalement autour de l’écosystème Web 🌍).

Je développe activement en Node.js depuis 2013 et me considère moi-même comme un Expert (J’ai à mon actif plusieurs centaines de projets en Node.js dont une majeure partie accessible et visible sur mon [github](https://github.com/fraxken)).

J’ai travaillé sur énormément de projets de diverses natures : APIs, CLI, Addon natif, packages npm, compilateur etc.. Je m’oriente de plus en plus sur des projets bas niveau (C++ et Rust) en lien avec le monde du Software. Je suis par ailleurs de plus en plus amoureux de l’étude de la sémantique et de la grammaire des langages de programmation (domaine un peu scientifique / philosophique 😵).

Au delà de tout cela je suis aussi actif sur divers projets, communautés et groupes:

* Fondateur et Mentor d’[ES-Community](https://github.com/ES-Community) 💪🚀 (communauté francophone de JavaScript et Node.js sur Discord [depuis 2015](https://dev.to/fraxken/communaute-javascript-es-community-4nm3)).
* Mainteneur principal d’un gros projet open-source: [SlimIO](https://github.com/SlimIO) 😍. (Une solution de monitoring en Node.js et C++).
* Collaborateur Node.js 🐢 (Membre du [Security Working Group](https://github.com/nodejs/security-wg) 🔓). Je travaille notamment sur des projets open-source en lien avec la sécurité: [Node-secure](https://github.com/ES-Community/nsecure), [JS-X-Ray](https://github.com/fraxken/js-x-ray) (voir aussi l’organisation github NodeSecure [ici](https://github.com/NodeSecure)).
* Lead API & Node.js chez [MyUnisoft](https://www.welcometothejungle.com/fr/companies/myunisoft). Je gère une dizaine de développeurs Node.js. J’ai fait [un article](https://dev.to/fraxken/myunisoft-l-aventure-node-js-12i3) sur mon équipe si cela t’intéresse.

| 🐱 Sentez vous libre de me contacter sur [Linkedin](https://www.linkedin.com/in/thomas-gentilhomme/) si vous avez des questions ou que vous souhaitez revoir vos profils et projets avec moi. |
| --- |

# 

## **👊 Pour qui ?**

Ce document s'adresse principalement à des développeurs (débutants ou non 🐣) qui souhaitent s’orienter vers une carrière **back-end** Node.js. Potentiellement à des personnes souhaitant établir une carrière en transverse ayant la motivation pour subir une montée en compétences aussi bien back que front.

| 👀 En ce qui me concerne j’avais déjà une expérience non négligeable en front-end lors de mes débuts en **2013**. Je ne peux donc pas vous en vouloir si vous souhaitez avoir les deux cordes à votre arc (**mais attention c’est loin d’être simple**). |
| --- |

Le document n’a pas forcément pour objectif de partir dans de la vulgarisation dès lors **que des ressources existantes le font déjà très bien** (sauf si je pense avoir une opportunité et les compétences nécessaires pour le faire mieux). Je préfère le voir comme **un portail vers un ensemble de références** de bien meilleure qualité.

Beaucoup de références ne sont pas en français alors **il est fortement recommandé d’avoir un bon niveau en anglais** (si ce n’est pas le cas il va vous falloir apprendre).

## **💬 Pourquoi ?**

C’est toujours un plaisir pour moi d’aider les autres et de partager mon savoir et mes références ❤️. Je le fais déjà au quotidien au travers des différentes communautés et groupes dans lesquelles je suis actif.

C’est aussi pour moi l’occasion de travailler à la construction d’un document rassemblant une majeure partie des éléments qui ont constitué mon parcours 📚 (et potentiellement **me remettre en question et faire un nouveau travail de recherche**). Cela devrait me permettre à titre personnel de former des juniors et mentee beaucoup plus facilement en les aidant avec mon propre document 💡.

De plus, je suis attristé de voir qu’en France **nous manquons encore cruellement de développeurs qualifiés en Node.js**. Beaucoup de mes clients ont un réel manque de candidats ayant une réelle connaissance ou intérêt envers le monde du back-end (la plupart des développeurs se concentrent bien souvent sur leur parcours front et/ou mobile).

## 

## **😬 L’investissement nécessaire**

Devenir un développeur back-end Node.js **demande énormément de courage 😳 et un investissement souvent bien plus ciblé sur JavaScript** ainsi que diverses notions comme l’architecture, les tests etc. Le front-end est très largement dominant dans l’écosystème JavaScript et quelques fois cela peut être une source de pression pour un débutant (même à mon niveau d’expertise ce n’est pas tout le temps évident… L'année dernière on m'a recommandé de faire du front-end pour avoir plus de missions 😒).

Il y a forcément moins d’opportunités que sur le front mais en contrepartie cela veut aussi dire qu’il y a **beaucoup moins de candidats Junior qui se spécialisent** en back-end JavaScript. Beaucoup d'entreprises se retrouveront dans une position ou les candidats pour une offre sont majoritairement des développeurs ayant une expérience non corrélée au back-end (ou pas suffisamment 😱).

Évidemment il va falloir vous investir très sérieusement dans l’apprentissage et la pratique pour prouver votre motivation avec des actes. De nos jours cela commence souvent par casser la croyance que faire une API CRUD avec “**express**” va être suffisant pour vous octroyer un boulot (c’est de moins en moins le cas ⛔).

Ce document aura justement pour mission de vous offrir un ensemble de références qui vous aideront tout au long de votre parcours (du moins au possible.. ce document **reste imprégné de mes propres forces et faiblesses**).

**Note complémentaire pour les reconversions 🎓**

Sortir d’une formation accélérée ne vous aidera probablement pas (et croire le contraire vous assurera un échec quasiment certain 😟). Les formateurs sont bien souvent peu qualifiés et sous-payés et les taux d’embauches après formation bien plus faibles que ce que l'on vous a vendu (voir inexistant selon le degré de médiocrité du programme). Beaucoup idéalisent le métier et ne sont pas psychologiquement prêt à vivre le quotidien d’un réel développeur 🙉🙈.

Bref, il vous faut absolument fournir de très grands efforts pour vous en sortir et espérer décrocher un premier emploi (si c’est votre objectif). Je vous invite par ailleurs à lire l’excellent article de Rayed sur [Devenir développeur après une reconversion](https://practicalprogramming.fr/devenir-developpeur/).

## **📌 Légendes**

Le document comporte un ensemble de références (liens) vers des vidéos et articles qui sont segmentés par niveau de difficulté et langue ( **FR** français ou  **EN** anglais).

**[Débutant]**

Ce sont bien souvent des ressources de vulgarisation ou d’informations accessibles pour des débutants. Attention sur certains sujets coriaces la notion débutant peut néanmoins sous-entendre que vous avez des acquis.

**[Intermédiaire]**

Ce sont des ressources qui nécessitent une certaine fondation pour pouvoir les aborder. Cela ne veut néanmoins pas dire qu’elles nécessiteront techniquement longtemps pour être abordées.

**[Avancé]**

A la différence d'intermédiaire, les ressources avancées sont beaucoup plus chevelues sur le plan technique et pratique.

**[A savoir]**

Ce sont des liens traitant de sujets qu’il est intéressant de connaître (souvent des détails sur le fonctionnement et les comportements peu compris etc). Ce sont des ressources bien souvent débutant/intermédiaire.

| Les notations **ne sont pas parfaites** et elles sont ici principalement pour essayer de vous guider au maximum. |
| --- |

# 

# JavaScript

## **🌞 Introduction**

Devenir développeur back-end c’est avant tout comprendre que vous **allez passer vos journées à écrire des codes en JavaScript** 😵 (un développeur front aura bien souvent en comparaison beaucoup d’autres tâches comme implémenter des maquettes ce qui lui permettra de respirer).

Je rencontre tous les jours des développeurs qui passent leur temps à galérer car ils ne font jamais les efforts nécessaires **pour combler leur manque de compréhension** des fonctionnalités qui composent JavaScript ou Node.js 😐 (Promise, Symbol, Reflection, Generators, Streams, Event-loop …).

Il est donc nécessaire de vous entraîner et vous former bien plus intensivement que vos collègues qui aspirent à devenir “***majoritairement***” développeur front-end (Même si le front-end lui aussi devient de plus en plus technique avec quelquefois des frameworks qui demandent une maîtrise importante de JavaScript).

Les entretiens **se jouent majoritairement sur votre maîtrise du langage** (plus vous êtes à l’aise et plus vous aurez une chance d’obtenir le boulot). Que ce soit pour un algorithme, un challenge ou même la création d’un projet (une API http par exemple).

Si vous passez dix minutes pour vous souvenir de comment mettre une clé dans un objet, alors **il est fort probable que vous avez grillé votre entretien** 😭. Avoir de l’expérience avec un framework ou une librairie est une bonne chose, mais cela ne doit pas se faire au prix d’un manque de compétences en JavaScript.

| 💡 Les conseils et références qui seront abordés dans **le chapitre JavaScript** ont été “un peu” discutés dans [un talk](https://www.youtube.com/watch?v=k4cABQN46qE) entre Rayed Benbrahim et moi. Le format vidéo vous conviendra peut-être mieux pour une première approche. |
| --- |

## 

## **📕 Dédier un temps quotidien à la lecture**

En tant que développeur, il est important de prendre le temps de lire des ebooks (cours) en ligne sur JavaScript. Même **après plusieurs années de pratique**, je continue de lire très activement tout un tas de livres, études et articles 😇.

Cela va vous permettre d’améliorer et d’acquérir tout un tas de compréhension (comme par exemple la différence entre Primitives et Objets, la coercion etc..). Mais aussi de travailler votre vocabulaire technique (même inconsciemment 💤).

Parmis les références que je vous recommande de lire 👇:

* **EN [Débutant]** [JavaScript for impatient programmers](https://exploringjs.com/impatient-js/index.html) (Axel)
* **EN [Débutant]** [You Don't Know JS](https://github.com/getify/You-Dont-Know-JS) (Kyle)
* **EN [Débutant]** [Eloquent JavaScript](https://eloquentjavascript.net/) (Marijn Haverbeke)
* **EN [Débutant]** [JavaScript ES1 -> ES5](http://speakingjs.com/) (Axel)
* **EN [Intermédiaire]** [Functional-Light JavaScript](https://github.com/getify/Functional-Light-JS/) (Kyle)
* **EN [Avancé]** [Deep JavaScript: Theory and techniques](https://exploringjs.com/deep-js/index.html) (Axel)
* **EN NEW ! [Avancé]** [Multithreaded JavaScript (Concurrency Beyond the Event loop)](https://www.oreilly.com/library/view/multithreaded-javascript/9781098104429/) (Thomas Hunter, Bryan English)

Le Dr. Axel Rauschmayer possède un style qui diffère de Kyle Simpson. Je vous invite à choisir le style d'écriture qui vous convient le mieux. **Attention à Kyle** qui a souvent des idées et un style de code considérés assez “**extrême**”.

### 📖📐 La pratique avec un livre

Il est aussi possible de vous entraîner efficacement à l’aide d’un livre 😉. Prenez tout simplement l’habitude d’expérimenter les fonctionnalités au fur et à mesure que vous les découvrez. Le style d’écriture d’Axel Rauschmayer est plutôt propice à cela (la version en ligne de ces livres et articles est vraiment cool).

N’hésitez pas à rejoindre la  **FR** [communauté JavaScript et Node.js](https://es-community.github.io/Code-of-conduct/) sur Discord pour poser des questions (pas de panique c’est l’objectif d’une communauté et c’est une incroyable opportunité pour vous d’apprendre de développeurs plus expérimentés).

## 

## **💪 Mettez-vous au défi régulièrement**

S'entraîner par le biais de différents exercices et challenges est important car cela va vous permettre d’améliorer plusieurs branches de votre spectre de compétences:

* Algorithmie (**ne pas non plus s’attendre à une progression incroyable** 😟).
* Améliorer votre vitesse d’exécution (fluidité pour écrire le code).
* Pratiquer (et **potentiellement** découvrir) des fonctionnalités du langage. À noter que dans ce cas-là il est important de synchroniser la découverte avec une volonté de vous documenter (articles, talks, ebook).
* Vous permettre de découvrir une part non négligeable de notre quotidien et des difficultés auxquelles nous devons faire face.

| Je vous recommande de vous faire régulièrement des sessions de code plus ou moins intensives. Je vous joins une [vidéo de la chaîne de Tony qui](https://www.youtube.com/watch?v=8eBSC9o8fe8) parle de son expérience et son parcours sur son apprentissage autour des algorithmes. |
| --- |

Les liens suivant vous permettront d’effectuer un ensemble de workshops et challenges:

* **EN [Débutant]** <https://www.freecodecamp.org/>
* **EN FR [Débutant]** <https://nodeschool.io/> (Il y a plus d’une dizaine de workshops vraiment très intéressants à faire, **voir la section suivante**).
* **EN NEW ! [Débutant]** [JavaScript Algorithms Crash Course - Learn Algorithms & "Big O" from the Ground Up!](https://www.youtube.com/watch?v=JgWm6sQwS_I)
* **EN [Intermédiaire]** <https://jskatas.org/>
* **EN [Intermédiaire]** <https://warriorjs.com/>
* **EN [Intermédiaire]** [Challenge “Build Your Own Array in Javascript!”](https://github.com/waterlink/Challenge-Build-Your-Own-Array-In-Js)
* **FR [Avancé]** <https://github.com/fraxken/test-challenge> (Des mini-challenges où l’objectif est l’écriture de test unitaire.. n’hésitez pas à prendre contact avec moi si besoin).

En France il y aussi le [Battle Dev](https://battledev.blogdumoderateur.com/) qui est organisé fréquemment. Les membres de la communauté JS participent depuis quelques éditions.

### 

### 🏫 Nodeschool

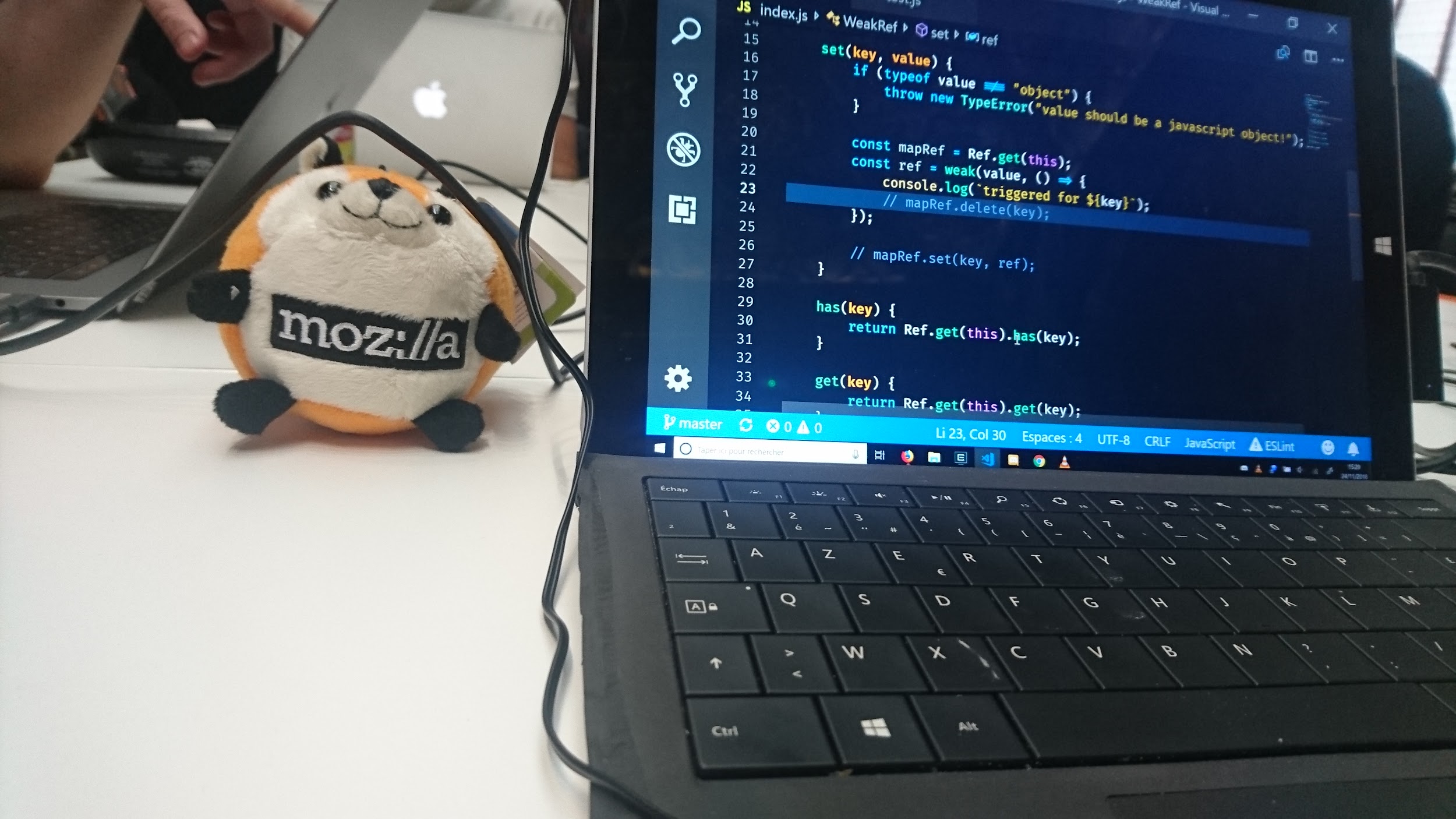
Si vous êtes débutant ou que vous ne connaissez pas encore les workshops de [Nodeschool](https://nodeschool.io/), je vais vous guider dans les différents choix qui s’offrent à vous.

Noter que vous allez devoir devenir à l’aise avec les instructions en ligne de commande 😬 (**c’est une bonne occasion pour vous y mettre si ce n’est pas encore le cas**).

Parmis les workshops que je vous recommande (plus ou moins dans l’ordre) :

1. [JavaScripting](https://www.github.com/sethvincent/javascripting) (relativement simple)
2. [Learnyounode](https://github.com/workshopper/learnyounode)
3. [How to npm](https://github.com/workshopper/how-to-npm) (découverte)
4. [Functional JavaScript](https://github.com/timoxley/functional-javascript-workshop)
5. [Planet proto](https://github.com/sporto/planetproto) (découverte)
6. [Count to 6](https://github.com/domenic/count-to-6) (relativement simple)
7. [Promise it won't hurt](https://github.com/stevekane/promise-it-wont-hurt) (**important**)
8. [Learn generators](https://github.com/isRuslan/learn-generators)
9. [Bytewiser](https://github.com/maxogden/bytewiser)
10. [Stream Adventure](https://www.github.com/substack/stream-adventure) (un peu plus difficile)

Il y a d’autres workshops (libre à vous de regarder s’ils vous intéressent). A noter qu’il existe aussi un évènement [NodeSchool Paris](https://www.meetup.com/fr-FR/NodeSchool-Paris/) avec l’excellent orateur Christophe PORTENEUVE (mais avec le COVID forcément il n’y a plus d’éditions pour le moment).



### 🔎 Les autres sites

Divers sites internet proposant divers challenges et exercices:

* **EN** <https://www.codingame.com/>
* **EN** <https://www.codewars.com/>
* **EN** <https://leetcode.com/>
* **EN** <https://www.hackerrank.com/>(Il y a notamment des catégories pour les Mathématiques et Algo).

Je n'ai pas forcément de recommandation pour ceux-ci (en ce qui me concerne je n’accroche pas forcément avec ces sites).

### 🏆 Competitive programming

Si certains d’entre vous sont des grands motivés et que vous souhaitez vous orienter dans les compétitions de code (ou que les mathématiques et la résolution d’algorithmes c’est votre truc ^^) eh bien je vous recommande les liens suivants :

* **EN** [What is Competitive Programming?](https://www.youtube.com/watch?v=ueNT-w7Oluw)
* **EN** [Competitive Programmer’s Handbook](https://cses.fi/book/book.pdf)
* **EN** [Starting Competitive Programming - Steps and Mistakes](https://www.youtube.com/watch?v=bVKHRtafgPc)
* **EN** [How to start Competitive Programming? For beginners!](https://www.youtube.com/watch?v=xAeiXy8-9Y8)
* **EN** [Fundamentals of algorithms](https://www.geeksforgeeks.org/fundamentals-of-algorithms/)
* **EN** [Codeforces](https://codeforces.com/)

C’est un domaine qui vous demandera de travailler très sérieusement les mathématiques et d’étudier très longuement les diverses structures et algorithmes existants.

| 👀 À noter que j’ai moi-même pour objectif de monter dans le classement du site Codeforces et de participer à plusieurs rounds dans les prochaines années. |
| --- |

## 

## **🌌 Cours en ligne, talks et articles**

### Udemy vs Frontend Masters

Beaucoup de développeurs font l’erreur de choisir [Udemy](https://www.udemy.com/) comme plateforme d’apprentissage **pour JavaScript et Node.js**. La plateforme possède un modèle économique très peu recommandable et déjà plusieurs [scandales](https://www.troyhunt.com/the-piracy-paradox-at-udemy/) ont éclaté au travers du temps 😔.

Il est important de comprendre, que quand votre objectif est de réussir à trouver un emploi dans les prochains mois, il est plutôt problématique de se baser sur du contenu **ayant une qualité très variable, peu ou pas à jour depuis des années et quelquefois à la limite de l’éthique** 😮 (**ce qui ne veut pas non plus dire qu’il n’y a pas du bon contenu j’entends bien**).

Un exemple plutôt flagrant des mauvaises formations est bien souvent des croyances du genre “Une fonction async / une promesse génère un thread en arrière-fond 💩” (**phrase que j’ai moi-même entendue dans un cours d’un formateur réputé “expérimenté”**). Beaucoup ont pour habitude de diminuer les impacts de ses fausses croyances.. Néanmoins elles [ont des impacts réels en production](https://www.youtube.com/watch?v=XV-u_Ow47s0).

Je vous recommande plutôt d’investir sur un compte [Frontend Masters](https://frontendmasters.com/) qui vous permettra d’avoir accès à un ensemble de cours à jour avec[**des experts** qui sont reconnus et qualifiés dans leurs domaines respectifs](https://frontendmasters.com/teachers/) (c’est d’autant plus nécessaire si vous visez le front et le back).

[Les étudiants auront d’ailleurs bien souvent le droit à six mois gratuits.](https://frontendmasters.com/welcome/github-student-developers/)

Quelques-uns de leur cours sur Javascript (je vous invite à regarder vous-même le catalogue des nombreuses autres formations disponibles) :

* **EN** <https://frontendmasters.com/courses/getting-started-javascript-v2/>
* **EN** <https://frontendmasters.com/courses/javascript-hard-parts-v2/>
* **EN** <https://frontendmasters.com/courses/functional-js-fundamentals/>

### 

### 📄 Articles

#### Axel Rauschmayer

Quand il est question d'articles de qualité sur JavaScript, je vous recommande très fortement le [blog d’Axel Rauschmayer](https://2ality.com/).

| ⚠️ Attention néanmoins il est tout de même nécessaire d’avoir bien souvent établi une fondation avant de pouvoir apprécier ces articles. ⚠️ |
| --- |

Voici une liste d’articles de son blog que je vous recommande de lire:

* **EN [Débutant]** [Categorizing values in JavaScript](https://2ality.com/2013/01/categorizing-values.html)
* **EN [Débutant]** [Type coercion in JavaScript](https://2ality.com/2019/10/type-coercion.html)
* **EN [Débutant]** [Remainder operator vs. modulo operator](https://2ality.com/2019/08/remainder-vs-modulo.html)
* **EN [Débutant]** [Synchronous and asynchronous sequential execution of functions](https://2ality.com/2015/11/sequential-execution.html)
* **EN [Débutant]** [How do JavaScript’s global variables really work?](https://2ality.com/2019/07/global-scope.html)
* **EN [Débutant]** [ES2018: Rest/Spread Properties](https://2ality.com/2016/10/rest-spread-properties.html)
* **EN [Débutant]** [Three ways of understanding Promises](https://2ality.com/2016/10/understanding-promises.html)
* **EN [Débutant]** [A different way of understanding this in JavaScript](https://2ality.com/2017/12/alternate-this.html)
* **EN [Débutant]** [undefined vs. null revisited](https://2ality.com/2021/01/undefined-null-revisited.html)
* **EN [Intermédiaire]** [Iterables and iterators in ECMAScript 6](https://2ality.com/2015/02/es6-iteration.html)
* **EN [Intermédiaire]** [JavaScript Promise combinators: .all(), .race(), .allSettled()](https://2ality.com/2019/08/promise-combinators.html)
* **EN [Intermédiaire]** [ES6 Generators in depth](https://2ality.com/2015/03/es6-generators.html)

Bien d’autres articles sont disponibles (ce n’est qu’un faible échantillon de son blog).

#### 

#### Jake Archibald

J’ai tout récemment ré-exploré le blog de Jake et franchement je ne me souvenais plus qu’il avait écrit autant de pépites sur JavaScript 😲.

* **EN [A savoir]** [Don't use functions as callbacks unless they're designed for it](https://jakearchibald.com/2021/function-callback-risks/)
* **EN [A savoir]** [Event listeners and garbage collection](https://jakearchibald.com/2020/events-and-gc/)
* **EN [A savoir]** [await vs return vs return await](https://jakearchibald.com/2017/await-vs-return-vs-return-await/)
* **EN [A savoir]** [Promises: resolve is not the opposite of reject](https://jakearchibald.com/2014/resolve-not-opposite-of-reject/)
* **EN [Intermédiaire]** [JavaScript: What is the meaning of this?](https://web.dev/javascript-this/)
* **EN [Intermédiaire]** [Arrays, symbols, and realms](https://jakearchibald.com/2017/arrays-symbols-realms/)
* **EN [Intermédiaire]** [Async iterators and generators](https://jakearchibald.com/2017/async-iterators-and-generators/)
* **EN [Intermédiaire]** [Iterators gonna iterate](https://jakearchibald.com/2014/iterators-gonna-iterate/)
* **EN [Avancé]** [Streaming template literals](https://jakearchibald.com/2016/streaming-template-literals/)

| 👀 Je vous recommande de lire ces articles plus orientés front-end si le sujet vous intéresse aussi. |
| --- |

#### Surma

Aussi un excellent développeur et vulgarisateur de concepts et fonctionnalités autour du Web (JavaScript et plus). Je vous recommande donc fortement d’aller lire son blog [surma.dev](https://surma.dev/things/index.html) !

* **EN [A savoir]** [My most useful RegExp trick](https://surma.dev/things/regexp-quote/index.html)
* **EN [Intermédiaire]** [Deep-copying in JavaScript](https://surma.dev/things/deep-copy/index.html)
* **EN [Avancé]** [Streams for reactive programming](https://surma.dev/things/streams-for-reactive-programming/index.html)

#### 

#### Lydia Hallie

* **EN [Débutant]** [JavaScript Promises: An introduction](https://web.dev/promises/)
* **EN [Débutant]** [JavaScript Visualized: the JavaScript Engine](https://dev.to/lydiahallie/javascript-visualized-the-javascript-engine-4cdf)
* **EN [Débutant]** [JavaScript Visualized: Promises & Async/Await](https://dev.to/lydiahallie/javascript-visualized-promises-async-await-5gke)
* **EN [Débutant]** [JavaScript Visualized: Prototypal Inheritance](https://dev.to/lydiahallie/javascript-visualized-prototypal-inheritance-47co)
* **EN [Débutant]** [JavaScript Visualized: Generators and Iterators](https://dev.to/lydiahallie/javascript-visualized-generators-and-iterators-e36)
* **EN [Débutant]** [JavaScript Visualized: Hoisting](https://dev.to/lydiahallie/javascript-visualized-hoisting-478h)

#### Autres articles

* **EN [Débutant]** [13 useful JavaScript array tips and tricks you should know](https://dev.to/duomly/13-useful-javascript-array-tips-and-tricks-you-should-know-2jfo)
* **FR [Débutant]** [Les propriétés et descripteurs de propriétés en JavaScript](https://blog.lesieur.name/les-proprietes-et-descripteurs-de-proprietes/)
* **FR [Débutant]** [19 JAVASCRIPT NUGGETS!](https://delicious-insights.com/en/posts/js-nuggets/)
* **FR NEW ! [Débutant]** [Natively Format JavaScript Dates and Times](https://elijahmanor.com/blog/format-js-dates-and-times)
* **EN NEW ! [A savoir]** [Javascript and the next decade of data programming](https://benschmidt.org/post/2020-01-15/2020-01-15-webgpu/)
* **EN NEW ! [A savoir]** [JS classes are not “just syntactic sugar”](https://webreflection.medium.com/js-classes-are-not-just-syntactic-sugar-28690fedf078)
* **EN [A savoir]** [10 Awesome Github Repos Every Web Developer Should Know](https://dev.to/simonholdorf/10-awesome-github-repos-every-web-developer-should-know-27oa)
* **FR [A savoir]** [Expression versus structure de contrôle en JavaScript](https://blog.lesieur.name/expression-versus-structure-de-controle-en-javascript/)
* **EN [A savoir]** [Sometimes undefined is defined](https://medium.com/@bmeurer/sometimes-undefined-is-defined-7701e1c9eff8)
* **EN [A savoir]** [+-0, NaN and Object.is in JavaScript](https://www.stefanjudis.com/today-i-learned/0-nan-and-object-is-in-javascript/)
* **EN [A savoir]** [JSON.stringify takes toJSON methods into consideration](https://www.stefanjudis.com/today-i-learned/json-stringify-takes-tojson-methods-into-consideration/)
* **EN [A savoir]** [What is the difference between parseInt() and Number()?](https://stackoverflow.com/questions/4090518/what-is-the-difference-between-parseint-and-number/4090577#4090577)
* **EN [A savoir]** [3 Ways to Clone Objects in JavaScript](https://dev.to/samanthaming/3-ways-to-clone-objects-in-javascript-2oie)
* **EN [À savoir]** [When "Zoë" !== "Zoë". Or why you need to normalize Unicode strings](https://withblue.ink/2019/03/11/why-you-need-to-normalize-unicode-strings.html)
* **EN [A savoir]** [Property order is predictable in JavaScript objects since ES2015](https://www.stefanjudis.com/today-i-learned/property-order-is-predictable-in-javascript-objects-since-es2015/)
* **EN [A savoir]** [Quick tip: reusable Array search predicates](https://jasonformat.com/reusable-array-search-predicates/)
* **EN [A savoir]** [Javascript and the next decade of data programming](https://benschmidt.org/post/2020-01-15/2020-01-15-webgpu/)

| **Le reste des liens sur la prochaine page** |
| --- |

* **EN [Intermédiaire]** [What I Talk About When I Talk About Sorting: Untangling Array#sort](https://alistapart.com/article/what-i-talk-about-when-i-talk-about-sorting/)
* **EN [Intermédiaire]** [This in JavaScript](https://zellwk.com/blog/this/)
* **EN [Intermédiaire]** [The global `Reflect` object, its use cases and things to watch out for](https://www.stefanjudis.com/today-i-learned/the-global-reflect-object-its-use-cases-and-things-to-watch-out-for/)
* **EN [Intermédiaire]** [Metaprogramming in ES6: Symbols and why they're awesome](https://www.keithcirkel.co.uk/metaprogramming-in-es6-symbols/)
* **EN [Intermédiaire]** [Localized Number Parsing](https://observablehq.com/@mbostock/localized-number-parsing)
* **EN NEW ! [Intermédiaire]** [Sorting JavaScript Arrays By Nested Properties](https://elijahmanor.com/byte/js-array-sort-nest)
* **EN [Avancé]** [JavaScript's Promise Leaks Memory](https://alexn.org/blog/2017/10/11/javascript-promise-leaks-memory.html)
* **EN [Avancé]** [A cartoon intro to ArrayBuffers and SharedArrayBuffers](https://hacks.mozilla.org/2017/06/a-cartoon-intro-to-arraybuffers-and-sharedarraybuffers/)
* **EN [Avancé]** [JavaScript async iterators](https://www.nodejsdesignpatterns.com/blog/javascript-async-iterators/)

| 👀 Si vous êtes à la recherche d'articles débutant sur JavaScript, je vous recommande de jeter un œil aux articles de  **EN** [Samantha Ming](https://dev.to/samanthaming). |
| --- |

### 

### 📺 Talks

J’ai toujours beaucoup apprécié Christophe PORTENEUVE en tant qu’orateur 😍. Je vous recommande donc quelques-uns de ces talks:

* **FR [Débutant]** [ES6+ maintenant!](https://www.youtube.com/watch?v=KJzlllc7Jq8)
* **FR [Débutant]** [Deep dive dans async/await](https://www.youtube.com/watch?v=2P9fbuNEM5w)
* **FR [Débutant]** [Alors, quoi de neuf en ES2022 ?](https://www.youtube.com/watch?v=L2Ul--lVML4)
* **FR [Débutant]** [Writing Modern Async JavaScript (par Christophe Porteneuve)](https://screencasts.delicious-insights.com/writing-modern-async-javascript)
* **FR [Intermédiaire]** [Christophe Porteneuve - Fun and Games With ES Proxies](https://vimeo.com/364362111)

Les autres talks

* **EN [Débutant]** [Apprentissage de la programmation fonctionnelle avec JavaScript](https://www.youtube.com/watch?v=e-5obm1G_FY)
* **EN [Débutant]** [Immutable data structures for functional JS](https://www.youtube.com/watch?v=Wo0qiGPSV-s)
* **EN [Débutant]** [Wes Bos - Async + Await](https://www.youtube.com/watch?v=9YkUCxvaLEk)
* **EN [Débutant]** [!!Con 2019- Tail Call Optimization: The Musical!!](https://www.youtube.com/watch?v=-PX0BV9hGZY&feature=youtu.be)
* **EN [A savoir]** [Navigating Numbers in JavaScript by Meggan Turner](https://www.youtube.com/watch?v=4zveh5TzB6U&list=PL37ZVnwpeshEHcw37PA29vZCJRoIER9r3&index=20)
* **FR [Intermédiaire]** [Les bases des fonctions generators (de moi et tonygo\_)](https://www.youtube.com/watch?v=nrGpYTQW_O0&t=199s)
* **EN [Intermédiaire]** [Fast by default: algorithmic performance optimization in practice](https://www.youtube.com/watch?v=bwA9i6wjfhw&list=PLMW8Xq7bXrG5ifmqyUChS9buBfVnoa3wh&index=3)
* **EN [Avancé]** [Shared Memory & Parallel Programming with JavaScript](https://www.youtube.com/watch?v=vvqfmskTIjE&list=PL37ZVnwpeshFmAPr65sU2O5WMs7_CGjs_&index=26)

| Les épisodes du podcast  **EN** [OpenHive.js](https://www.youtube.com/watch?v=OPLUfbaOYLQ&list=PL0CdgOSSGlBZZu6UQ8r0kAjf-EfyJTC2u) sont aussi assez intéressants à suivre (avec des interlocuteurs d’une très grande expertise à chaque fois). |
| --- |

## 

## **🔧 Engine & VM JavaScript (V8 etc)**

Lorsque **vous serez plus à l’aise avec JavaScript** il vous sera nécessaire d’étudier un minimum le fonctionnement des moteurs modernes comme V8, JS Core, SpiderMonkey etc (ce sont eux qui ont la responsabilité d'interpréter et exécuter votre code JavaScript que ce soit dans Node.js ou même dans le navigateur).

Ce n'est clairement pas un investissement à la portée d’un débutant mais plutôt d’un développeur de niveau intermédiaire ou expérimenté 📚. Pour pouvoir mieux comprendre comment votre code sera géré et optimisé il vous sera donc nécessaire d’apprendre les rouages de la machine 😈.

Parmis les articles que je vous recommande très fortement de lire:

* **EN [Débutant]** [How JavaScript works: an overview of the engine, the runtime, and the call stack](https://blog.sessionstack.com/how-does-javascript-actually-work-part-1-b0bacc073cf)
* **EN [Débutant]** <https://mathiasbynens.be/notes/shapes-ics>
* **EN [Débutant]** <https://mathiasbynens.be/notes/prototypes>
* **EN [Intermédiaire]** [Whats up with monomorphism](https://mrale.ph/blog/2015/01/11/whats-up-with-monomorphism.html)

Il existe néanmoins des dizaines d’articles tous aussi passionnants que j’ai pris la peine de rassembler ici 👀: <https://github.com/fraxken/VM-Resources>

Quelques talks en plus pour votre plus grand bonheur:

* **FR** [Comprendre le fonctionnement d'un moteur Javascript (Adrien MARET)](https://www.youtube.com/watch?v=E1hohefv8QI)
* **EN** [JavaScript engines - how do they even?](https://www.youtube.com/watch?v=p-iiEDtpy6I)
* **EN** [The Past, Present and Future of JavaScript Engines](https://www.youtube.com/watch?v=iLWDUJMCoWs&list=PLfMzBWSH11xZPfWcC0DqFqKo_reMP58mw&index=45)
* **EN** [JavaScript Engines: The Good Parts](https://www.youtube.com/watch?v=5nmpokoRaZI&list=PL37ZVnwpeshG2YXJkun_lyNTtM-Qb3MKa&index=11)
* **EN** [A sneak peek into super optimized code in JS frameworks](https://www.youtube.com/watch?v=_VHNTC67NR8&list=PL37ZVnwpeshHwJPVBqEnZild7QHWhdufu&index=2)
* **EN** [Embedding V8 in the real world by Stanimira Vlaeva](https://www.youtube.com/watch?v=wz7Znu6tqFw&list=PL37ZVnwpeshHwJPVBqEnZild7QHWhdufu&index=16)
* **EN** [Parsing JavaScript - better lazy than eager?](https://www.youtube.com/watch?v=Fg7niTmNNLg&list=PL37ZVnwpeshFmAPr65sU2O5WMs7_CGjs_&index=10)
* **EN** [Essentials of Interpretation](https://www.youtube.com/watch?v=8OYqvwQlJVI&list=PLGNbPb3dQJ_4WT_m3aI3T2LRf2R_FKM2k)

### 

### 😡 La dangereuse mode des benchmarks

Je pense qu’il est important d’aborder le sujet des benchmarks pendant que nous sommes dans la section engine JS. Les développeurs adorent les utiliser comme argument pour justifier divers choix ou idéologies 😰…

Le problème c’est que la plupart du temps ces benchmarks sont complètement ratés et/ou ne sont pas représentatifs d’un workload de production 😂. Même s’ils sont concrets il va vous falloir de l’expérience pour en déduire des conclusions (et encore rien ne dit que le souci vous concerne par ailleurs).

Voici quelques articles pour vous éveiller au sujet:

* **EN** [Dangers of cross language benchmark games](https://mrale.ph/blog/2011/05/12/dangers-of-cross-language-benchmark-games.html)
* **EN** [The trap of the performance sweet spot](https://mrale.ph/blog/2011/11/05/the-trap-of-the-performance-sweet-spot.html)
* **EN** [Performance tuning as the art of weather forecast](https://mrale.ph/blog/2013/04/29/performance-tuning-as-weather-forecast.html)
* **EN** [The Black Cat of Microbenchmarks](https://mrale.ph/blog/2014/02/23/the-black-cat-of-microbenchmarks.html)
* **EN** [JavaScript MicroBenchmarks (from Benedikt Meurer)](https://github.com/bmeurer/js-micro-benchmarks)

Même si cela peut paraître difficile à entendre, je pense qu’une personne n’ayant pas de solide connaissance sur le fonctionnement des moteurs JavaScript n’a pas de légitimité à formuler des conclusions à partir des résultats d’un benchmark 💥.

*“The hardest thing of all is to find which operation is more expensive inside the darkness of VM, especially when no operation is performed.” (Vyacheslav Egorov)*

Et même les personnes ayant beaucoup d’expérience (dont notamment les contributeurs des moteurs eux-mêmes) ont toujours le doute et préfèrent prendre des pincettes pour chacune de leurs conclusions 😯. C’est vous dire la difficulté de la tâche… savoir si vous allez faire **for-in**, **for-of** ou **.forEach** n’a pas vraiment grand intérêt ici.

*“ 👉 L'optimisation prématurée est la racine de tous les maux (ou, du moins, la plupart d'entre eux) en programmation.” (Donald Knuth)*

# 

# ⚡ ECMAscript

## **JavaScript ou ECMAScript ?**

JavaScript est inventé en 1995 par Brendan Eich alors qu’il est employé de la société Netscape Communications. Microsoft lui emboîte le pas en incluant JavaScript dans son logiciel Internet Explorer, alors en version 3. Pour des raisons de droits de marque, il y est dénommé JScript 😐.

La spécification est ensuite validée par l’organisme Ecma International en juin 1997 sous le nom d’ECMAScript, standard ECMA-262.

Le terme JavaScript est resté dans le vocabulaire courant, mais, en fait, il s’agit bien d’ECMAScript 💃.

| Texte repris du chapitre  **FR**  “[Qu’est que JavaScript ?](https://oncletom.io/node.js/chapter-03/#definition)” du livre de Thomas Parisot. |
| --- |

## **ECMAScript 6 (ES2015)**

En 2015 est sortie la version 6 d’ECMAScript qui a ajouté bon nombre de fonctionnalités dans le langage. Aujourd’hui j’aurais plutôt tendance à parler de JavaScript moderne pour désigner l’apprentissage et la maîtrise des nouvelles fonctionnalités qui sont implémentés chaque année (et pas seulement ES6).

A partir de 2015 il est courant d’utiliser l’année pour désigner la version de JavaScript… Au lieu d’ECMAScript 8 on utilisera plutôt ES2017.

## **JavaScript: 20 ans 🎉**

Il n’y a pas si longtemps de ça, Brendan Eich et [Allen Wirfs-brock](http://www.wirfs-brock.com/allen/about) ont coécrit un excellent ouvrage qui parle des 20 premières années de JavaScript.

**EN** [JavaScript: the first 20 years](https://dl.acm.org/doi/10.1145/3386327)

## 

## **TC39**

À un certain degré, il sera intéressant pour vous de vous pencher plus précisément sur le travail effectué par [TC39](https://tc39.es/) qui est le groupe responsable de la spécification du langage 💖. Pour mieux comprendre ce qu’est TC39 ainsi que le rôle des différents membres composant ce groupe je vous invite à écouter le [talk suivant](https://www.youtube.com/watch?v=slA06pbTRi4&list=PL37ZVnwpeshHwJPVBqEnZild7QHWhdufu&index=13) (ou bien celui de l’année [précédente](https://www.youtube.com/watch?v=Hj5q8uyqGYc&list=PL37ZVnwpeshG2YXJkun_lyNTtM-Qb3MKa&index=8)).



Il est par exemple possible de suivre les prochaines fonctionnalités qui arriveront dans JavaScript grâce au repo git suivant: <https://github.com/tc39/proposals>. L’intégralité des propositions suit un processus bien défini qui est décrit [ici](https://tc39.es/process-document/).

En 2018 nous avions organisé  **FR** [un Weektalk](https://www.youtube.com/watch?v=_onjCxViPu8) sur la communauté Discord francophone de JavaScript pour parler de quelques proposals. Si vous souhaitez apprendre à lire et comprendre la [spécification ECMAScript](https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf), je vous invite à consulter [les articles suivants](https://v8.dev/blog/tags/understanding-ecmascript) de l’équipe V8.

Autres liens:

* **EN** [Archives d’ECMAScript](https://www.ecma-international.org/ecmascript-development-archive/)
* **EN** [ECMAScript forum](https://es.discourse.group/)
* **EN** [ES Discuss (Forum d'échange sur ECMAScript)](https://esdiscuss.org/)

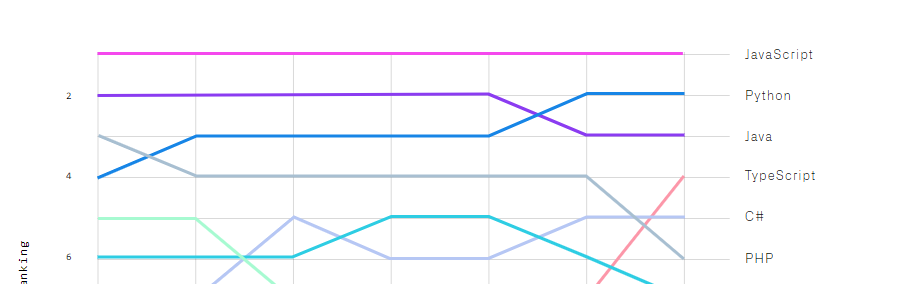
# TypeScript

## **Introduction**

[TypeScript](https://www.typescriptlang.org/) est un superset de JavaScript (donc du JavaScript mais avec des annotations de types). Le compilateur propose aussi des fonctionnalités qui permettent une compilation source à source plus stricte.

Dans l’écosystème Javascript, TypeScript devient de plus en plus populaire et est utilisé dans bon nombre de situations et ce même en JavaScript classique (par le biais de [JSDoc](https://www.typescriptlang.org/docs/handbook/jsdoc-supported-types.html) ou de fichier de définition). Si vous utilisez VSCode pour écrire du JavaScript alors sans vous rendre compte TypeScript vous rend aussi service :)

[JavaScript #1 et TypeScript #4](https://octoverse.github.com/)



Il y a un bon nombre de débats sur Internet pour savoir quels sont les avantages et inconvénients de TypeScript. Comme toute chose dans la vie rien n’est tout blanc ou tout noir et il faut aussi savoir conserver les pieds sur terre. En ce qui me concerne, j'aime bien jongler entre des projets en JavaScript et des projets en TypeScript (tout dépend du contexte et du projet).

Néanmoins, je pense qu’il est important de dire qu’ici nous parlons de TypeScript strict (j’ai fait de nombreuses missions sur des codes avec des any partout…).

Il est vrai que, lorsqu’un code est proprement annoté avec des interfaces et types, il est souvent plus simple de maintenir le code (et certainement plus simple de lire et comprendre pour des développeurs avec moins d’expérience). Enfin cela reste évidemment un sentiment qui n’a certainement rien de concret (il faut aussi avoir confiance en ses collègues.. rien ne dit qu’ils ont annoté le code correctement ^^).

## **🙊 TypeScript est secondaire**

De plus en plus de développeurs débutants sont poussés à faire du TypeScript **sans réellement comprendre de quoi il s’agit** (ce qui les embrouille encore plus dans leur parcours).

Aujourd’hui il n’est pas rare de croiser des développeurs qui pensent que TypeScript est complètement décorrélé de JavaScript.. Ce qui est complètement faux (c’est même tout le contraire).

Il est important d’apprendre les choses dans le bon ordre… (ce qui ne veut pas dire que vous ne pouvez pas vous entraîner en parallèle à [mieux comprendre les différentes annotations de types disponibles en TypeScript](https://2ality.com/2018/04/type-notation-typescript.html) et vous [challenger sur l'écriture de celle-ci.](https://github.com/type-challenges/type-challenges)).

Prenez le temps de vous construire une fondation en JavaScript pour mieux comprendre les avantages et désavantages quand vous aurez à choisir TypeScript pour vos projets.

| ✨ Je n’ai rien contre TypeScript et je l’utilise quotidiennement à travers divers projets d’entreprises et perso… même dans le cadre de projets purement écrits en JavaScript : TypeScript reste omniprésent (VS Code, Définition de types, JSDoc …).  Mon objectif est de mettre en garde sur les raccourcis qui sont souvent faits et qui vont potentiellement nuire à la veille des plus débutants. ✨ |
| --- |

### 

## 🐲 **Ressources sur l’écriture d’annotation de types**

Je me rends compte aujourd’hui qu’il m’est difficile de vous offrir un maximum de ressources sur le sujet, car j’ai vécu et travaillé avec les différentes versions de TypeScript au fur et à mesure qu’elles ont été publiées (depuis la version 1.1).

Du coup j’ai toujours eu un apprentissage plus ou moins linéaire 😊.

Je pense néanmoins qu’il est impossible de devenir un Expert en TypeScript si vous n’avez pas les fondations en JavaScript (forcément oui… par exemple si vous ne comprenez pas ce qu’est une Primitive ou un Objet comment espérez-vous pouvoir annoter correctement ?).

Voici néanmoins quelques liens :

* **EN [Débutant]** [How does TypeScript work? The bird’s eye view](https://2ality.com/2020/04/typescript-workflows.html)
* **EN [Débutant]** [The TypeScript handbook](https://www.typescriptlang.org/docs/handbook/intro.html) && [TypeScript new handbook](https://github.com/microsoft/TypeScript-New-Handbook)
* **EN [Débutant]** [Tackling TypeScript: Upgrading from JavaScript](https://exploringjs.com/tackling-ts/index.html)
* **EN [Débutant]** [TypeScript - A Love Tale with JavaScript](https://www.youtube.com/watch?v=9YOHg3rt3W8)
* **EN [Intermédiaire]** [TypeScript Advanced Types](https://www.typescriptlang.org/docs/handbook/advanced-types.html)
* **EN [Intermédiaire]** [Learn Advanced TypeScript Types](https://medium.com/free-code-camp/typescript-curry-ramda-types-f747e99744ab)
* **EN [Intermédiaire]** [Advanced TypeScript Types cheat sheet (with examples)](https://dev.to/ibrahima92/advanced-typescript-types-cheat-sheet-with-examples-5414)
* **EN [A savoir]** [JSDoc typings: all the benefits of TypeScript, with none of the drawbacks](https://gils-blog.tayar.org/posts/jsdoc-typings-all-the-benefits-none-of-the-drawbacks/)
* **EN [A savoir]** [Duck Typing in JavaScript and TypeScript](https://blog.bitsrc.io/duck-typing-in-javascript-and-typescript-7cc834fadd64)

Quelques articles du Dr. Axel Rauschmayer:

* **EN [Débutant]** [Typing objects in TypeScript](https://2ality.com/2020/01/typing-objects-typescript.html)
* **EN [Débutant]** [Typing Arrays in TypeScript](https://2ality.com/2020/02/typing-arrays-typescript.html)
* **EN [Débutant]** [Typing functions in TypeScript](https://2ality.com/2020/04/typing-functions-typescript.html)
* **EN [Débutant]** [Adding special values to types in TypeScript](https://2ality.com/2020/01/special-values-typescript.html)
* **EN [Intermédiaire]** [Computing with types in TypeScript](https://2ality.com/2020/06/computing-with-types.html)
* **EN [Intermédiaire]** [TypeScript: narrowing types via type guards and assertion functions](https://2ality.com/2020/06/type-guards-assertion-functions-typescript.html)

# 

# 🐢 Node.js

## **🌝 Introduction**

En première étape, il est vraiment important de bien comprendre ce qu’est Node.js. Je vous recommande vivement de visionner l’excellente playlist  **FR** “[Formation Node.js](https://www.youtube.com/watch?v=HZVATyDtsyc&list=PL4NbGBfr4aJk5ATFWA-8UkvL8LbHqpS-v)” de [Louistiti](https://twitter.com/louistiti_fr) qui saura vulgariser le sujet bien mieux que moi (malheureusement pas encore terminé).

Node.js est un [runtime](https://fr.wikipedia.org/wiki/Environnement_d%27ex%C3%A9cution#:~:text=Un%20environnement%20d'ex%C3%A9cution%20ou,un%20langage%20de%20programmation%20donn%C3%A9.) JavaScript utilisant le moteur [V8](https://v8.dev/) de Google Chrome. **Node n’est pas un langage de programmation ni même un framework**.

Node.js est une composition de code et librairies dont Libuv, OpenSSL, Zlib ([liste complète ici](https://nodejs.org/en/docs/meta/topics/dependencies/)). Tout cela va nous permettre, en tant que développeurs, de pouvoir interagir par le biais de différents protocoles (http, tcp/ip, udp). avec le système de fichiers de votre ordinateur, le réseau etc...

Node.js est comme une boîte d’outils qui peut vous permettre de réaliser une vaste quantité de projets de nature diverses et variées (et pas seulement des API http comme beaucoup le pensent… par exemple [NodeSecure](https://github.com/ES-Community/nsecure) ou [SlimIO](https://github.com/SlimIO)). C’est pour cela que l'on parlera d’environnement d’exécution.

Bonus:  **EN** [Things to Love About Working with Node.js (Matteo Collina)](https://www.youtube.com/watch?v=xdLfCCq2R4I)

## 

## **🐥 Ressources pour les débutants**

Depuis maintenant un ou deux ans, il y a le site [nodejs.dev](https://nodejs.dev/learn) qui offre un excellent moyen de démarrer l’aventure avec Node.js.

En français on m’a récemment recommandé  **FR**  [le livre (Node.js apprendre par la pratique)](https://oncletom.io/node.js/#Chapitrage-et-contenu-du-livre) de Thomas Parisot que je trouve excellent (un Thomas c’est normal 😆). Il couvre énormément de sujets et **c’est plutôt cool pour les débutants**.

Parmis les autres ressources que je vous recommande :

* **EN [Intermédiaire]** [Les guides officiels de Node.js](https://nodejs.org/en/docs/guides/).
* **EN [Débutant]** L’article de Liz Parody: [ABC’s of JavaScript and Node.js](https://nodesource.com/blog/ABC-of-JavaScript-and-Nodejs) pour acquérir et comprendre les différents mots clés employés dans l’écosystème Node.
* **EN [Débutant]** [Art of Node](https://github.com/maxogden/art-of-node) par Maxogden (plutôt pas mal pour ceux ayant déjà une expérience en programmation).
* **EN [Intermédiaire]** Si vous êtes à la recherche d’une lecture plus pointue je vous recommande: [Node.js design patterns](https://www.nodejsdesignpatterns.com/).
* **EN [À savoir]** Les [meilleures pratiques](https://github.com/goldbergyoni/nodebestpractices) pour les projets Node.js.
* **EN [Débutant]** [Free Introduction to Node.js (OpenJS)](https://openjsf.org/blog/2021/02/18/free-node-js-online-training-now-available/)
* **EN [Débutant]** [Apprendre Node.js avec Microsoft](https://www.youtube.com/playlist?list=PLbl2SbVIi-Wo0EkNoLEnx4BE_xm4SsSRj)

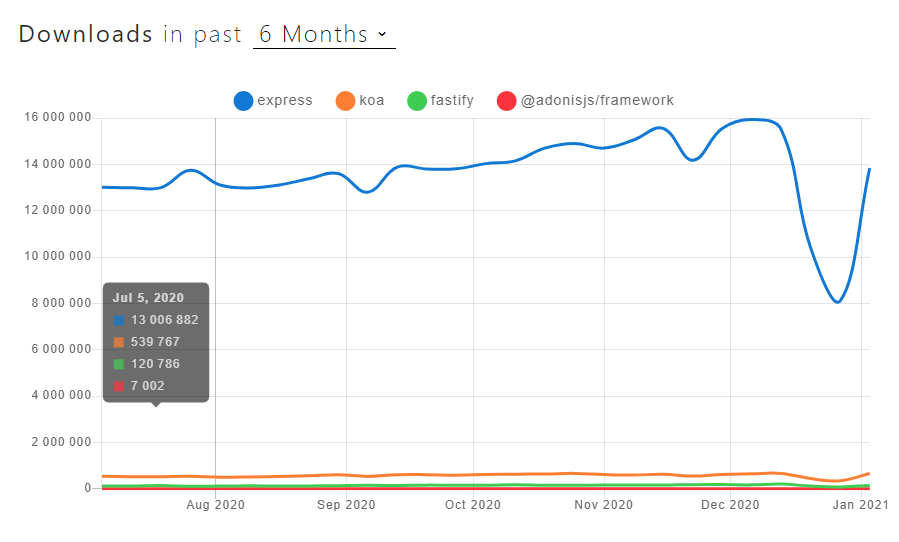
Nous avions abordé [NodeSchool](#_70tu7zxaybuj) dans la section JavaScript (c’est un excellent moyen en plus des autres liens pour débuter et s'entraîner sur Node.js).

## 

## **🌐 Découverte de l’écosystème HTTP Node.js**

### 🚣 Introduction

Il est assez fréquent que les nouveaux développeurs commencent avec [express](https://expressjs.com/fr/) car c’est le framework qui nous est directement vendu comme “le plus populaire” (et évidemment le plus présent dans les différents tutoriels accessibles en surface).



Néanmoins [ce n’est pas forcément le meilleur choix](https://dev.to/romainlanz/why-you-should-drop-expressjs-in-2021-711) car l’équipe derrière **express** maintient difficilement le framework! (**une V5 attendu depuis six ans 🙊**) De plus, beaucoup des patterns qu’ils recommandent sont complètement démodés (toujours pas de support async/await …). Vous allez écrire du code qui n'inclut rien des pratiques modernes.

**Comment choisir le bon framework alors ?** Eh bien il nous faut déjà essayer de segmenter ces frameworks par famille pour mieux comprendre et débattre des différents trade-offs que chaque solution implique.

Ces familles sont les suivantes;

* Small scope
* Big scope
* Full Stack scope

| *La segmentation est inspirée d’une explication de* [*Romain Lanz*](https://twitter.com/romainlanz) *dans un talk* [*Adonis.js*](https://anchor.fm/the-adonisjs-podcast/episodes/Overture---Why-Should-we-use-AdonisJS-emlgjg)*! (qu’il a repris lui-même* [*d’Evan You*](https://www.youtube.com/watch?v=ANtSWq-zI0s)*).* |
| --- |

### 💡 Serveur HTTP natif avec Node.js et un peu de liens sur le protocole.

Avant de rentrer dans le vif du sujet et de parler des différentes familles de “frameworks” dans l’écosystème Node.js, il me semble important de faire un petit chapitre pour vous parler du [module core http](https://nodejs.org/api/http.html).

Il peut être intéressant d’étudier et travailler [le fonctionnement d’un serveur HTTP Natif](https://www.digitalocean.com/community/tutorials/how-to-create-a-web-server-in-node-js-with-the-http-module). Cela ne vous sera pas forcément directement nécessaire pour du code de production, mais va vous permettre de comprendre, un peu mieux, comment interagir en HTTP sans avoir l’abstraction d’un framework qui s’occupera de gérer la difficulté et les détails relous pour vous (***Si vous n’avez pas déjà une expérience conséquente sur le sujet évidemment***).

C’est aussi intéressant pour vous permettre d’avoir une meilleure compréhension du [protocole HTTP](https://tools.ietf.org/html/rfc7231). Je vous recommande aussi de lire le book **NEW !** [High Performance Browser Networking](https://hpbn.co/).

Mon collègue youtuber [Tonygo\_](https://www.youtube.com/channel/UC0yiy-XPDRVAgLaAiA8kvrQ) a par ailleurs produit quelques vidéos sympas dessus;

* **FR [Débutant]** [TOUT SUR LES MÉTHODES HTTP EN 12 MINUTES !](https://www.youtube.com/watch?v=B-3TGySiSLo)
* **FR [Débutant]** [CONNAÎTRE TOUS LES STATUS CODES HTTP PAR COEUR ?](https://www.youtube.com/watch?v=twKgr_ERZJI)

Autres talks Node.js autour de HTTP 1, 2 et 3 (QUIC).

* **EN** [Implementing HTTP/2 for Node.js Core](https://www.youtube.com/watch?v=7uNGKCao8gA)
* **EN** [A QUIC Update](https://www.youtube.com/watch?v=Fu9i2knjq8g)
* **FR [Débutant]** [Workshop API REST, RPC & ResourceFUL (talk que j'ai fait pour une mission)](https://www.dropbox.com/s/gdny6hc2calq0k5/Workshop%20API.mp4?dl=0)

Documentation sur l’écriture d’un serveur HTTP Node.js sans frameworks:

* **EN** [Node.js server without a framework (MDN)](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Node_server_without_framework)

Je vous recommande aussi de garder à l'œil le projet [undici](https://github.com/nodejs/undici) qui devrait être le prochain client HTTP Node.js (plutôt que d’utiliser got ou axios).

### 

### 🚁 Framework Small scope

Ces frameworks ou “micro-frameworks/libs” pour certains 😏 sont bien souvent de petite taille et imposent très peu d’opinions au développeur (le framework ne vous contraint pratiquement pas à suivre des pratiques, des patterns, un outil X plutôt que Y etc). Ils sont bien souvent de simples routeurs http avec une modularité fournie par le biais d’un simple système de middleware.

Il est souvent de la responsabilité du développeur pour chaque API d’assembler/coller un ensemble de packages npm et middlewares entre eux pour aboutir au résultat attendu.

C’est une façon de faire qui donne “souvent” beaucoup de flexibilité et une certaine sensation de liberté (ce qui peut être perçu positivement ou négativement selon les besoins et les mentalités de chacun). C’est certainement aussi une façon de faire qui correspond à beaucoup de développeurs qui n’aiment pas particulièrement suivre des modèles prédéfinis.

Liste non exhaustive de ces frameworks

* [express](http://expressjs.com/)
* [koa.js 2](https://koajs.com/) (petit frère d’express, encore vivant mais je ne recommande pas l’utilisation de ce framework pour 2021+).
* [hapi](https://hapi.dev/) (j’aurais recommandé il y a encore peu... mais je pense que les dernières annonces ont sacrément impacté la maintenance des différents packages).
* [polka](https://github.com/lukeed/polka#readme)
* [fastify.js](https://www.fastify.io/)
* [micro](https://github.com/vercel/micro)

À ce jour, le framework le mieux maintenu est **Fastify** (qui est supporté par la fondation [OpenJS](https://openjsf.org/)). J’utilise aussi beaucoup la version @next de polka pour construire des micro API très light (notamment pour de l’http embarqué et des CLI).

Cela inclut aussi les frameworks spécialisés dans la création de micro-service comme [Moleculer](https://moleculer.services/). Néanmoins il est moins fréquent de les rencontrer en entreprise (moi-même je n'ai pas réellement d’expérience avec un de ces frameworks).

### 

### ✈️ Framework Big scope

Ces frameworks vous imposent bien souvent dès le début un ensemble d’opinions et vous fournissent un ensemble de packages et modules pré-intégrés pour gérer divers besoins (Template engine, Form Validator, ORM, Auth, Queue, Sécurité etc).

L’objectif est de vous apporter **très rapidement une plus-value business et une meilleure expérience développeur** sans que vous ayez à reconstruire la roue pour chaque fonctionnalité qui ne serait pas liée au besoin métier de votre application.

La courbe d’apprentissage de ces frameworks peut être légèrement plus longue car ils demandent souvent d’acquérir tout un ensemble de notions supplémentaires qui sont elles même souvent en lien avec les opinions du framework (Structure d’un projet, MVC etc...) ainsi que prendre en main les éléments et code souvent propre à chaque framework (**Ce qui n’est pas forcément un point négatif sachant que bien souvent les développeurs s’amusent à recréer cela eux mêmes sur express**).

Néanmoins vous y gagnerez certainement en productivité sur le long terme (surtout si vous êtes amenés à souvent construire de nouveaux projets avec des besoins métier assez importants).

Comme pour le Small-scope ce sont des frameworks purement back-end. La liste des frameworks “sérieux” est néanmoins assez courte car il est vraiment très difficile de construire et maintenir de tels projets sur le long terme:

* [Adonis.js](https://adonisjs.com/) (Je vous recommande aussi de lire la page [Why Adonis.js ?](https://preview.adonisjs.com/why-adonisjs))
* [Nest.js](https://nestjs.com/)

**⚠️ Attention** car de nos jours beaucoup de vieux frameworks qui ne sont plus réellement maintenus sont toujours activement recommandés :

* [Sails.js](https://sailsjs.com/) (même chose pour le fork Trails.js). Je vous recommande de lire [Don’t Use Sails (or Waterline)](https://kevin.burke.dev/kevin/dont-use-sails-or-waterline/).
* [Loopback 3](https://loopback.io/) (je ne recommande pas non plus la version 4).

| J’ai eu l’occasion de travailler professionnellement avec ces deux frameworks et je vous recommande très fortement de les fuir pour tout nouveau projet.  Il n’est pas rare qu’aujourd’hui des développeurs maintiennent encore des projets sur ces frameworks 😨 |
| --- |

### 🌠 Framework Full-stack

Je ne suis pas un grand spécialiste sur les frameworks fullstack (je n’ai d’ailleurs encore aucune expérience avec ces frameworks car je fais que très peu de front-end ces dernières années).

Néanmoins je peux vous dire que les frameworks fullstack sont “**souvent**” des frameworks qui vont essayer de réunir front-end et back-end pour construire le tout au sein d’une abstraction unifiée. Leur objectif est de simplifier le cycle de développement (et l’expérience du développeur A.K.A **DX**) par le biais d'abstractions qui vont cacher divers détails techniques (enfin c’est l’idée.. après dans la réalité cela peut être source de débats).

Il existe néanmoins une assez grande diversité de choix et de façon de faire les choses, alors il est assez difficile de mettre des mots précis sur ces choses-là. Le plus important pour moi est de comprendre qu’il y a toujours la composante front et back qui vont s'entremêler.

Liste des frameworks (uniquement ceux que je connais):

* [Next.js](https://nextjs.org/) (Utilise React.js pour le front, ne pas confondre avec Nest.js)
* [Nuxt.js](https://fr.nuxtjs.org/) (Utilise Vue.js pour le front, voir [le weektalk #8](https://www.youtube.com/watch?v=wte-XtcaTCo) par Romain Lanz).
* [Sapper](https://sapper.svelte.dev/) (Utilise Svelte.js pour le front).
* [Gatsby](https://www.gatsbyjs.com/)

| ⛔ Je vous recommande fortement d’éviter le framework [Meteor](https://www.meteor.com/) (même délire que le chapitre précédent). |
| --- |

### 

### 🎯 Comment faire un choix

Si vous êtes un développeur plutôt front-end avec un petit intérêt pour le back-end, alors il est fort probable que les frameworks fullstack soient plus intéressants pour vous (Next, Nuxt ou Sapper).

Si vous souhaitez plutôt vous spécialiser en back-end alors je vous recommande de vous intéresser aux deux familles (small & big scope). En ce qui me concerne je vous recommande [Fastify](https://www.fastify.io/) et [Adonis](https://adonisjs.com/) / [Nest.js](https://nestjs.com/).

* **EN** [Intro to Fastify - Matteo Collina](https://www.youtube.com/watch?v=FQu8FnTzOR0&feature=emb_logo) (podcast [bonus ici](https://webrush.io/episodes/episode-96-expressway-to-fastify-with-matteo-collina)).
* **EN** [Adonis 5 Tour - Romain Lanz](https://www.youtube.com/watch?v=TysfaNcFX_Y&feature=emb_logo)
* **FR [débutant]** [LMHB #3: MONTER UNE APP WEB RAPIDEMENT AVEC ADONIS.JS FEAT @ROMAINLANZ](https://www.youtube.com/watch?v=vcrjgZezaI0&feature=youtu.be)

| 👀 Un récent article de Snyk parle d’ailleurs  **EN**  [du succès de Fastify](https://snyk.io/advisor/hub/what-makes-fastify-a-successful-project/). |
| --- |

Si vous êtes plutôt un puriste ou un développeur bas niveau et que vous aimez bien tout maîtriser de A à Z, alors il est fort probable que la famille small-scope soit pour vous. Il peut être intéressant pour vous dès lors d’explorer les différentes solutions (même si elles ne sont pas forcément mainstream).

| 📌 Notons tout de même que Adonis possède aussi une version light qui n’embarque que le strict minimum ce qui peut être aussi une piste viable à visiter. |
| --- |

Il existe un framework du nom de [Kuzzle](https://kuzzle.io/) que je ne savais pas trop comment classer (il représente certainement à lui seul une famille ^^). Je souhaitais tout de même en parler très rapidement sachant que le framework à un support non négligeable en France.

## 

## **📰 Conférences et Articles**

Il existe beaucoup de conférences sur Node.js mais nous allons principalement nous concentrer sur les interventions des conférences Node.js + interactive, [dotJS](https://www.youtube.com/user/dotconferences/playlists) et [JSConf](https://www.youtube.com/channel/UCzoVCacndDCfGDf41P-z0iA) (ou d’autres conférences avec des intervenants de qualité).

La plupart de ces talks sont disponibles sur la chaîne youtube [Node.js](https://www.youtube.com/channel/UCQPYJluYC_sn_Qz_XE-YbTQ) ou [Nearform](https://www.youtube.com/c/nearForm/videos) (il y a plusieurs centaines de talks donc il ne sont pas tous référencés ci-dessous).

* **EN [Débutant]** [Node.js 12: A Decade of Node.js by Beth Griggs](https://www.youtube.com/watch?v=HP4N0u_dEgI&list=PLfMzBWSH11xZPfWcC0DqFqKo_reMP58mw&index=53)
* **EN [Débutant]** ["A Tale of Two Asyncs: Open Source Language Design in Rust and Node.js" by Ashley Williams](https://www.youtube.com/watch?v=aGJTXdXQN2o)
* **EN [Débutant]** [The Module Ecosystem - The Stress of Success](https://www.youtube.com/watch?v=swHbRSzDZxY&feature=emb_logo)
* **EN [Débutant]** [Modern JavaScript Infrastructures - Maël Nison](https://www.youtube.com/watch?v=F5OjiN9OcdQ&list=PL0CdgOSSGlBalMPxFFycq7OIqQF8cJS28&index=20)
* **EN [Débutant]** [Bringing JavaScript Back to Life](https://www.youtube.com/watch?v=XQIo9knnb2s)
* **EN [Débutant]** [Web APIs in Node.js Core: Past, Present, and Future](https://www.youtube.com/watch?v=ceiUozUFF3Y&list=PL37ZVnwpeshHwJPVBqEnZild7QHWhdufu&index=41)
* **EN [Débutant]** [Anna Henningsen - Node.js and the struggles of being an EventTarget](https://www.youtube.com/watch?v=SOPC3aLoD4U&list=PL0CdgOSSGlBalMPxFFycq7OIqQF8cJS28&index=10)
* **EN [Débutant]** [Aligning Node.js with the Web Platform - James M Snell, NearForm](https://www.youtube.com/watch?v=6EDaayYnw6M)
* **EN [À savoir]** [Broken Promises by James Snell](https://www.youtube.com/watch?v=XV-u_Ow47s0&feature=emb_logo)
* **EN [À savoir]** [Broken Promises, The Workshop Edition - Matteo Collina and James Snell](https://www.youtube.com/watch?v=yRyfr1Qcf34&list=PLyspMSh4XhLP-mqulUMcaqTbLo-ZJxSX5&index=11)
* **EN [A savoir]** [Lightning Talk: Everything you Didn't Want to Know about Source Maps](https://www.youtube.com/watch?v=JuKIUYi8-Ec)
* **EN [A savoir]** [Behind the Scenes of Node.js Distributions](https://www.youtube.com/watch?v=1nYswjtEUvE&list=PLyspMSh4XhLP-mqulUMcaqTbLo-ZJxSX5&index=13)
* **EN [A savoir]** [Packaging a node.js binary for a platform or two - Irina Shestak](https://www.youtube.com/watch?v=1t1VgMaEs0I&list=PL0CdgOSSGlBalMPxFFycq7OIqQF8cJS28&index=21)
* **EN [A savoir]** [Node.js startup performance - Joyee Cheung](https://www.youtube.com/watch?v=G36lrPrF09c&list=PL0CdgOSSGlBalMPxFFycq7OIqQF8cJS28&index=18)

| **Le reste des liens sur la prochaine page** |
| --- |

* **EN [Intermédiaire]** [How Node.js bootstraps itself by Joyee Cheung](https://www.youtube.com/watch?v=bwiLlcGvFEk&list=PLfMzBWSH11xZPfWcC0DqFqKo_reMP58mw&index=52)
* **EN [Intermédiaire]** [The Unsung Hero Story of Events, Streams and Promises](https://www.youtube.com/watch?v=qOHgQAV2ydo)
* **EN [Intermédiaire]** [Async Iterators: A New Future for Streams by Stephen Belanger](https://www.youtube.com/watch?v=YVdw1MDHVZs&list=PLfMzBWSH11xZPfWcC0DqFqKo_reMP58mw&index=37)
* **EN [Intermédiaire]** [Stream into the Future - Matteo Collina, NearForm](https://www.youtube.com/watch?v=dEFdt_6fW-0&feature=emb_logo)
* **EN [Intermédiaire]** [llhttp - new HTTP 1.1 parser for Node.js by Fedor Indutny](https://www.youtube.com/watch?v=x3k_5Mi66sY&list=PL37ZVnwpeshHwJPVBqEnZild7QHWhdufu&index=15)
* **EN [Intermédiaire]** [Experimenting with remote debugging: Node.js runtime code injection](https://blog.sqreen.com/remote-debugging-nodejs-runtime-code-injection/)
* **EN NEW ! [Intermédiaire]** [Using AbortSignal in Node.js](https://www.nearform.com/blog/using-abortsignal-in-node-js/)
* **EN NEW ! [Avancé]** [Multithreaded Logging with Pino - Matteo Collina](https://www.youtube.com/watch?v=vETUVN-KEgc)

Divers articles sur des notions fondamentales en Node.js et sur différents modules du core.

* **EN [À savoir]** [Callbacks, synchronous and asynchronous](https://blog.ometer.com/2011/07/24/callbacks-synchronous-and-asynchronous/)
* **EN [A savoir]** [One-line node modules](https://github.com/sindresorhus/ama/issues/10)
* **EN [A savoir]** [Asynchronous stack traces: why await beats Promise#then()](https://mathiasbynens.be/notes/async-stack-traces)
* **EN [A savoir]** [Uncaught exceptions in Node.js](https://joyeecheung.github.io/blog/2019/08/25/uncaught-exceptions-in-node-js/)
* **EN [À savoir]** [Memory Leaks Demystified](https://nodesource.com/blog/memory-leaks-demystified)
* **EN**  **[A savoir]** [Making Promises safer in Node.js](https://www.nearform.com/blog/making-promises-safer-in-node-js/)
* **EN**  **[A savoir]** [Adding a permission system to Node.js](https://www.nearform.com/blog/adding-a-permission-system-to-node-js/)
* **EN [A savoir]** [Node.js sends warnings when you add too many listeners to an event emitter](https://www.stefanjudis.com/today-i-learned/nodejs-sends-warnings-when-you-add-too-many-listeners-to-an-event-emitter/)
* **FR [A savoir]** [Gérer le CORS avec Node.js](https://boutdecode.fr/article/cors-avec-nodejs)
* **EN NEW ! [Intermédiaire]** [A Visual Guide to Node.js Streams](https://blog.insiderattack.net/a-visual-guide-to-nodejs-streams-9d2d594a9bf5)
* **EN [Intermédiaire]** [Learning to Swim with Piscina, the node.js worker pool](https://www.nearform.com/blog/learning-to-swim-with-piscina-the-node-js-worker-pool/)
* **EN [Intermédiaire]** [Understanding Streams in Node.js](https://nodesource.com/blog/understanding-streams-in-nodejs)
* **EN [Intermédiaire]** [Understanding Worker Threads in Node.js](https://nodesource.com/blog/worker-threads-nodejs)
* **EN [Intermédiaire]** [How to improve JWT performance in Node.js](https://www.nearform.com/blog/improve-json-web-tokens-performance-in-node-js/)
* **FR [Avancé]** [Implémenter et consommer les Streams « Readable » et « Writable » de Node.js](https://blog.engineering.publicissapient.fr/2020/02/19/implementer-et-consommer-les-streams-readable-et-writable-de-node-js/)
* **EN [Avancé]** [Our Journey to a High-Performance Node.js Library](https://hazelcast.com/blog/our-journey-to-a-high-performance-node-js-library/)

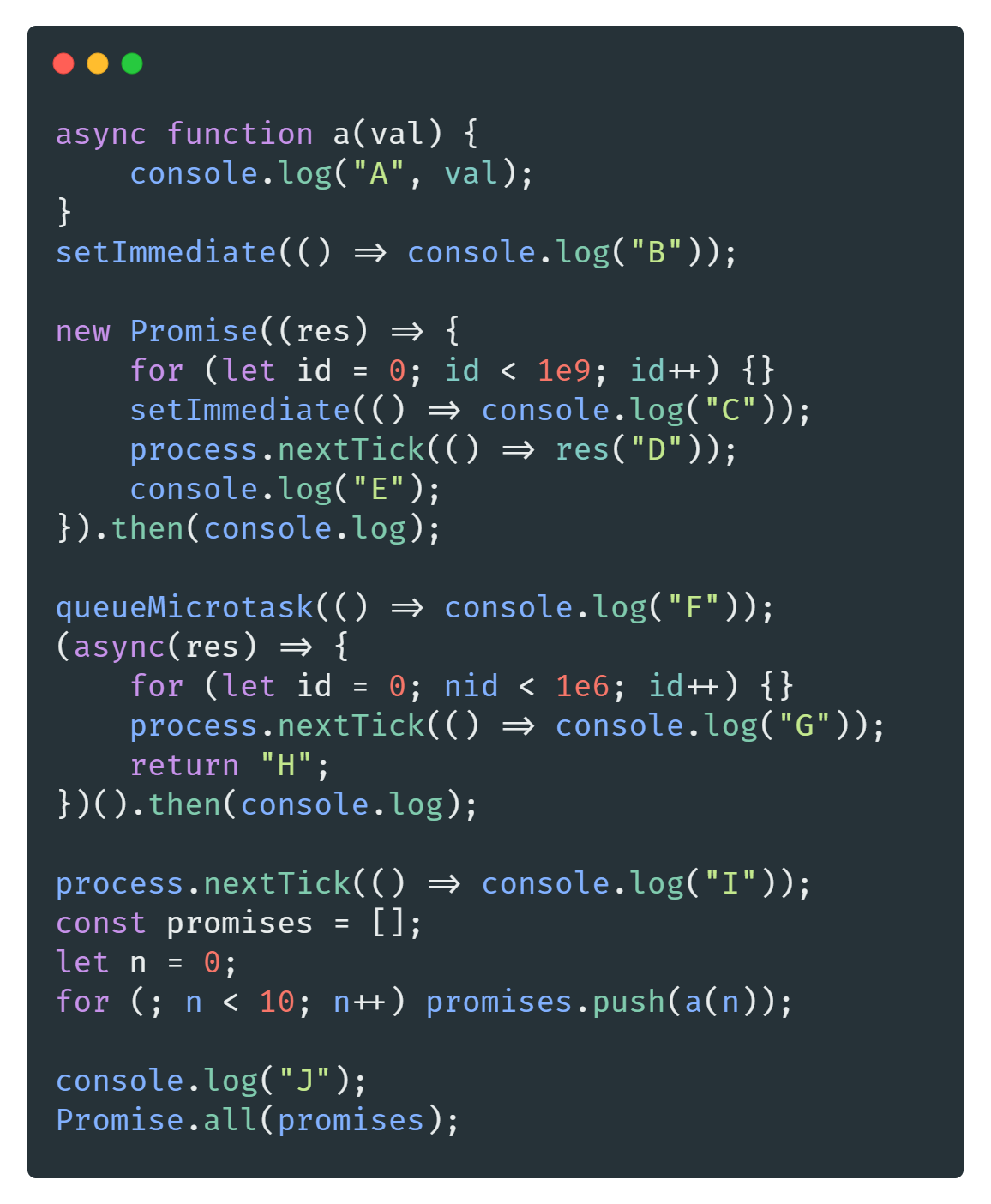
## **🎡 Event-loop**



### Introduction

Quand vous souhaitez devenir meilleur en Node.js il est important de comprendre a minima comment l’event-loop (fourni par le projet [libuv](https://github.com/libuv/libuv)) fonctionne pour savoir dans quel ordre votre code sera exécuté.

Pouvez-vous deviner l’ordre des logs ?

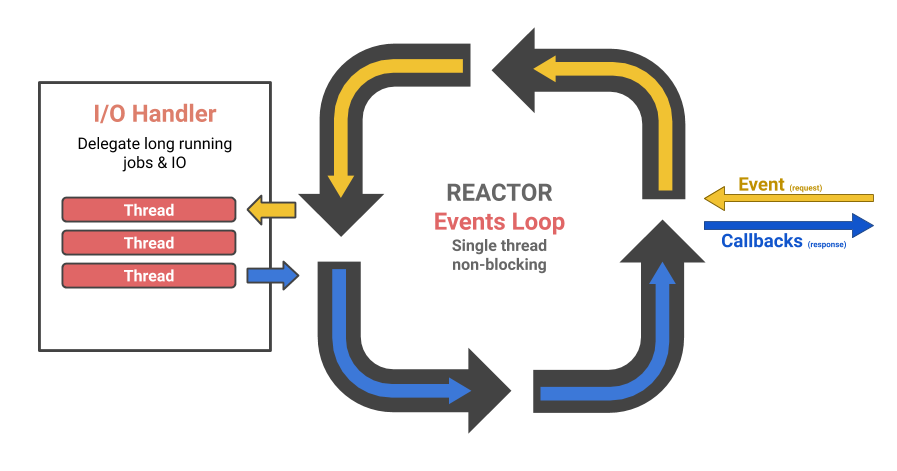


### Reactor pattern

Quand il est question d’events-loop on parle souvent de [Reactor pattern](https://en.wikipedia.org/wiki/Reactor_pattern) depuis 1996 (c'est le principe qui définit les fondamentaux et qui pourra notamment vous permettre de comprendre d’autres pattern de concurrence comme [Proactor](https://en.wikipedia.org/wiki/Proactor_pattern)).

Dans le cadre d’une Event-loop/Reactor on parlera souvent aussi de [Round-robin](https://en.wikipedia.org/wiki/Round-robin_scheduling) et de [Multiplexing](https://en.wikipedia.org/wiki/Multiplexing) (et Demultiplexing).

Schéma simple d’un Reactor (events loop).



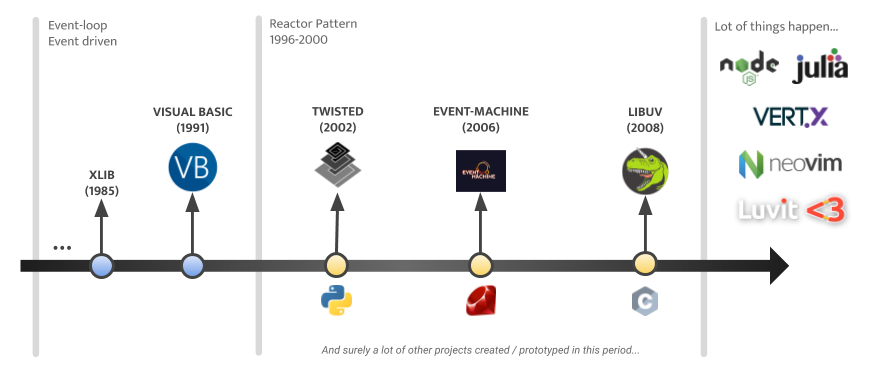
Le réacteur prend en input un évènement (lire un fichier, envoyer un paquet sur le réseau) qui aura un cycle de vie prédéfini au sein de la loop en fonction de sa nature (et de l’implémentation). Les I/O bloquant seront, la plupart du temps, gérés au sein d’abstractions bas niveau fournies par le système comme epoll, kqueue et event ports (tout dépend du système d’exploitation cible). Quand il n’est pas possible d’utiliser les ressources du système, des threads seront bien souvent créés.

Une fois le traitement terminé le réacteur s’occupera de déclencher le callback lié à l’évènement pour signaler que le traitement est terminé (avec succès ou en erreur). Je parle ici de callback pour rester bas niveau, mais il peut s’agir d’une Promise/Future ou de toute autre structure ayant pour objectif de gérer la résolution d’un événement Asynchrone.

Lien bonus pour les motivés:  **EN** [Reactor - An Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Synchronous Events](https://www.dre.vanderbilt.edu/~schmidt/PDF/reactor-siemens.pdf)

### 

### 🎥 Un peu d’histoire



La notion [d’event](https://en.wikipedia.org/wiki/Event_(computing)), [event-driven](https://en.wikipedia.org/wiki/Event-driven_programming) et [event-loop](https://en.wikipedia.org/wiki/Event_loop) ne date pas d’hier et les premières apparitions date des années 80 (même si le pattern est devenu fort populaire depuis une dizaine d’années grâce à l’apparition de lib comme Libuv ou plus récemment [Tokio](https://github.com/tokio-rs/tokio) sur Rust).

Il existe très probablement des librairies équivalentes ou des implémentations très sérieuses sur les différents runtime (Python, Ruby, PHP, Lua, Perl etc). Le langage de programmation [Julia](https://julialang.org/) est d’ailleurs basé sur Libuv.

Aujourd’hui il devient très clair que le pattern a fait ses preuves et qu’il est très largement apprécié par les développeurs du monde entier pour construire des programmes concurrents (même s’il faut toujours garder en tête qu’il y aura toujours des points forts ainsi que des points faibles).

Il n’est donc ici pas uniquement question de devenir meilleur en Node.js ou en JavaScript, mais d’acquérir des compétences et des notions qui vous seront utiles tout au long de votre carrière.

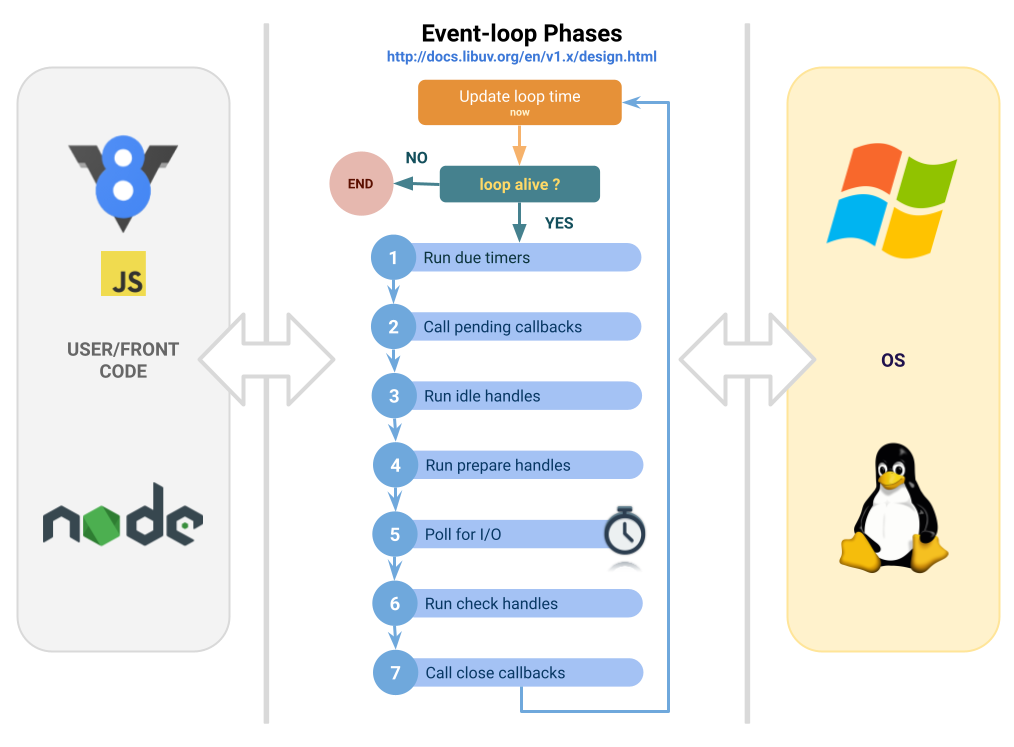
### 

### Libuv

Libuv est donc la librairie qui est utilisée dans Node.js pour l’event-loop. Son fonctionnement ne vous impacte pas directement dans votre code (elle est transparente pour les développeurs… c’est l’objectif de Node.js ^^).

Il est important de comprendre comment elle fonctionne a minima car l’exécution des différentes phases va définir comment votre code fonctionnera et dans quel ordre il sera exécuté (ce qui vous permettra de résoudre le challenge de l’introduction).

Le schéma ci-dessous est un schéma que j’ai construit pour représenter les différentes phases de l’event-loop (vous noterez la claire séparation entre votre code, la loop et le système d’exploitation).

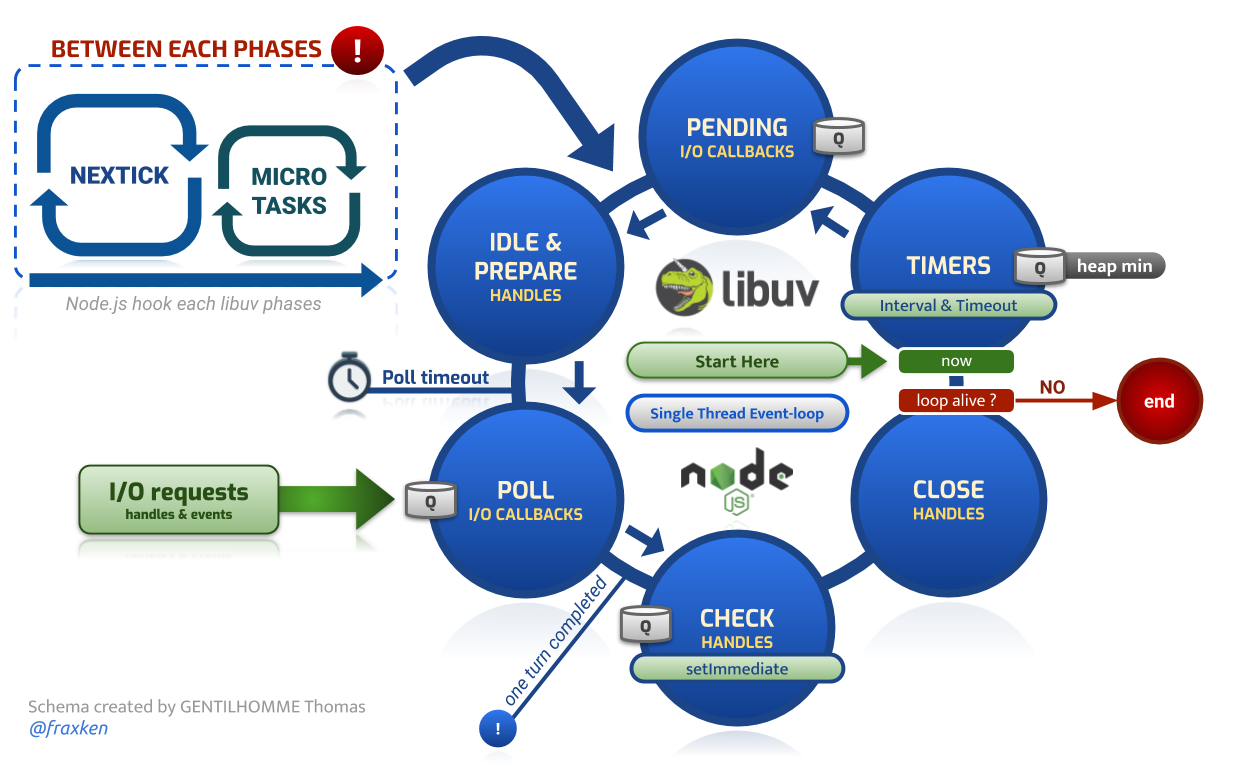


Sur le sujet je vous recommande d’aller lire en premier lieu les pages suivantes :

* **EN [Débutant]** [The Node.js Event Loop, Timers, and process.nextTick()](https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/)
* **EN [Débutant]** [Libuv design overview](http://docs.libuv.org/en/v1.x/design.html) (documentation officielle de Libuv).
* **EN [Avancé]** [An introduction to libuv](https://nikhilm.github.io/uvbook/An%20Introduction%20to%20libuv.pdf).

### 📜 Articles et talks

Node.js event-loop par moi ([version HD ici](https://docs.google.com/drawings/d/1UOJqpU8Zwju7kaPRFBOtbO0iRGBuzW_gr2zzKFOHaAU/edit?usp=sharing)).



Divers articles de vulgarisation. Ils peuvent vous permettre de mieux comprendre divers sujets vus plus haut d’une façon plus abordable :

* **EN [Débutant]** [What you should know to really understand the Node.js Event Loop](https://medium.com/the-node-js-collection/what-you-should-know-to-really-understand-the-node-js-event-loop-and-its-metrics-c4907b19da4c)
* **EN [Débutant]** [Event Loop and the Big Picture — NodeJS Event Loop Part 1](https://blog.insiderattack.net/event-loop-and-the-big-picture-nodejs-event-loop-part-1-1cb67a182810)
* **EN [Débutant]** [Timers, Immediates and Process.nextTick— NodeJS Event Loop Part 2](https://blog.insiderattack.net/timers-immediates-and-process-nexttick-nodejs-event-loop-part-2-2c53fd511bb3)
* **EN [Débutant]** [Promises, Next-Ticks, and Immediates— NodeJS Event Loop Part 3](https://blog.insiderattack.net/promises-next-ticks-and-immediates-nodejs-event-loop-part-3-9226cbe7a6aa)
* **EN [Débutant]** [JavaScript Visualized: Event Loop (pas forcément en lien direct avec Node.js)](https://dev.to/lydiahallie/javascript-visualized-event-loop-3dif)
* **EN [Intermédiaire]** [Introduction to Event Loop Utilization in Node.js](https://nodesource.com/blog/event-loop-utilization-nodejs)

Divers talks sur Node.js et libuv (les deux derniers sont en français) :

* **EN [Débutant]** [Everything You Need to Know About Node.js Event Loop](https://www.youtube.com/watch?v=PNa9OMajw9w)
* **EN [Débutant]** [Introduction to libuv: What's a Unicorn Velociraptor?](https://www.youtube.com/watch?v=_c51fcXRLGw&list=PLfMzBWSH11xZPfWcC0DqFqKo_reMP58mw&index=6)
* **EN [Débutant]** [The Node.js Event Loop: Not So Single Threaded](https://www.youtube.com/watch?v=zphcsoSJMvM)
* **EN [A savoir]** [Node.js Event-Loop: How even quick Node.js async functions can block the Event-Loop, starve I/O](https://snyk.io/blog/nodejs-how-even-quick-async-functions-can-block-the-event-loop-starve-io/)
* **EN [Intermédiaire]** [Uncovering Libuv secrets, a practical approach - Santiago Gimeno](https://www.youtube.com/watch?v=0DF8uYO7RV0&list=PL0CdgOSSGlBalMPxFFycq7OIqQF8cJS28&index=17)
* **EN [Intermédiaire]** [LXJS 2012 - Bert Belder - libuv](https://www.youtube.com/watch?v=nGn60vDSxQ4)
* **FR [Débutant]** [Weektalk#1 ES-Community sur l’event-loop (date un petit peu mais toujours cool).](https://www.youtube.com/watch?v=zv1oFDkP2tI)
* **FR [Débutant]** [Apprendre Node.js #5 - L'event loop](https://www.youtube.com/watch?v=g25LIAIlcbE) (Vulgarisation de Louistiti).

## 

## **👽 Native API (création d’addon natif en C, C++ et Rust)**

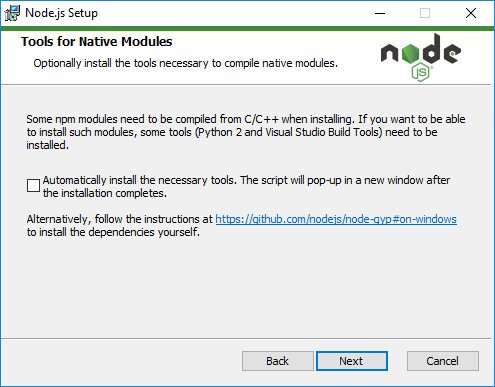
Avec Node.js il est aussi possible de créer des [interfaces (addons) bas niveau](https://nodejs.org/api/n-api.html) pour par exemple exécuter des librairies écrites en C, C++.

Cela peut amener un avantage en matière de performance (mais ce n’est pas forcément tout le temps l’objectif.. passer de C++ à JavaScript a un coup non négligeable). Quelquefois c’est aussi tout simplement car les librairies C/C++ sont déjà très complètes et maintenues (ce serait potentiellement une perte de temps de tout ré implémenter en JavaScript).

Quelques exemples de packages NPM étant des addons natif :

* [node-argon2](https://github.com/ranisalt/node-argon2)
* [node-sqlite3](https://github.com/mapbox/node-sqlite3)
* [Windrive (fait par moi même pour récupérer des informations du Système)](https://github.com/SlimIO/Windrive)

D’ailleurs pour installer ces packages, il vous est bien souvent nécessaire d’installer diverses dépendances pour être en capacité de les compiler. Je vous invite à consulter la section [installation](https://github.com/nodejs/node-gyp#installation) de Node-gyp si vous n’étiez pas au courant de cela. (Il y a aussi une option à l’installation de Node.js sur Windows).



Si c’est un sujet qui vous intéresse j’ai [moi-même fait un talk en 2019](https://www.youtube.com/watch?v=rvmnnlYf3lk) pour vulgariser et expliquer mes premiers pas en création d’addons natif pour mon projet SlimIO.

Voici néanmoins une liste complémentaire de ressources :

* **EN** [Node-addon-api](https://github.com/nodejs/node-addon-api) (Pour ceux qui veulent écrire du code en C++).
* **EN** [Node-gyp](https://github.com/nodejs/node-gyp) (C’est l’outil qui va vous servir à compiler votre addon).
* **EN** [Node.js addons examples](https://github.com/JoseJPR/nodejs-addons)
* **EN** [Prebuildify](https://github.com/prebuild/prebuildify) (Pour la création de prebuilds).

Documentation et outil GYP :

* **EN** [GYP User Documentation](https://gyp.gsrc.io/docs/UserDocumentation.md)
* **EN** [GYP Input Format Reference](https://gyp.gsrc.io/docs/InputFormatReference.md)
* **EN** [GYP Parser](https://github.com/addaleax/gyp-parser)

Talks sur la native API de Node.js :

* **EN** [N-API - Next Generation Node API for Native Modules](https://www.youtube.com/watch?v=-Oniup60Afs)
* **EN** [N-API: The Next Generation Node.js API is Ready!](https://www.youtube.com/watch?v=BrJcsYjp8Nw&list=PLfMzBWSH11xaZvhv1X5Fq1H-oMdnAtG6k&index=54)
* **EN** [Next Generation N-API: A Hands-on Workshop](https://www.youtube.com/watch?v=-v4Q0y4CeRA&list=PLfMzBWSH11xZPfWcC0DqFqKo_reMP58mw&index=44)
* **EN** [N-API on JerryScript](https://www.youtube.com/watch?v=Pxabz_FA1IU&list=PLfMzBWSH11xaZvhv1X5Fq1H-oMdnAtG6k&index=69)

### Rust

Divers liens pour faire des addons natif avec Rust plutôt que C et C++ :

* [Node.js N-API for Rust.](https://www.youtube.com/watch?v=UzTPBy2acio)
* [NAPI-RS](https://github.com/napi-rs)
* **EN NEW !** [Supercharge Your Node.js With Rust](https://yieldcode.blog/supercharge-nodejs-with-rust/)

## 

## **📟 CLI**

L’écosystème Node.js et JavaScript est très vaste et donc très intéressant pour la construction d'outils de type [CLI](https://fr.wikipedia.org/wiki/Interface_en_ligne_de_commande) (command line interface).

Ces outils sont très couramment utilisés et développés pour simplifier la vie aux développeurs et intégrateurs dans divers projets. Beaucoup de framework et librairies embarquent d’ailleurs leur propre CLI pour offrir aux développeurs une expérience optimale. Nous les utilisons au quotidien (parfois sans même nous en rendre compte).

Je recommande souvent au débutant de découvrir Node.js par le biais de la construction d’un CLI (par exemple un petit jeu RPG au format texte jouable avec des commandes.. c’est vraiment une idée de petit projet qui vous pousse à améliorer votre niveau de maîtrise en JavaScript).

Si vous voulez vous y mettre, je vous recommande [ce petit guide](https://x-team.com/blog/a-guide-to-creating-a-nodejs-command/) ou également [Créer un outil en ligne de commande](https://oncletom.io/node.js/chapter-08/) (français) qui va vous apprendre à mettre en place une CLI simple. Aussi je vous recommande de lire la doc [npm link](https://docs.npmjs.com/cli/v6/commands/npm-link) ainsi que la doc sur le champ [bin](https://docs.npmjs.com/cli/v6/configuring-npm/package-json#bin) du package.json.

D’autres ressources (talks et articles) sur le sujet :

* **EN** [Building Great CLI Experiences in Node](https://www.youtube.com/watch?v=Izx3-KSuaM8&list=PLfMzBWSH11xaZvhv1X5Fq1H-oMdnAtG6k&index=45)
* **EN** [Building Interactive npm Command Line Modules](https://www.youtube.com/watch?v=QLat0Y3jqUA)
* **EN** [CLI Tools for All the Things](https://www.youtube.com/watch?v=E0Oz5s9ZjKY)
* **EN** [Ma liste d'awesome crafted packages CLI](https://github.com/fraxken/awesome-crafted-nodejs#cli-tty-etc)
* **EN** [Node.js CLI Apps Best Practices](https://github.com/lirantal/nodejs-cli-apps-best-practices)

Quelques packages que je vous recommande :

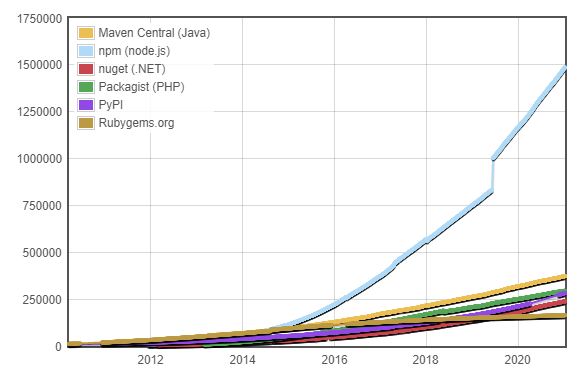
* [kleur](https://github.com/lukeed/kleur) (pour faire du texte en couleur dans le terminal, il y a aussi [chalk](https://github.com/chalk/chalk)).
* [inquirer.js](https://github.com/SBoudrias/Inquirer.js) (il y a aussi [enquirer](https://github.com/enquirer/enquirer)).
* [cliui](https://github.com/yargs/cliui)
* [sade](https://github.com/lukeed/sade)
* [boxen](https://github.com/sindresorhus/boxen)
* [Elegant terminal spinner](https://github.com/sindresorhus/ora)
* [gauge](https://github.com/npm/gauge)
* [qoa](https://github.com/klaussinani/qoa)

Si jamais vous souhaitez voir un cas concret je vous invite à regarder le CLI d’un de mes projets [Node-secure](https://github.com/ES-Community/nsecure) ou bien [npq](https://github.com/lirantal/npq) de Liran tal.

## **📦 Packages**

L’utilisation et la création de packages dans l’écosystème JavaScript est une réelle force et une source d'incroyables opportunités.

(comptage des modules par écosystème)



Malheureusement beaucoup n’arrivent pas à y voir l’émergence d’un pattern qui est de pouvoir segmenter des projets très simplement en différents modules qui seront bien plus facilement maintenables et testables sur long terme. La majorité des développeurs ne réalisent pas la puissance même du système de modules qu’ils ont entre les mains (et il n’est ici pas uniquement question d’open-source ou de npm).

C’est une des remarques que je fais le plus à mes clients en tant qu’Expert Node.js... Tout le monde a une tendance à construire par instinct des monolithes non-modulaire alors que par exemple les algorithmes en lien avec le métier peuvent être gérés au sein d’un module à part (ce genre de détails peut accélérer la productivité d’une équipe par un ratio de 5.. **c’est du vécu**).

Et il n’est pas question ici de nécessairement segmenter en 50 modules... 2 à 4 modules c’est déjà un bon début pour une équipe.

Si vous avez l’occasion ou une idée de packages alors je vous recommande fortement d’essayer l’aventure au moins une fois.

### Articles et talks

* **EN** [Getting started NPM](https://docs.npmjs.com/getting-started)
* **EN** [NodeSchool - How to NPM](https://github.com/workshopper/how-to-npm)
* **EN NEW !** [The package.json guide](https://nodejs.dev/learn/the-package-json-guide)
* **EN** [CJ Silverio, CTO of npm](https://www.youtube.com/watch?v=HH3aNjjhMg8)
* **EN** [npm and the Future of JavaScript - Laurie Voss, npm, Inc.](https://www.youtube.com/watch?v=0PU-4GGLzGg)
* **EN** [Radical Modularity - Aria Stewart, npm, Inc.](https://www.youtube.com/watch?v=SsIdWFtp2QA)
* **EN** [Awesome npm](https://github.com/sindresorhus/awesome-npm)

Il est possible d’utiliser la commande [npm link](https://docs.npmjs.com/cli/v6/commands/npm-link) pour travailler avec des packages ou des modules qui ne sont pas encore publiés sur le registre npm.

### **NEW !** Workshops

Vous êtes débutant et vous voulez vous entraîner ? N’hésite pas à découvrir le workshop suivant qui est vraiment simple:  **EN**  [How to npm](https://github.com/workshopper/how-to-npm)

### 

### Monorepo & workspaces

Il peut aussi être intéressant d’explorer la piste des monorepos et workspaces qui sont maintenant inclus par défaut sur yarn et npm 7. Il est aussi possible d’utiliser le package lerna.

C’est une solution très intéressante notamment pour n’avoir qu’un seul repo git mais tout en conservant l’avantage de la segmentation en modules/packages. Quelques liens pour vous documenter sur le sujet :

* **EN** [Simplify your monorepo with npm 7 workspaces](https://dev.to/limal/simplify-your-monorepo-with-npm-7-workspaces-5gmj)
* **EN** [npm Workspaces (RFC)](https://github.com/npm/rfcs/blob/latest/implemented/0026-workspaces.md)
* **EN** [Workspaces in Yarn](https://classic.yarnpkg.com/blog/2017/08/02/introducing-workspaces/)
* **EN** [pnpm Workspace](https://pnpm.js.org/en/workspaces)
* **EN** [🐉 A tool for managing JavaScript projects with multiple packages.](https://github.com/lerna/lerna)

Je commence pour ma part à peine à les utiliser à travers différents projets, donc je compléterai la section au cours des prochains mois.

## 

## **🔍 Debugging & Profiling**

S’améliorer et s’éveiller sur des sujets comme le debug est très important en tant que développeur pour ne pas être perdu sur les actions à suivre lorsque l'on rencontre un souci un peu chevelu (qui demandera donc un peu plus que quelque console.log).

| Notez tout de même que si vous avez besoin d’aller aussi loin, il y a peut-être un souci dans votre façon d 'architecturer votre code. |
| --- |

Voici une liste de ressources qui pourraient vous êtres nécessaire de lire :

* **EN** [Node.js debugging best practices - Joyent](https://www.joyent.com/node-js/production/debug)
* **EN** [Node.js official debugging guide](https://nodejs.org/en/docs/guides/debugging-getting-started/)
* **EN** [Node.js debugger API](https://nodejs.org/api/debugger.html)
* **EN** [VS Code integrated tools for Node.js](https://code.visualstudio.com/docs/nodejs/nodejs-tutorial#_debugging-your-node-application)
* **EN** [Easy profiling for Node.js Applications](https://nodejs.org/en/docs/guides/simple-profiling/)
* **EN** [Create a Dump of the V8 Heap in Nodejs](https://medium.com/better-programming/make-a-dump-of-the-v8-heap-and-inspect-for-your-node-app-b69f7b68c162)
* **EN** [Node.js console API](https://nodejs.org/api/console.html) (car il n’y a pas que console.log ^^).
* **EN** [Clinic.js rises to the top in diagnosing Node.js performance issues](https://www.nearform.com/blog/clinicjs-rises-top-diagnosing-nodejs-performance/)
* **EN** [Experimenting with remote debugging: Node.js runtime code injection](https://blog.sqreen.com/remote-debugging-nodejs-runtime-code-injection/)

Si vous êtes plutôt à la recherche d’outils (pour de la génération de flamegraphs par exemple).

* **EN** [0x🔥](https://github.com/davidmarkclements/0x)
* **EN** [Clinicjs Flame](https://clinicjs.org/flame/) (l’outil possède bien d’autres modes que je vous recommande aussi).
* **EN** [Investigates v8/Node.js function deoptimizations.](https://github.com/thlorenz/deoptigate)
* **EN** [Collect runtime type information 😻 of your JavaScript code.](https://github.com/fhinkel/type-profile)
* **EN** [Why is node running](https://github.com/mafintosh/why-is-node-running) (cool pour trouver une référence non close).

Si vous avez besoin de faire du benchmark vis à vis de vos tests alors je vous recommande [autocannon](https://github.com/mcollina/autocannon).

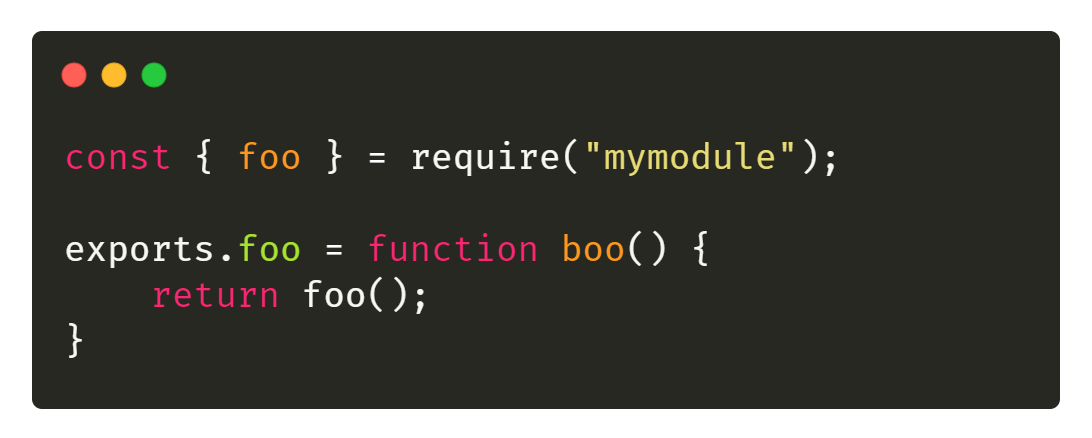
## 

## **Modules (CJS & ESM)**

En Node.js il existe deux façons de gérer des modules:

* [CJS](https://nodejs.org/api/modules.html) pour Common JavaScript
* [ESM](https://nodejs.org/api/esm.html) pour ECMAScript modules

ESM est le nouveau standard intégré depuis ECMAScript 6 pour la gestion des modules. Néanmoins historiquement Node.js utilise CJS depuis le début et à l’époque il y a eu beaucoup de soucis sur la spécification d’ESM (ce qui a rendu complexe l’implémentation dans l’écosystème Node js).



Avec CJS il est nécessaire d’utiliser la function **require()** et des mots-clés comme **module.exports** alors qu’en ESM il vous sera possible d’utiliser les mots-clés **import** et **export**.



Le support d’ESM sur Node.js devient de plus en plus stable et il n’est donc pas étonnant que cela deviennent le système de modules utilisé par défaut dans un futur proche.

Je vous recommande d’écouter les talks suivant pour mieux comprendre le sujet:

* **EN** [Extra Special Modules - Myles Borins, Google](https://www.youtube.com/watch?v=bP0tlIcA3ns)
* **EN** [Keynote: The Future of JavaScript is Universal - Myles Borins, Developer Advocate, Google](https://www.youtube.com/watch?v=o2M7g8Xwc7g)
* **EN** [ES Modules in Node.js - Gil Tayar](https://www.youtube.com/watch?v=mMEKrHmZLpQ&list=PL0CdgOSSGlBalMPxFFycq7OIqQF8cJS28&index=7)
* **EN** [Testing EcmaScript Modules - David Mark Clements](https://www.youtube.com/watch?v=ZdrNw7zxa5Y)

Articles en lien avec ESM

* **EN** [Get Ready For ESM - Sindresorhus](https://blog.sindresorhus.com/get-ready-for-esm-aa53530b3f77)
* **EN** [Pure ESM package - Sindresorhus](https://gist.github.com/sindresorhus/a39789f98801d908bbc7ff3ecc99d99c)
* **EN NEW !** [Using ES Modules (ESM) in Node.js: A Practical Guide (Part 1)](https://gils-blog.tayar.org/posts/using-jsm-esm-in-nodejs-a-practical-guide-part-1/)

## 

## **WebSocket**

Il n’est pas rare qu’en tant que Junior ou nouveau venu, une de nos idées soit la réalisation d’une application utilisant du temps réel par le biais des WebSockets (que ce soit pour un tchat, un jeu ou autres).

Bien souvent tout le monde aura le réflexe de débuter avec la librairie [socket.io](https://socket.io/) (ce que j’ai moi-même fait) qui est plutôt accessible pour les débutants et vous apporte un tas de fonctionnalités cool comme les rooms.

Néanmoins, maintenant avec mon expérience, je vous recommanderais aussi de prendre sérieusement en considération le fait de ne pas utiliser directement socket.io qui va vous abstraire bon nombre d'éléments et qui peut parfois vous empêcher de mieux comprendre différents fondamentaux (ou de profiter de fonctionnalités natives).

C’est aussi je pense un excellent entraînement (si vous en êtes capable alors socket.io ce sera un camp de vacances à côté).

Bref n’hésitez pas à vous intéresser à l’API native [WebSockets](https://developer.mozilla.org/fr/docs/Web/API/WebSockets_API) du navigateur (et le package npm [WebSockets](https://www.npmjs.com/package/websocket) pour le back-end). Vous pouvez aussi vous intéresser à [gRPC](https://www.youtube.com/watch?v=fl9AZieRUaw) qui utilisera notamment [protocol buffers](https://developers.google.com/protocol-buffers) pour les messages (plus technique mais ça fait tout de suite moins tâche sur votre CV).

Si votre besoin nécessite de très haute performance je vous recommande de jeter un œil à [uWebSockets](https://github.com/uNetworking/uWebSockets.js) (qui possède un système de room minimaliste aussi).

| Faite néanmoins attention à la mentalité abrasive du mainteneur (il n'est pas connu pour être un enfant de chœur et il peut être parfois assez méchant). |
| --- |

## 

## **📊 Test unitaire et coverage**

Réaliser des tests unitaires est [une bonne solution pour s'entraîner](https://github.com/fraxken/test-challenge) et mieux comprendre comment un code X ou Y fonctionne (et vous force à étudier diverses fonctionnalités de JavaScript). Néanmoins il peut vite être difficile de s’y retrouver dans un écosystème où le choix de framework/lib est vaste :

[Mocha](https://mochajs.org/), [Ava](https://github.com/avajs/ava), [Jest](https://jestjs.io/), [Japa](https://github.com/thetutlage/japa), [Tape](https://www.npmjs.com/package/tape), [uvu](https://github.com/lukeed/uvu) et beaucoup d’autres...

Certains auront l’avantage d’être plus complet (plus lourd) et d’autres plus simpliste. Quelquefois cela se joue sur différents détails comme le choix de la librairie d’assertion ([Chai.js](https://www.chaijs.com/) par exemple) ou bien l’inclusion du coverage par défaut.

En ce qui me concerne, après plusieurs années d’expérience avec ces différentes librairies, ma préférence penche plutôt pour [Jest](https://jestjs.io/) qui offre une expérience d’ensemble plutôt complète et agréable. Néanmoins, il m’arrive pour des projets très spécifiques, d’utiliser des libs de testing simples en combinaison avec C8 (de plus [Jest peut s’avérer problématique à cause de la réécriture du scope global sur Node.js](https://github.com/facebook/jest/issues/2549)).

Lorsque le coverage n’est pas inclus par défaut il vous faudra potentiellement réfléchir à l’inclure vous même avec [Nyc](https://github.com/istanbuljs/nyc) ou [C8](https://github.com/bcoe/c8) (petit frère plus récent et plus complet utilisant [le coverage natif de V8 Engine](https://v8.dev/blog/javascript-code-coverage)). C8 est notamment capable d’offrir le coverage même quand le code est exécuté au travers de différents child process (ou worker).

Talks et articles:

* **EN** [Rethinking JavaScript Test Coverage](https://www.youtube.com/watch?v=SKNySdVs1DA&list=PLfMzBWSH11xZPfWcC0DqFqKo_reMP58mw&index=5)
* **EN** [Workshop: Dead-Simple Testing with Mocha](https://www.youtube.com/watch?v=JhQ-PuwoWAE&list=PLyspMSh4XhLP-mqulUMcaqTbLo-ZJxSX5&index=24)
* **EN** [Rogelio Guzman - Jest Snapshots and Beyond](https://www.youtube.com/watch?v=HAuXJVI_bUs&feature=emb_logo)
* **EN** [Establishing testing patterns with software design principles](https://www.youtube.com/watch?v=_pnW-JjmyXE&feature=emb_logo)
* **EN** [Comprehensive and exhaustive JavaScript & Node.js testing best practices (January 2021)](https://github.com/goldbergyoni/javascript-testing-best-practices)
* **EN** [JavaScript testing best practices](https://github.com/goldbergyoni/javascript-testing-best-practices)

### 

### 💃 Méthodologies

De nos jours, il n’est pas rare que les juniors soient **forcés** à appliquer des méthodologies sans que leur mentor n’apporte de réelle réflexion ou arguments: “**c’est comme ça si tu veux devenir un bon développeur**”.

Je pense qu’apprendre à écrire des tests est essentiel pour un développeur (et que c’est plutôt sur cette fondation commune que nous devons nous appuyer pour guider les débutants).

Il est important de conserver un fort esprit critique sur les différents choix que l'on essayera de vous imposer comme des pratiques qu'il faut **constamment appliquer** car celles-ci vous enferment certainement dans une bulle idéologique (**ce qui n'est pas une invitation à ne rien faire**).

Par exemple, apprendre TDD et autres méthodologies sera bénéfique pour ajouter des cordes à votre ARC (surtout sur certains projets où leur pratique sera une plus- value). Néanmoins, cela ne veut pas dire qu’elles sont absolues à l’intégralité des contextes et projets **ou qu’il ne faut pas en débattre**.

Quelques liens pour vous faire une “opinion”:

* **FR** [BDD, DDD, ATDD et TDD expliqués !](https://www.youtube.com/watch?v=jxBmKvS7lAo)
* **EN** [Why Most Unit Testing is Waste](https://rbcs-us.com/documents/Why-Most-Unit-Testing-is-Waste.pdf)
* **EN** [Test-induced design damage.](https://dhh.dk/2014/test-induced-design-damage.html)
* **EN** [TDD is dead. Long live testing.](https://dhh.dk/2014/tdd-is-dead-long-live-testing.html)
* **EN** [TW Hangouts | Is TDD dead?](https://www.youtube.com/watch?v=z9quxZsLcfo)
* **EN** [TDD, Where Did It All Go Wrong](https://www.youtube.com/watch?v=EZ05e7EMOLM)
* **FR** [LMHB #5: KEEP CALM & TDD ! FEAT. LIODA (Test-driven development)](https://www.youtube.com/watch?v=sFTgS0qxNa0)

Dans notre domaine nous parlons très peu de cela tellement ils sont sujets à des tensions extrêmes entre nous. Je trouve quelquefois dommage que l'on ne puisse pas discuter sans y amener son lot de toxicité et d’ego.

## 

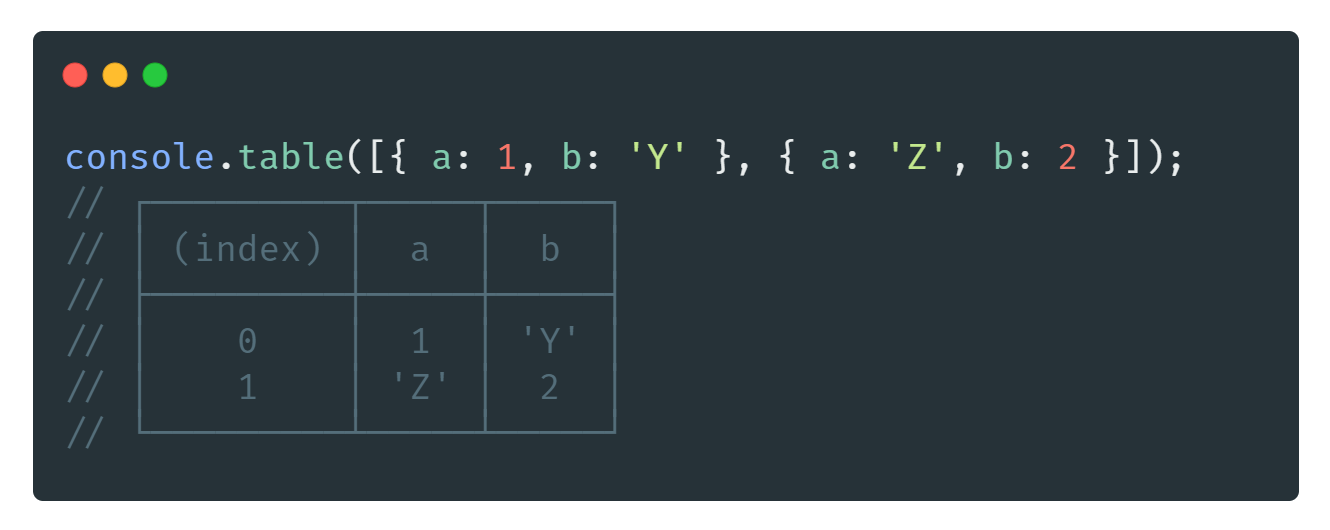
## **🌟 Les différents modules core**

### Console

Le module [console](https://nodejs.org/api/console.html) est celui que nous utilisons le plus (par le biais des méthodes console.log et console.error par exemple). Évidemment ici l’objectif est de simuler le plus possible le navigateur pour ne pas avoir à gérer les sorties stdout et stderr directement nous-mêmes par le biais de la variable process (ce qui peut être tout de même intéressant, je vous invite à aller voir la doc [TTY](https://nodejs.org/api/tty.html) pour y découvrir des manipulations bien plus avancées de la sortie standard).

Bien souvent les développeurs n’utilisent pas ¼ des méthodes présentes dans le module et c’est plutôt dommage. Prenez le temps d’explorer la documentation pour découvrir les différentes méthodes et ce qu’elles vous permettent de faire.

Quelques exemples;



Connaître (approximativement) le temps d’exécution d’un code avec console.time.



### 

### Events

Le module [Events](https://nodejs.org/api/events.html) va permettre de créer et manipuler un des éléments les plus importants de Node.js: l’Event Emitter (il y a aussi depuis peu Event Target qui est une API compatible avec le navigateur).

Les différents modules de Node.js comme HTTP, Net, Child process ainsi que les Streams sont majoritairement construits à l’aide d’Event Emitter.



Exemple ci-dessus avec un Read Stream ou nous exploitons les évènements “data”, “end” ou encore “error” (**merci de pas reproduire ce code en production ^^**).

Ce sont des structures complètement **Synchrone** dans lesquelles nous pouvons lier un nom d’évènement (emitter) à une ou plusieurs functions (listeners). Les évènements qui seront émis sont gérés en FIFO (First in First out).

Cela va nous permettre de mettre en place diverses patterns ou le code fonctionnera par le biais de communication entre différents objets.

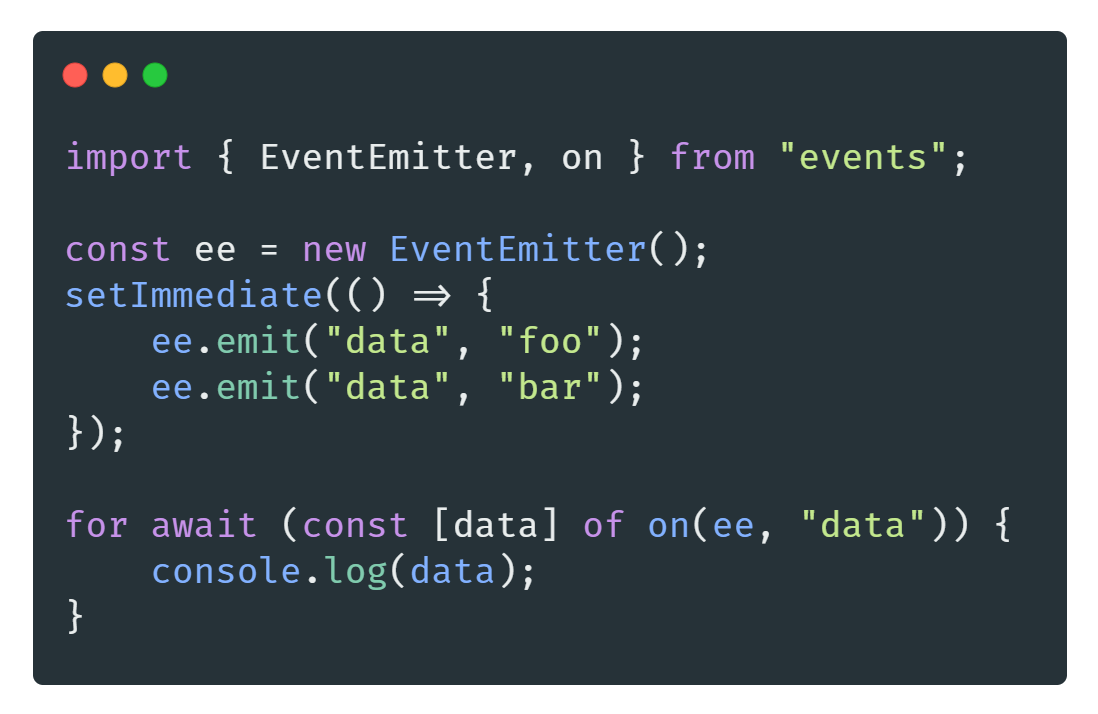


Une class en JavaScript peut s'étendre d’un EventEmitter.



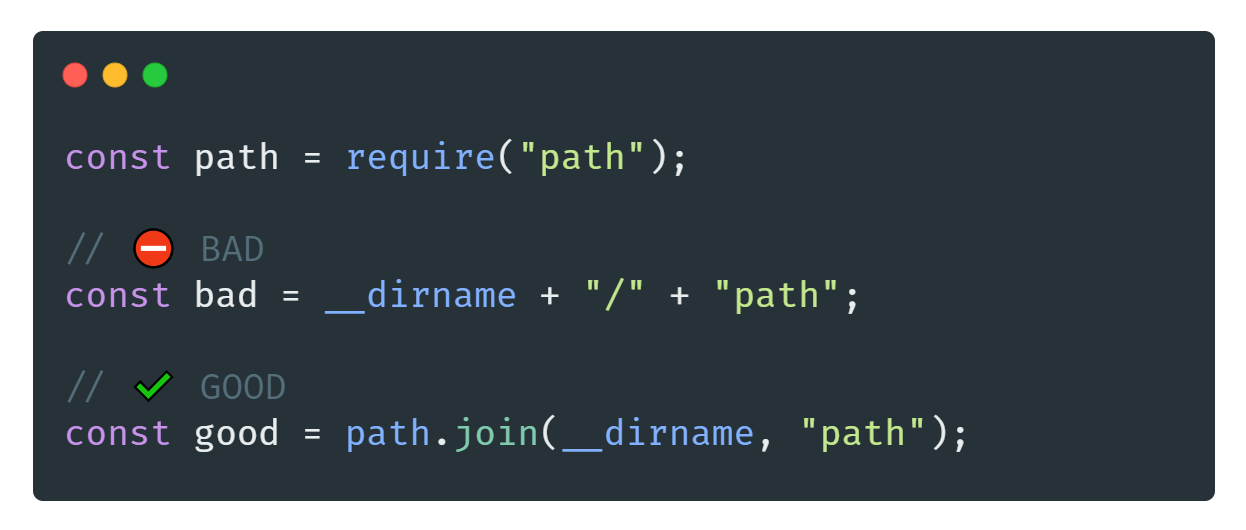
C’est pratique si vous voulez construire un système modulaire ou différent objets (class) communiquent par le biais de messages (les Addons dans SlimIO utilisent beaucoup ce principe).

Il est aussi bon de noter que depuis quelques versions la méthode “on” d'events est AsyncIterable.



### Path

Le module [path](https://nodejs.org/api/path.html) fournit des utilitaires pour travailler avec les chemins de fichiers et de répertoires.

Il sera la plupart du temps utilisé pour de la composition de chemin (si vous les faites à la **main vous-même c’est que votre code n’est probablement pas cross-plateforme**).

Les chemins ne sont pas construits de la même manière selon le système d’exploitation qui sera utilisé (UNIX, Windows, MAC etc). Si vous êtes intéressés par les bonnes pratiques d'écriture de code cross-plateforme, je vous recommande [Awesome Node.js cross platform](https://github.com/bcoe/awesome-cross-platform-nodejs).

Le module path possède aussi un ensemble de méthodes qui vont vous permettre de récupérer les différentes parties qui composent le chemin d’un fichier ou dossier. Comme par exemple l’extension d’un fichier (que tous les juniors s’amusent à récupérer avec une RegExp ou avec des bidouilles .split etc).



### 

### FileSystem (fs)

Comme son nom l’indique ce module permet d'interagir avec le système de fichiers. Si vous avez besoin de lire, modifiez un fichier, alors ce qu’il vous faut est très clairement dans ce module (et il en va de même pour les dossiers).

**Bonus**: **NEW !** [Article sur le module FS pour les débutants](https://catalins.tech/a-beginners-guide-to-the-file-system-module-in-nodejs).

Pour un junior ce module est une bonne opportunité pour s'entraîner sur la maîtrise de l’Asynchrone (Promise, Callback) sans nécessité de déployer de grand moyen (vous pouvez simuler des scénarios divers et variés sur votre OS).

Par exemple aller récupérer la taille d’un répertoire, les extensions de tous les fichiers dans le répertoire etc.. Et optimalement de manière asynchrone en évitant au possible de faire des séries (en utilisant des méthodes comme [Promise.all](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Promise/all) et [Promise.allSettled](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Promise/allSettled)).



Les opérations sur FS sont gérées en arrière fond par Libuv et distribuées sur [des Threads](http://docs.libuv.org/en/v1.x/threadpool.html).

### 

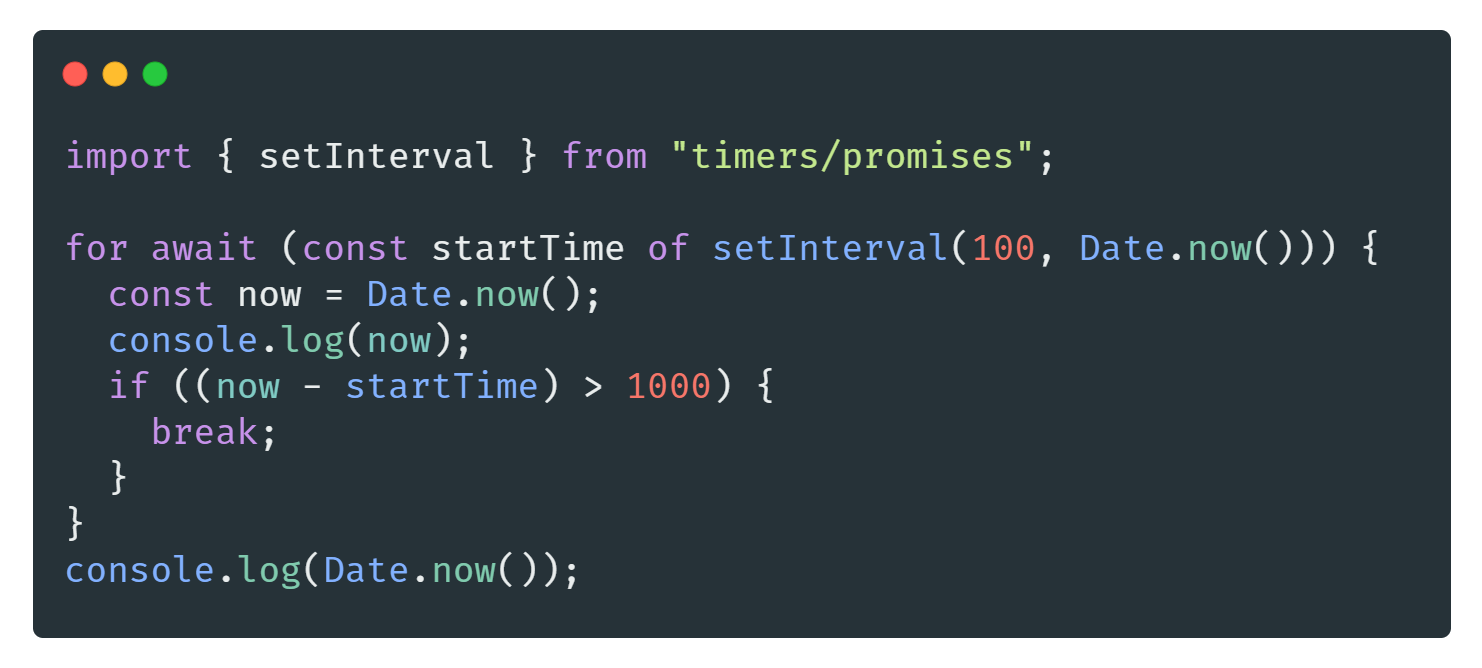
### Timers

Le module [Timers](https://nodejs.org/api/timers.html) de Node.js contient des fonctions qui exécutent le code après une période de temps déterminée. Il n'est pas nécessaire d'importer les timers via require(), car toutes les méthodes sont disponibles globalement pour émuler l'API JavaScript du navigateur.

| Je vous recommande d’aller lire le guide officiel de Node.js sur les [timers](https://nodejs.org/en/docs/guides/timers-in-node/). |
| --- |

Les functions disponibles sont le setImmediate, setTimeout et setInterval.

**setImmediate** est une function unique à Node.js (elle n’existe pas dans le navigateur) et elle permet d’exécuter du code au début de la prochaine itération de l’event-loop.



Noter qu’il est possible d’appeler la méthode unref sur un timer pour que l’event-loop puisse s’arrêter même si le timer est encore actif. C’est très utile dans les cas où vous ne voulez absolument pas qu’un timer (interval par exemple) puisse empêcher l’event-loop de s’arrêter.

C’est par exemple le cas dans le package [SlimIO/Config](https://github.com/SlimIO/Config/blob/master/src/config.class.js#L272) ou un setInterval est utilisé pour constamment nettoyer des Observables qui ont été clôturés (pour éviter un memory-leak). Néanmoins je ne souhaite pas que le timer en question empêche l’arrêt de l’event-loop dans le cas où quelqu’un utiliserait mon package :)

### 

### URL (API WHATWG)

Implémentation de l’API [URL](https://nodejs.org/api/url.html) du navigateur. Par défaut disponible sur le namespace Global comme pour les timers. URL est vraiment pratique pour la création et manipulation d’URL (comme pour path beaucoup de développeurs peu expérimentés vont construire des solutions maison alors que cet API leur permettrait de faire ce qu’ils souhaitent très simplement).

À mon goût cela devrait être beaucoup plus utilisé (même pour déclarer et identifier une URL dans un code JavaScript). L’évolution et la maintenance des codes n’en seront que bien plus simple.



Le constructeur s’occupera de valider l’URL pour vous (une erreur sera throw s’il l’URL n’est pas valide). Vous pourrez ensuite récupérer toutes les informations que vous souhaitez sur votre URL (protocol, username, password, hostname, port, pathname, query etc).

URL peut être aussi utilisé en ESM avec import.meta.url pour remplacer \_\_dirname.



Bonus: [How to migrate from querystring to URLSearchParams in Node.js](https://www.linkedin.com/pulse/how-migrate-from-querystring-urlsearchparams-nodejs-vladim%25C3%25ADr-gorej/?trackingId=OEdtd%2BKZRxWNkgptsoWKlA%3D%3D)

### 

### OS

Le module core [OS](https://nodejs.org/api/os.html) est composé d’un ensemble d’utilitaires qui va vous permettre d'interagir et récupérer des informations sur votre système. Comme par exemple :

* Les interfaces réseaux sur votre machine.
* Des informations sur vos CPU (load, endianness, model, speed).
* Des informations sur la mémoire de votre machine.
* Des informations sur votre système d’exploitation (version, répertoire temporaire, répertoire racine, etc..).

### Child process

Le module child process va nous permettre de créer et gérer des nouveaux processus Node.js. Que ce soit pour simplement exécuter des commandes sur le système ou bien profiter d’avoir plusieurs CPU virtuels et exploiter les ressources de votre machine de manière plus optimale (pour par exemple gérer des tâches annexes). A ce titre les [Worker threads](https://nodejs.org/api/worker_threads.html) peuvent aussi servir.

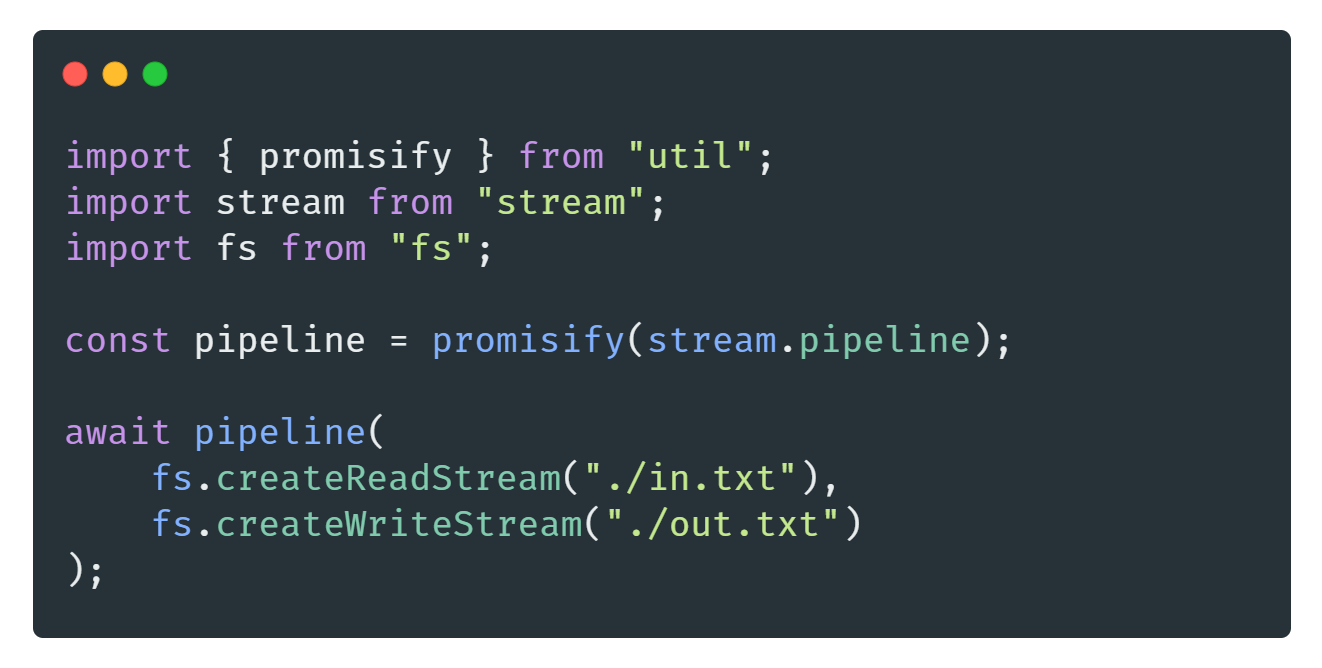


Si vous êtes à la recherche d’un module simplifiant ces tâches, je vous recommande de jeter un œil à [execa](https://github.com/sindresorhus/execa#readme). (il y aussi [cross-spawn](https://github.com/moxystudio/node-cross-spawn)).

| ⛔ Attention à ne pas exécuter des commandes à partir de chaîne de caractères venant de l’extérieur sans effectuer de vérification stricte (un attaquant pourrait s’en servir pour exécuter du code sur votre système). |
| --- |

### Utilities

Le module util contient différentes petites functions utilitaires. Nous l’utilisons le plus souvent pour récupérer promisify qui nous permettent de convertir un callback en Promise.



Il existe aussi une function pour passer d’une Promise à un callback ([callbackify](https://nodejs.org/api/util.html#util_util_callbackify_original)). Néanmoins je ne l'ai jamais utilisée^^..

La function [deprecate()](https://nodejs.org/api/util.html#util_util_deprecate_fn_msg_code) peut être utile si vous voulez par exemple rendre deprecated une méthode d’un package ou d’un projet. Cela permet d’avertir les développeurs à l’avance d’une prochaine mise à jour SemVer major qui supprimera la dit fonctionnalité.

La méthode [inspect()](https://nodejs.org/api/util.html#util_util_inspect_object_showhidden_depth_colors) peut-être utile si vous avez besoin de logs des objets d’une manière spécifique (avec différents niveaux de profondeur etc). C’est la méthode utilisée under the hood quand vous utilisez des méthodes console.

### 

### VM

Le module [VM](https://nodejs.org/api/vm.html) vous permet d’exécuter du code JavaScript au sein d’un contexte V8 différent et d’avoir notamment un [Realm](https://github.com/tc39/proposal-realms/blob/main/explainer.md) à part. C’est plutôt utile quand votre besoin est d’exécuter du code sans interférer avec le Realm en cours (vôtre processus Node.js).

Néanmoins le module core ne dispose pas d’énormément de fonctionnalités et ne garantit pas que vous pouvez exécuter du code qui vient de l’extérieur (quelqu’un de malveillant pourrait donc quand même s’échapper de la sandbox). J’utilise moi-même ce genre de fonctionnalités dans le cadre de SlimIO pour exécuter des codes fournis par des intégrateurs (dans le cadre d’une modularité par UI).

Si vous avez un besoin similaire je vous recommande jeter un oeil aux modules suivants :

* [Isolated VM](https://github.com/laverdet/isolated-vm)
* [VM2](https://github.com/patriksimek/vm2)

### Assert

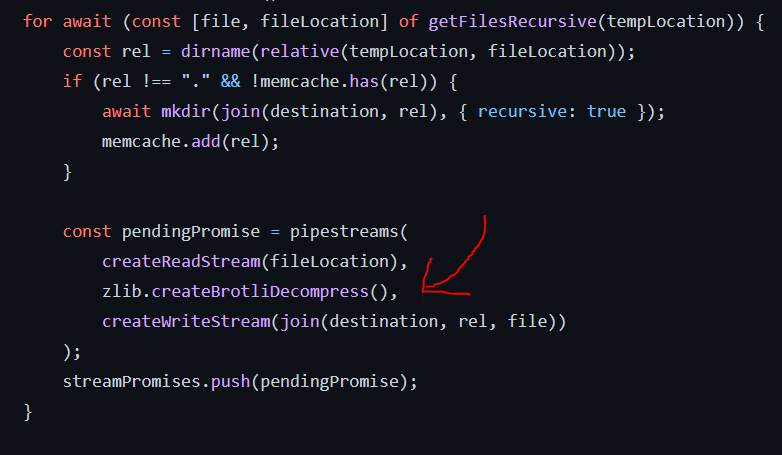
Un module assez simpliste contenant diverses fonctions d’assertion. Cela peut s’avérer utile si vous êtes du genre à faire des tests unitaires de manière assez simpliste (sans passer par de gros frameworks). Néanmoins cela peut très bien être utilisé pour des codes (c’est souvent utilisé dans des exercices ou des challenges par exemple).

### 

### 

### Zlib

Le module [zlib](https://nodejs.org/api/zlib.html) fournit des fonctionnalités de compression mises en œuvre à l'aide de Gzip, Deflate/Inflate et Brotli. Vous aurez à l’utiliser le plus souvent si vous manipulez des archives sur votre système.



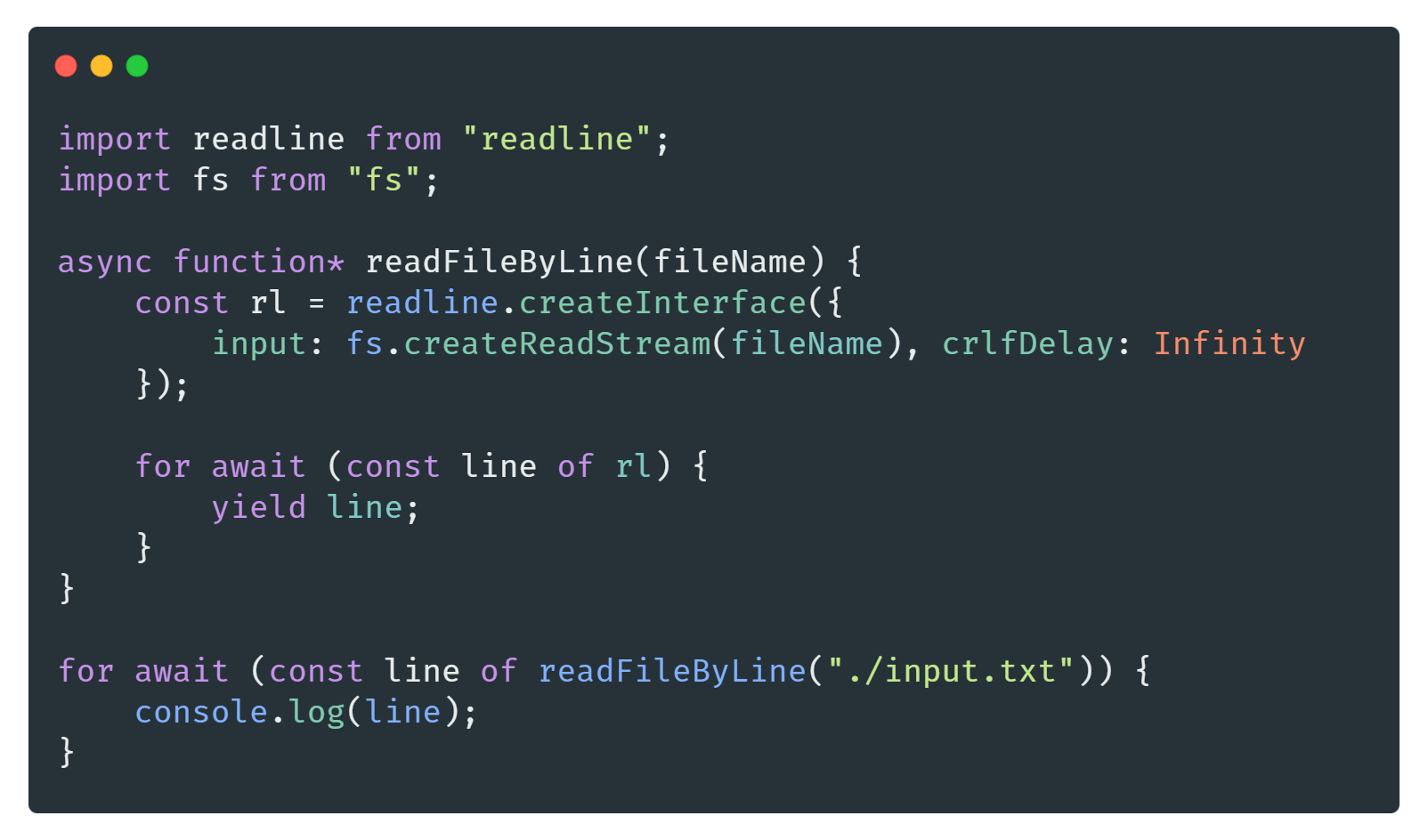
Si vous avez à gérer des archives en .tar ou .zip je vous recommande les packages suivants :

* [tar-fs](https://github.com/mafintosh/tar-fs)
* [Unzipper](https://github.com/SlimIO/unzipper)

### 

### Readline

Le module [readline](https://nodejs.org/api/readline.html#readline_readline) va permettre de consommer des Streams du type “Readable” ligne par ligne. Par exemple si vous souhaitez lire un fichier ligne par ligne :



Parce que process.stdin est un Stream Readable il est aussi possible de créer des CLI (ou tout simplement des prompts).



### perf\_hooks

Ce [module](https://nodejs.org/api/perf_hooks.html) fournit une implémentation d'un sous-ensemble des [API de performance Web du W3C](https://w3c.github.io/perf-timing-primer/) ainsi que des API supplémentaires pour les mesures de performance spécifiques à Node.js.

C’est un module vraiment cool qui va vous permettre de mesurer bien plus précisément les performances de méthodes JavaScript. Il est d’ailleurs fortement recommandé d’utiliser [performance.now()](https://nodejs.org/api/perf_hooks.html#perf_hooks_performance_now) plutôt que Date.now().



Dans SlimIO [nous utilisons](https://github.com/SlimIO/Addon/blob/5c5963c36fca462452f04b724e6050664fb27efc/src/callback.class.js) le module en combinaison avec des hooks Asynchrone pour nous permettre de générer des métriques sur les temps d'exécution de nos callbacks (ce qui nous permet de faire du self monitoring).

Le module perf\_hooks vous permet aussi de faire du monitoring sur l’event-loop (voir la section monitoring en bas pour plus de ressources sur le sujet).

### 

### async\_hooks

Je ne suis pas encore un grand expert sur les hooks Asynchrone (et le module est encore expérimental). Néanmoins je vous recommande de jeter un œil tout particulièrement à [l'API AsyncLocalStorage](https://nodejs.org/api/async_hooks.html#async_hooks_class_asynclocalstorage) créé par Vladimir.

Articles et vidéos intéressantes sur le sujet:

* **EN** [Node.js 14 & AsyncLocalStorage: Share a context between asynchronous calls](https://blog.kuzzle.io/nodejs-14-asynclocalstorage-asynchronous-calls)
* **EN NEW !** [Making async context management faster, safer, and more stable by doing less - Stephen Belanger](https://www.youtube.com/watch?v=WIVO_CfQ5ek)
* **EN NEW !** [Fun with async hooks and stack traces - Zbyszek Tenerowicz | WFH Conf 2020](https://www.youtube.com/watch?v=M5SoPVMZz0I)

L’intégration est en cours sur différents frameworks comme Adonis.js :

* [feat: add AsyncHttpContext (wip)](https://github.com/adonisjs/http-server/pull/18)

### 

### Crypto

Le module [Crypto](https://nodejs.org/api/crypto.html) (ou plus récemment [Web crypto](https://nodejs.org/api/webcrypto.html) qui est la version avec API compatible Web) est un module qui va vous permettre d’effectuer un ensemble d’opérations en lien avec le monde de la cryptographie.

Je ne suis pas spécialement très à l’aise avec tout cela mais en tant que développeur nous avons tout de même des besoins assez fréquents comme générer une chaîne SHA256 ou ce genre de choses (pour la signature d’un fichier par exemple).



Évidemment je ne vous recommande pas de gérer un chiffrage de mot de passe vous même car c’est très souvent le meilleur moyen de faire des bêtises (utilisez plutôt une lib comme [Argon2](https://github.com/ranisalt/node-argon2)). Si vous voulez savoir pourquoi je vous recommande d’aller lire l’article [suivant](https://medium.com/analytics-vidhya/password-hashing-pbkdf2-scrypt-bcrypt-and-argon2-e25aaf41598e).

En ce qui me concerne j’utilise aussi souvent le module crypto pour générer des valeurs aléatoires :



## 

## **📦 Archive de packages npm**

J’ai moi même, il y a plusieurs mois de cela, [créer une awesome-list](https://github.com/fraxken/awesome-crafted-nodejs) de packages Node.js que je vous recommande fortement. Ce sont des packages qui répondent à des besoins précis bien souvent avec moins (ou pas du tout) de dépendances ou avec des performances bien plus optimales.

Ma liste comporte aussi des personnes à suivre sur github (ils publient tous plus ou moins souvent des packages très intéressants). N’hésitez pas à PR pour ajouter des packages qui entrent dans la description du repo.

Il y a aussi l’awesome-list de Sindresorhus accessible [ici](https://github.com/feross/awesome-mad-science).

Liste d’autres packages sur lesquels je garde un oeil :

* [undici (A HTTP/1.1 client, written from scratch for Node.js.)](https://undici.nodejs.org/#/)
* [Fluent JSON Schema](https://github.com/fastify/fluent-json-schema)
* [Tesseract.js (OCR)](https://github.com/naptha/tesseract.js)
* [BoxedNode - Ship a JS file with Node.js in a box](https://github.com/mongodb-js/boxednode)
* [iobuffer](https://github.com/image-js/iobuffer) (Read and write binary data in ArrayBuffers)
* [Node re2](https://github.com/uhop/node-re2) (Fast, safe alternative to backtracking regular expression engines)
* [v8 compile cache](https://github.com/zertosh/v8-compile-cache) (Require hook for automatic V8 compile cache persistence)
* [Parsr](https://github.com/axa-group/Parsr) (Transforms PDF, Documents and Images into Structured Data)
* [cuid](https://github.com/ericelliott/cuid) (Collision-resistant ids optimized for horizontal scaling and performance)
* [dayjs](https://github.com/iamkun/dayjs) (2KB immutable date-time library alternative to Moment.js)
* [ncc](https://github.com/vercel/ncc) (Compile a Node.js project into a single file).
* [Private npm Registry](https://github.com/verdaccio/verdaccio)
* [Node watch](https://github.com/yuanchuan/node-watch#readme) (A wrapper and enhancements for fs.watch)
* [ms](https://github.com/vercel/ms#readme) (Tiny millisecond conversion utility)

| 📌 Sentez-vous libre de me faire des propositions. |
| --- |

# 

# 💾 Database

## Introduction

Les bases de données sont des composants centraux du parcours d’un développeur back-end. Elles permettent de stocker et gérer de la donnée en plus ou moins grande quantité pour répondre à des besoins et problèmes variés.

Il est assez complexe de les définir simplement tellement elles sont composées de plus de 50 ans d'ingénierie et d’innovation pour toujours rendre nos vies plus simples (et évidemment répondre à toujours plus de scale). Les développeurs ont d’ailleurs souvent la mauvaise habitude de les sous-estimer mais elles sont souvent bien plus efficaces que nos languages back-end dans bon nombre de tâches.

Chacune d’entre elles auront des forces et des faiblesses (que ce soit dans l’accessibilité des drivers, dans la gestion d’une donnée persistante ou volatile, dans la façon de structurer la donnée, de la filtrer etc). Elles forment des grandes familles tout comme les paradigmes en programmation (Relationnelle, In-memory, Graph etc).

## Influence de Node.js sur le chapitre

Ce chapitre sera fortement influencé par le fait que nous soyons en Node.js et mon objectif n’est pas d'aborder toutes les solutions de bases de données au monde, mais de tout simplement vous guider vers l’essentiel.

Il est important de comprendre que certaines bases de données n’ont aucun connecteur officiel en Node.js et cela sous-entend donc qu’il est laissé à la communauté le devoir de maintenir des bindings bas niveau qui nécessitent souvent de très grandes compétences techniques.

Les bases de données qui seront citées dans les chapitres suivants sont pour moi celles qui ont le plus de support ainsi que les qualités nécessaires pour être utilisées dans des contextes professionnels durables.

Cela ne sous-entend à aucun moment qu’il n’est pas possible d’utiliser une autre base (soyons clair).

## 

## L’importance d’apprendre SQL

Avec la mode (ou le buzzword ?) des bases de données NoSQL ainsi que des abstractions de plus en plus fréquentes utilisant GraphQL ([hasura](https://hasura.io/) par ex) il n’est pas rare de rencontrer des développeurs n’ayant jamais écrit de requêtes SQL.

Néanmoins SQL est et restera toujours un grand classique qu’il est très important d’apprendre. Il n’est d’ailleurs pas rare que SQL tout comme les RegEx soient utilisés bien au-delà du domaine même de la programmation (elles sont littéralement partout).

Bien souvent d’ailleurs même les techniciens dans l’IT ont des notions en SQL.

À titre personnel, j’ai commencé à écrire des requêtes SQL vers l’âge de 13 ans et j’ai continué à en faire activement au travers des années. La plupart du temps il n’est pas nécessaire de devenir un dieu vivant sur le sujet, mais de simplement savoir faire la base (SELECT, INSERT, UPDATE, DELETE) et de savoir a minima quelques notions comme les différentes possibilités de jointure.

Je vous recommande le site [sql.sh](https://sql.sh/) pour apprendre les diverses notions en lien avec SQL (Expérimenter les différentes fonctionnalités directement avec une base de données). Vous pourrez certainement apprendre le reste sur le tas pendant votre projet / mission :)

Prenez aussi le temps de voir si la base de données que vous utilisez ne propose pas des guides sur les spécificités et les optimisations possibles.

## 

## PostgreSQL

Dans l’écosystème Node.js Postgre est devenu pour moi un incontournable en termes de base open-source SQL.

Nous avons la chance d’avoir des drivers activement maintenus par la communauté comme [pg-promise](https://github.com/vitaly-t/pg-promise) ou [postgres](https://www.npmjs.com/package/postgres). Je n'ai jamais eu l’occasion de tester encore mais il y a aussi:

* [slonik](https://github.com/gajus/slonik)
* **NEW !** [massive.js](https://massivejs.org/).

Le projet lui-même est aussi très activement maintenu et suivi. C’est vraiment un choix idéal pour les projets aussi bien personnels que professionnels. Ces deux dernières années, cela se ressent d’ailleurs le nombre d’offres précisant PostgreSQL en base de données (en ce qui me concerne c’était la base de données en place sur l’intégralité de mes dernières missions).

Bref je vous incite donc fortement à vous intéresser à cette base de données avec la [page suivante](https://www.postgresql.org/about/) qui présente le projet et les avantages de celui-ci.

### **NEW !** Vidéo d’introduction

[Learn PostgreSQL Tutorial - Full Course for Beginners](https://www.youtube.com/watch?v=qw--VYLpxG4)

## 

## MySQL et MariaDB

MySQL et MariaDB sont des bases de données assez populaires et qui ont jusqu’à maintenant été énormément utilisées dans des projets Node.js car une multitude de drivers ont été créés par des développeurs comme [mscdex](https://github.com/mscdex).

Néanmoins je remarque avec les années que les développeurs qui étaient en charge du développement et de la maintenance de ces drivers ne sont plus autant actifs (voir complètement inactif) ce qui laisse les différents drivers avec “moins” de support. Je ne parle même pas du driver officiel MySQL qui a certainement été développé par un stagiaire.

Bref les implémentations et les mises à jour importantes prennent plusieurs années à être implémentées sur les drivers (je pense notamment à toutes les nouveautés avec MySQL 8.. les nouveaux standard d’auth sécurisé etc).

Les drivers disponibles et **actif** :

* [node-mysql2](https://github.com/sidorares/node-mysql2)
* [MariaDB Node.js connector](https://github.com/mariadb-corporation/mariadb-connector-nodejs) (semble être plutôt bien maintenue pour le moment).

| MariaDB est largement moins impactée par mon constat. Donc si vous avez le choix ne prenez pas MySQL. |
| --- |

## 

## MongoDB

La fameuse base de données... celle qui génère joie, pleurs et colère dans l’écosystème. Soyons clair: **MongoDB est la base la plus populaire de l’écosystème**.

Il n’y a rien à dire sur le support du [driver natif](https://github.com/mongodb/node-mongodb-native)… ce qui en fait pour moi une des bases de données les plus solides en termes de support. L’entreprise MongoDB possède d’ailleurs des développeurs Node.js très talentueux comme [Addaleax](https://github.com/addaleax) ce qui renforce tout de même le sentiment de support et de qualité vis à vis des outils que l’éditeur propose à l’écosystème.

J’ai fait mes débuts en Node.js et j’ai comme beaucoup suivi la danse en découvrant MongoDB. **Ce fut une excellente expérience pour moi** qui n’avait fait que du SQL et je dois dire que j’ai toujours plutôt apprécié faire du Mongo (cela m'a ouvert à d’autres horizons et façons de penser).

Néanmoins, j’ai moi-même pris dès le début le temps de bien lire la documentation car j’avais compris **qu’il me fallait arrêter de réfléchir en relationnelle** (sans pour autant trop savoir ce que cela voulait vraiment dire au début). Ma mission chez BNP Paribas sur la construction d’une PaaS m’a permis de pousser ma maîtrise notamment sur la gestion des données shard (avec toutes les contraintes que tu manges sur le coin du nez).

D’ailleurs je n'ai jamais compris le délire derrière [Mongoose](https://mongoosejs.com/) et l’idée de construire du relationnelle sur une base qui n’est pas relationnelle…

<https://twitter.com/matteocollina/status/894488535595614208>



Après quelques années, je peux maintenant vous dire que je comprends très bien pourquoi beaucoup de développeurs et architectes râlent sur le fait de choisir MongoDB pour des projets où une base de données relationnelles aurait très largement fait le boulot.

Il est vrai que beaucoup commencent des projets sur MongoDB alors qu’une base comme PostgreSQL aurait été un bien meilleur choix à tous les niveaux. Beaucoup construisent par conséquent des usines à gaz ou ils essayent de faire de Mongo une base relationnelle 😨.

Je pense qu'il ne faut pas être dans les extrêmes. Si vous êtes un développeur junior je vous recommande d’expérimenter le plus possible avec différentes bases de données.

## 

## Redis

Redis est une base de données très populaire dans l’écosystème (et populaire généralement pour les mêmes raisons dans les autres langages aussi). Elle stocke la donnée sous forme de clé/valeur directement en mémoire pour permettre des performances d’accès très rapide. Néanmoins la donnée est forcément volatile et en cas de problèmes (perte de courant ou autres incidents) elle sera définitivement perdue.

C’est pour cela que nous l’utiliserons plutôt dans les situations suivantes :

* Partager des informations entre plusieurs instances Node.js (sessions, state divers et variés, messages temporaires etc).
* Échanger des évènements avec d’autres instances (ce qu'on appellera pub-sub.. nous en parlerons plus en détails dans les prochains chapitres).

Pour vous fournir un exemple, il serait intéressant d’utiliser Redis pour la construction d’un tchat temps réel dès lors que vous souhaitez répartir vos utilisateurs sur plusieurs instances Node.js. Redis vous servira à synchroniser les messages et les événements au travers des différentes instances en ligne.

Pour en savoir plus, je vous recommande un article français de Nicolas Brodin: [Découvrez Redis, une base de données rapide comme l'éclair !](https://blog.nicolas.brondin-bernard.com/redis-une-base-de-donnees-rapide-comme-l-eclair/)

Drivers dans l’écosystème :

* [IO Redis](https://github.com/luin/ioredis) (**je vous recommande d’utiliser celui-ci**).
* [Redis](https://github.com/NodeRedis/node-redis)

Articles ou talks:

* **EN** [Solving Head-Of-Line blocking with auto pipelining, NearForm](https://www.youtube.com/watch?v=0L0ER4pZbX4)
* **EN NEW !** [What is Distributed Caching? Explained with Redis!](https://www.youtube.com/watch?v=U3RkDLtS7uY)

Pour plus de talks autour de Redis n’hésitez pas à consulter la chaine youtube [Redis Labs](https://www.youtube.com/c/Redislabs/featured).

## 

## SQLite

Malgré plusieurs années à rechercher une solution bien plus optimale que [SQLite](https://www.sqlite.org/index.html) pour faire du stockage très “simplement” en local ou pour de l’embarqué (pour [SlimIO](https://github.com/SlimIO/events) notamment) et bien je peux vous dire que je n'ai pas encore trouvé mieux.

SQLite c’est vraiment un projet solide et mature qui fonctionne très bien. Pour SlimIO j’ai fait l’effort de lire l'intégralité de la [documentation officielle](https://www.sqlite.org/docs.html) et ce fut une énorme surprise pour moi… Tant de choses que j’ignorais sur le fonctionnement et les optimisations possibles (comme quoi.. toujours lire la doc!).

Drivers disponible dans l’écosystème :

* [sqlite3](https://github.com/mapbox/node-sqlite3)
* [better-sqlite3](https://github.com/JoshuaWise/better-sqlite3) (attention complètement synchrone).
* **NEW !** [esqlite](https://github.com/mscdex/esqlite) (driver en dev très récent par mscdex).

## Autres bases de données

Une liste d’autres solutions que vous pouvez explorer :

* [LevelDB](https://github.com/Level/level) - fast key-value storage
* [Neo4j](https://neo4j.com/developer/javascript/)
* [RethinkDB](https://rethinkdb.com/)
* Firebase

N’hésitez pas non plus à consulter [l’awesome liste suivante](https://github.com/agarcialeon/awesome-database) qui contient bon nombre de solutions de bases de données.

## 

## ORM ou non ?

C’est une question qui revient souvent: doit-on utiliser un ORM ou non ? **En ce qui me concerne** j’essaye de m’en passer le plus possible lorsque mon besoin reste relativement simple (Néanmoins ce n’est pas tout le temps possible).

Ce qui est certain c’est qu’il faut arrêter d’utiliser [Sequelize](https://sequelize.org/) à tort et à travers (pour avoir fait plusieurs projets avec je peux vous garantir que c’est une véritable horreur).

Je vous recommande plutôt de jeter un oeil aux projets suivants :

* [TypeORM](https://typeorm.io/#/) (TypeORM supports both Active Record and Data Mapper patterns)
* [MikroORM](https://mikro-orm.io/) (TypeScript ORM for Node.js based on Data Mapper, Unit of Work and Identity Map patterns).

A noter que si vous utilisez le framework Adonis.js ils ont aussi leur propre ORM [Lucid](https://adonisjs.com/docs/4.0/lucid).

Il y a tout un tas de packages disponibles dans l’écosystème, mais je ne suis pas forcément convaincu par la plupart d’entre eux (Après il y en a sûrement que je n’ai moi-même pas encore découvert).

# 

# 📡 MQTT (broker)

## Introduction

MQTT (Message Queuing Telemetry Transport) est un protocole de messagerie [publish-subscribe](https://fr.wikipedia.org/wiki/Publish-subscribe) basé sur le protocole TCP/IP ([Wikipédia](https://fr.wikipedia.org/wiki/MQTT)).



MQTT est plutôt populaire pour la création de composants (logiciels) qu'on appellera des [brokers de messages](https://en.wikipedia.org/wiki/Message_broker) (comme une Queue ou plusieurs acteurs pourront écrire et consommer des messages). L’idée c’est bien souvent d’éviter un coupable trop fort entre différents composants d’une architecture.

À noter que nous parlons ici souvent du protocole MQTT mais il y a beaucoup de solutions qui sont bien plus bas niveau et qui préféreront conserver leurs propres implémentations en TCP/IP pour maximiser les performances.

## 

## Redis

Redis est bien souvent la solution utilisée par beaucoup d’entreprises pour effectuer du pub-sub entre plusieurs instances de Node.js. C’est vraiment très simple à prendre en main (et même chose pour la mise en place côté Infrastructure). Les performances sont très largement respectables (après tout dépend évidemment du besoin.. mais pour le commun des mortels ce n’est pas mal).

De plus, la base de données peut être utilisée pour d’autres besoins ce qui permet souvent d’économiser le besoin de mettre en place différentes solutions. Néanmoins je ne pense pas que Redis soit une solution optimale si votre besoin s’oriente plutôt vers une Queue avec pub-sub (pour un Bus de messages par exemple).

## RabbitMQ, ZeroMQ

Lorsque votre besoin s’oriente sur une Queue avec la possibilité de pub-sub entre plusieurs targets il devient intéressant de jeter un oeil aux différentes solutions les plus couramments utilisées en entreprise :

* [ZeroMQ](https://zeromq.org/)
* [RabbitMQ](https://www.rabbitmq.com/)

J’ai personnellement un peu d’expérience avec ZeroMQ dans le cadre de mon projet SlimIO car nous gérons un bus de messages entre plusieurs modules sur les agents centraux (cela permet notamment de scale la capacité de traitement du produit assez monstrueusement).

Néanmoins dans toute mon expérience professionnelle je n’ai que rarement vécu une production avec un réel besoin en matière de message broker. Il faut réellement rentrer dans des besoins de charges de classe mondiale.

## Autres

TBC

# 

# 🔐 Sécurité

## Introduction

Il est important de rester conscient des faiblesses du runtime Node.js (ainsi que les éléments composant Node.js: [V8](https://v8.dev/), [Libuv](https://github.com/libuv/libuv), HTTP etc…) Cela vous permettra de mieux comprendre et identifier les risques potentiels au sein de vos projets.

Les risques d'attaques ne doivent pas être sous-estimés ou banalisés. Dans le cas de "services Web" exposés (IP publique), une vigilance accrue doit être exercée à tout moment. Une erreur de votre part peut entraîner des dommages et des pertes considérables pour votre client (ou votre projet).

L'utilisation de dépendances tierces, en particulier des "packages" ou des portions de code provenant de l'univers open source doit notamment faire l'objet d'une surveillance importante.

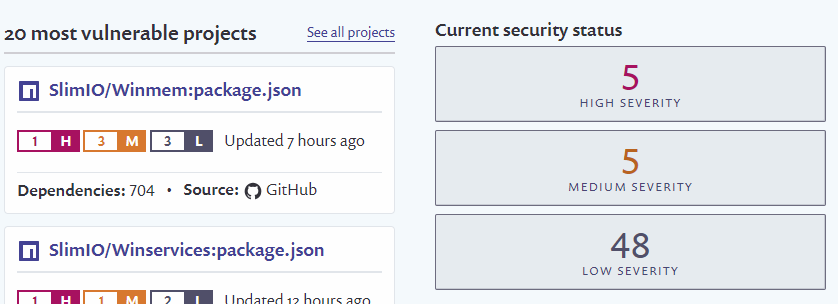
## 

## Audit

Il devient obligatoire de mettre en place des audits de sécurité automatique sur vos dépendances afin d'avoir une vision même minimale des vulnérabilités "connues" au sein de chaque projet. Bien sûr, ce n'est que le premier pas vers une sécurité optimale!

Des outils tels que [Snyk](https://snyk.io/), [sqreen](https://www.sqreen.com/), [npm audit](https://docs.npmjs.com/auditing-package-dependencies-for-security-vulnerabilities) vous permettent d'avoir une vue d'ensemble des CVE actifs ou des attaques possibles sur vos projets (l’offre gratuite de snyk est d’ailleurs plutôt généreuse).

*Exemple ci-dessous d’un tableau bord snyk sur le projet SlimIO*



Les vulnérabilités détectées pendant la phase de **runtime** sont bien **évidemment prioritaires (puisqu’elles concernent les codes exécutés en production)**.

| ⚠️ Cela ne veut pas pour autant dire que les vulnérabilités détectées sur les dépendances de développement ne sont pas importantes ! |
| --- |

Vous pouvez ouvrir l'un des projets pour avoir plus d'informations sur les différentes issues en cours. L'interface vous donnera tout un ensemble d'informations telles que des références et des explications sur le problème. Et, le cas échéant, des mesures correctives pour résoudre le problème.

# 

## 

## Garder sous contrôle votre Environnement

Il est impératif de ne pas publier sur Internet (git) un token, un login et/ou un mot de passe. Cela sous-entend qu’il faut absolument éviter de stocker en dur ces informations dans votre code source (quelquefois nous le faisons juste temporairement pour un test et nous ne faisons pas attention au moment de commit…).

Ces informations doivent être stockées dans l'environnement local (elles peuvent être gérées et chargées par projet en utilisant le fichier .env et le paquet npm [dotenv](https://www.npmjs.com/package/dotenv)). Les frameworks high-scope ont d'ailleurs bien souvent une gestion interne déjà opérationnelle (penser à vous documenter).

Dans le cadre de npm, nous vous conseillons d'utiliser [le guide suivant](https://snyk.io/blog/ten-npm-security-best-practices/) qui va vous permettre d’éviter différents vecteurs d’attaques.

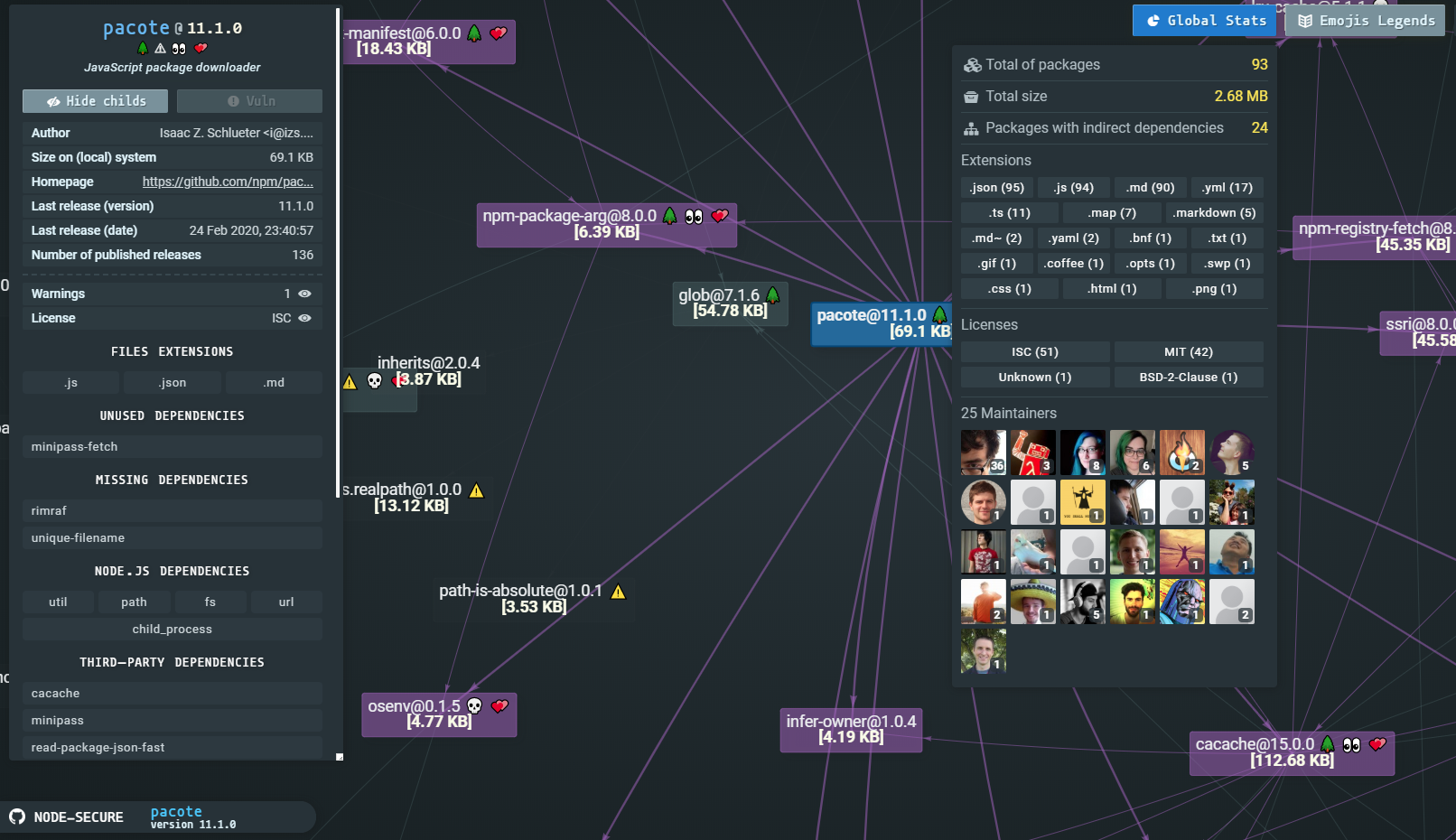
## 

## Dépendances directes et indirectes

Les projets possédant des dépendances indirectes sont bien plus vulnérables comme [le montre cette récente étude](https://arxiv.org/pdf/1902.09217.pdf) ou [cet article](https://snyk.io/blog/78-of-vulnerabilities-are-found-in-indirect-dependencies-making-remediation-complex/) de snyk. Le grand problème est qu’il devient difficile de simplement corriger une faille de sécurité rapidement puisqu’elle n’est pas directement liée à la dépendance que vous utilisez.

Évitez à tout prix les packages avec des dépendances dépassant une profondeur de 2-3 packages (cela introduit toujours plusieurs dépendances indirectes à maintenir et sécuriser ce qui peut très vite devenir complexe).

Le projet [nsecure](https://github.com/ES-Community/nsecure) permet d’analyser en profondeur les dépendances d’un projet ou d’un package npm.



| ⛔ Attention à ne pas non plus tomber **dans la parano**. Il n’est pas non plus, tout le temps simple, de résoudre des problématiques en quelques packages. L’important est d’être conscient du problème et de faire attention. |
| --- |

## 

## Faille de sécurité courante

Il est important de comprendre et se former sur les attaques les plus courantes en Node.js pour mieux les prévenir. Les attaques suivantes sont couramment employées par les pirates dans l’écosystème :

* [ReDos](https://medium.com/@liran.tal/node-js-pitfalls-how-a-regex-can-bring-your-system-down-cbf1dc6c4e02)
* [Directory Traversal](https://snyk.io/vuln/SNYK-JAVA-IOUNDERTOW-32074)
* Brute forces (si aucune limite ou vérification.. peut être couplé avec une attaque par dictionnaire si votre système indique si un mail est utilisé ou non).
* [Injection SQL](https://snyk.io/learn/sql-injection/) *(*[*aussi possible avec MongoDB*](https://blog.sqreen.com/mongodb-will-not-prevent-nosql-injections-in-your-node-js-app/) *etc.. ne vous croyez pas à l'abri et creuser le sujet).*

En bonus je vous recommande d’aller consulter [le top 10 des attaques répertoriés par OWASP](https://owasp.org/www-project-top-ten/).

## Analyse statique & AST

Dans le cadre de mon projet [Node-secure](https://github.com/ES-Community/nsecure), un de mes objectifs était de réussir à détecter des codes “malicieux” divers et variés qui ont été répertoriés à travers diverses attaques (plusieurs peuvent être retrouvés sur [badjs](https://badjs.org/)).

Quand nous parlons d’analyse statique, il s’agit donc d’analyser un code source sans jamais l’exécuter (sinon c’est plutôt une analyse dynamique). Pour cela nous allons transformer le code en format AST ce qui va rendre l’analyse du code bien plus simple.

J’ai récemment fait [un talk en français](https://www.youtube.com/watch?v=zSYrEbggqWA&feature=youtu.be) sur la manipulation d’AST en JavaScript que je vous recommande de visionner. C’est avec ces techniques que j’ai réussi à créer mon projet [JS-X-Ray](https://github.com/fraxken/js-x-ray) (qui est un outil SAST -> Static Application Security Testing).

Autres talks en lien avec les Abstract Syntax Trees :

* **EN** [Machine Powered Refactoring: Leverage AST's to Push Your Legacy Code (& the Web) Forward](https://www.youtube.com/watch?v=s8g_cBfm9d0&list=PLfMzBWSH11xaZvhv1X5Fq1H-oMdnAtG6k&index=2)
* **EN** [What the AST ?](https://www.youtube.com/watch?v=BtD2OrlLBhI&list=PLyspMSh4XhLP-mqulUMcaqTbLo-ZJxSX5&index=36)

## 

## Liens et ressources complémentaires à lire

Le repo git [Awesome Node.js security](https://github.com/lirantal/awesome-nodejs-security) est une pépite à ne surtout pas rater (il référence énormément de projets, articles et autres documents pour vous permettre de vous éveiller sur la sécurité en Node.js).

Je vous recommande aussi [Node.js Security Handbook](https://www.sqreen.com/resources/nodejs-security-handbook) par la société Sqreen pour apprendre à mieux sécuriser vos applications Node.js.

Autres ressources (articles, talks, études):

* **EN** [The State of Open Source Security 2020 - Snyk](https://snyk.io/opensourcesecurity-2019/)
* **EN** [The State of Open Source Security - Liran Tal, Snyk](https://www.youtube.com/watch?v=0-wuXhapMbE&list=PLfMzBWSH11xZPfWcC0DqFqKo_reMP58mw&index=23)
* **EN** [We’re under attack! 23+ Node.js security best practices](https://medium.com/@nodepractices/were-under-attack-23-node-js-security-best-practices-e33c146cb87d)
* **EN** [Top 11 Node.js security best practices - Sqreen](https://blog.sqreen.com/nodejs-security-best-practices/)
* **EN** [Snyking in – Directory traversal vulnerability exploit in the st package](https://snyk.io/blog/snyking-in-directory-traversal-vulnerability-exploit-in-the-st-package/)
* **EN** [Hacker-Powered Data: The Most Common Security Weaknesses and How to Avoid Them - David Horvath](https://www.youtube.com/watch?v=D5qeeeDADQQ&list=PLfMzBWSH11xZPfWcC0DqFqKo_reMP58mw&index=46)
* **EN** [Securing Node.js and JavaScript - VLADIMIR DE TURKHEIM](https://www.youtube.com/watch?v=LJqw6oCW_YQ)
* **EN** [NodeConf Remote 2020 - Matteo Collina - Why there are no incentives for security in Open Source](https://www.youtube.com/watch?v=l7EbDZRjwXA&list=PL0CdgOSSGlBalMPxFFycq7OIqQF8cJS28&index=12)
* **EN NEW !** [Trojan Source - Invisible Source Code Vulnerabilities](https://www.trojansource.codes/)

Si vous êtes à la recherche d’une formation je vous recommande celle-ci faite par [Vladimir](https://twitter.com/poledesfetes) (collaborateur Node.js français): [Writing Secure Code in Node.js](https://www.infosecinstitute.com/skills/learning-paths/writing-secure-code-in-node-js/)

## Si la sécurité est un domaine qui vous intéressent

Si certains d’entre vous veulent découvrir et s’intéresser au monde de la sécurité par eux même je vous recommande fortement les liens suivants :

* **EN** [Awesome security](https://github.com/sbilly/awesome-security)
* [Rootme (Une plateforme rapide, accessible et réaliste pour tester vos compétences en hacking)](https://www.root-me.org/)
* Participation à des évènements comme [Insomni'Hack](https://www.insomnihack.ch/) (PS: plusieurs membres de la communauté JavaScript y participent souvent).

# 

# GIT

GIT est sans aucun doute l’un des outils les plus indispensables pour un développeur (que ce soit en Node.js ou autres). Si vous ne savez pas ce qu’est GIT, voilà la page [Wikipédia](https://fr.wikipedia.org/wiki/Git).

Si vous êtes **un débutant complet 🐤** alors je vous recommande [git-it](https://github.com/jlord/git-it-electron) qui est un guide assez bien foutu avec une application electron. Prenez d’ailleurs très vite l’habitude d’enregistrer votre travail sur github pour pouvoir vous entraîner le plus possible avec les commandes de base (Je vous recommande aussi de suivre la [convention suivante](https://www.conventionalcommits.org/en/v1.0.0-beta.4/) pour la rédaction de vos messages de commit).

Il existe quelques ouvrages qui vont traverser tout un ensemble de notions vis-à-vis de GIT... Je vais être honnête je n’ai pas tout retenu moi-même mais la lecture de ces ebooks a été très bénéfique pour m’améliorer :

* **EN** [Pro GIT](https://git-scm.com/book/en/v2)
* **EN** [Mastering GIT](https://www.raywenderlich.com/books/mastering-git/v1.0) (**payant**).
* **EN** [Git Immersion](https://gitimmersion.com/)

Nous avons la chance en France d’avoir des articles, talks et formations de qualité grâce à Delicious insights (Christophe Porteneuve et ses collègues). Beaucoup du contenu ci-dessous **est gratuit,** mais si vous en avez les moyens, je vous recommande très fortement d’investir sur un cours :

* **FR** [Les concepts clés de Git](https://cours-video.delicious-insights.com/git-concepts)
* **FR** [Meilleures pratiques Git & GitHub par Christophe Porteneuve](https://www.youtube.com/watch?v=GH7wJ9voKK4)
* **FR** [Git pro tips: Maîtrisez Git comme un pro !](https://www.youtube.com/watch?v=OgVQDtEf0w8)
* **FR** [Bien utiliser Git merge et rebase](https://delicious-insights.com/fr/articles/bien-utiliser-git-merge-et-rebase/)
* **FR** [Git undo: ou le savoir-défaire](https://cours-video.delicious-insights.com/git-undo)
* **FR** [Git reset : rien ne se perd, tout se transforme](https://www.paris-web.fr/2019/conferences/git-reset-rien-ne-se-perd-tout-se-transforme.php)
* **EN** [Learn Git Branching](https://learngitbranching.js.org/?locale=fr_FR)

| 👀 Bien qu’il soit possible d’utiliser des interfaces graphiques, je vous recommande de prendre l’habitude d’en faire en ligne de commande pour ne pas vous contraindre / limiter dans le futur. |
| --- |

# 🌇 Architecture & Paradigmes

## SemVer

SemVer est le standard qui va nous permettre de gérer et incrémenter nos versions de logiciels, packages et tags dans l’écosystème Node.js (en tout cas c’est le plus fréquemment maîtrisé et employé).

Je ne vais pas vulgariser le sujet car cela est déjà fait [ici](https://semver.org/lang/fr/). (page qui devrait avoir été lue par tout développeur JavaScript/Node.js).

Autres ressources utiles :

* <https://semver.npmjs.com/>
* [semver npm package](https://github.com/npm/node-semver#readme)
* [semver npm doc](https://docs.npmjs.com/cli/v6/using-npm/semver)
* [Semver cheatsheet](https://devhints.io/semver)

## CalVer

J’entends de plus en plus parler de [Calendar Versioning](https://calver.org/). Bien que je ne connaisse pas beaucoup de projets utilisant cette spécification, je pense qu’il est tout de même intéressant de la garder à l'œil.

## 

## JSON Schema

Apprendre à écrire des [schémas JSON](https://json-schema.org/) est devenu pour moi un incontournable de l’écosystème JavaScript et Node.js ces derniers temps. De plus en plus d’outils et frameworks utilisent des schémas pour faire de la validation, création d’un JSON OpenAPI utilisable dans une interface Swagger etc.

L’exemple le plus parlant est certainement le framework Fastify.js. Néanmoins dans le cadre du projet SlimIO les schémas sont aussi omniprésent dans la gestion des configurations.

Le package [AJV](https://ajv.js.org/) est la solution incontournable si vous voulez expérimenter et travailler avec.



* [Get started with validation in Node.js](https://simonplend.com/get-started-with-validation-in-node-js/)

## 

## Documentation

Pour moi, savoir construire une documentation **est une des compétences les plus rares et précieuses chez les développeurs**. C’est d’ailleurs pour moi un point critique de non-embauche chez les candidats juniors! Venir avec des projets qui n’ont strictement aucune documentation (markdown) n’est pas acceptable selon moi.

Pour un projet sur github par exemple :

* Quels sont les prérequis du projet ? (version de Node.js, base de données utilisée etc).
* A quel besoin le projet répond-t-il ?
* Des diagrammes sont t’ils disponibles pour pouvoir mieux comprendre l’architecture et les choix fait dans le projet / code ?

Donc pensez à bien documenter vos projets personnels car je pense que cela à un impact très important aussi bien pour les personnes techniques et non techniques (et je ne parle pas de faire un copier-coller d’un getting started). Perso je n'ai pas besoin de voir le code d’un développeur qui construit des documentations rigoureusement…

Exemple de projets pour vous inspirer: [JS-X-Ray](https://github.com/fraxken/js-x-ray), [SlimIO Config](https://github.com/SlimIO/Config), [Flydrive](https://github.com/Slynova-Org/flydrive), [Hooks](https://github.com/poppinss/hooks).

Autres ressources, outils et packages pour construire des documentations :

* [JSDoc](https://jsdoc.app/) (voir aussi [ESDoc](https://esdoc.org/). A titre perso j’avais fait mon [propre fork](https://github.com/fraxken/minami) de minami).
* [TypeDoc](https://typedoc.org/) (pour TypeScript)
* [Github Pages](https://pages.github.com/)
* **EN** [What it takes to build great product documentation](https://blog.sqreen.com/great-product-documentation/)
* **EN** [Beyond the README: Creating Effective Documentation for Your Project by Rand McKinney, IBM](https://www.youtube.com/watch?v=NwUWuD9Idv4)

Si vous construisez une API: avoir un [OpenAPI](https://swagger.io/specification/) avec par exemple une interface [Swagger](https://swagger.io/tools/swagger-ui/) peut-être un gros plus (une collection postman peut aussi faire l’affaire).

## 

## Clean code

La notion de clean code a été introduite par le biais [du bouquin de Robert Martin](https://www.amazon.fr/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882). Néanmoins la plupart du temps le clean code c’est souvent un ensemble de conventions qui sont propres à chaque écosystème (avec des fondations parfois communes à plusieurs langages).

Par exemple nommer correctement ses variables **c’est plutôt du bon sens** (a titre personnel je faisais déjà du “clean code” avant de connaître la notion même de clean code).

Néanmoins il faut comprendre qu’il ne faut pas appliquer bêtement ces règles sans prendre en compte le contexte ou même l’avis de votre équipe. Il n’est maintenant pas rare de voir apparaître des outils comme SonarQube qui vont recommander de suivre des patterns dans le seul objectif de vous mettre une note "positive".

Autres liens que je vous recommande de lire :

* **EN** [Clean Code (Wiki C2)](https://wiki.c2.com/?CleanCode)
* **EN** [Effects of Clean Code on Understandability](https://www.duo.uio.no/bitstream/handle/10852/51127/master.pdf?sequence=1)
* **EN** [The Effect of Poor Source Code Lexicon and Readability on Developers’ Cognitive Load](http://veneraarnaoudova.ca/wp-content/uploads/2018/03/2018-ICPC-Effect-lexicon-cognitive-load.pdf)
* **EN** [It's probably time to stop recommending Clean Code](https://qntm.org/clean)
* **EN** [Clean Code JavaScript (github)](https://github.com/ryanmcdermott/clean-code-javascript)

Conférence assez récente;

* **EN** [Clean Code - Uncle Bob / Lesson 1](https://www.youtube.com/watch?v=7EmboKQH8lM&feature=emb_logo) (il y a plusieurs vidéos 1, 2, 3, 4, 5, …).

## 

## Programmation Orienté Objet

En ce qui me concerne, j’étudie sérieusement les racines et la sémantique des langages OO depuis deux ans et j’ai le sentiment d’être un alien 👽 quand j’entends les autres parler du sujet.

Pour commencer, il n’existe [aucun consensus sur une définition exacte de l’OO](https://wiki.c2.com/?NobodyAgreesOnWhatOoIs) (donc prenez avec des pincettes ceux qui vendent des définitions exactes 😞). Il est donc très risqué de prendre position car cela vous demandera d’avoir une compréhension et des arguments que 99,9% des développeurs n’ont pas (moi y compris pour le moment).

Ci-dessous deux liens qu’il est plutôt important de lire pour commencer à se faire une bonne idée des principes fondamentaux de l’OO :

* **EN** [Les définitions d’Alan Kays](https://wiki.c2.com/?AlanKaysDefinitionOfObjectOriented) (celui que beaucoup considèrent comme étant le père fondateur de l’OO avec Smalltalk).
* **EN** [A Proposal for Simplified, Modern Definitions of "Object" and "Object Oriented"](https://wcook.blogspot.com/2012/07/proposal-for-simplified-modern.html) (la définition que je trouve la plus sérieuse à l’heure actuelle et que j’ai découverte par le biais de Wirfs-brock).

Il existe bien d’autres définitions et spéculations que vous pouvez retrouver [ici](https://wiki.c2.com/?DefinitionsForOo). J’ai pour ma part aussi lu une bonne partie de [Theory of objects](http://lucacardelli.name/TheoryOfObjects.html) ce qui m’a permis d’ouvrir les yeux sur pas mal de notions à un niveau sémantique.

En second, je vous recommande vraiment de creuser la notion de Data abstraction. C’est tellement la fondation de ce qui constitue les langages de programmation que je ne comprends même pas comment la majorité des développeurs passent à côté .

Quelques ressources à lire / écouter :

* **EN** [How Data Abstraction changed Computing forever | Barbara Liskov | TEDxMIT](https://www.youtube.com/watch?v=_jTc1BTFdIo)
* **EN** [On Understanding Data Abstraction, Revisited](https://www.cs.utexas.edu/~wcook/Drafts/2009/essay.pdf)
* **EN** [Object-Oriented Programming Versus Abstract Data Types](https://www.cs.utexas.edu/~wcook/papers/OOPvsADT/CookOOPvsADT90.pdf)

Je vous recommande aussi [Programming Paradigms for Dummies](https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf) qui permet de faire un pont avec d’autres notions (et pas uniquement l’OO car c’est important de s’ouvrir aussi au reste).

### Ressources complémentaires

Bref si vous avez commencé à lire c’est que vous avez compris que je ne suis pas adepte des explications bateau en mode “**C’est un chat qui étend d’un animal pour faire miaou**” 😏.

Je n'ai jamais été un grand fan de ces articles haut niveau car j’ai toujours eu le sentiment qu’il manquait une réflexion fondamentale dans le processus. Abstraire les éléments techniques pour mieux les communiquer est une chose importante mais je pense qu’il ne faut pas non plus trop s’éloigner d’une réalité plus scientifique.

Si vous souhaitez continuer à réellement creuser la programmation orientée objet voici diverses références que j’ai moi-même lues (attention elles sont souvent difficile d’accès) :

* **EN** [Object-Oriented Programming in Scheme](https://mumble.net/~jar/pubs/oopis.pdf)
* **EN** [A Self Bibliography](https://bibliography.selflanguage.org/)

| 👀 Les références ci-dessus peuvent aussi vous permettre de mieux comprendre certaines racines de Javascript. J’étais personnellement vraiment choqué de découvrir que beaucoup de ce qu’on critique comme étant des fonctionnalités qui ont été faites au hasard en 10 jours **sont finalement des notions sémantiques réfléchies depuis des dizaines d’années**. |
| --- |

* **EN** [Inheritance Is Not Subtyping](https://www.cs.utexas.edu/~wcook/papers/InheritanceSubtyping90/CookPOPL90.pdf)
* **EN** [A Behavioral Notion of Subtyping](https://www.cs.cmu.edu/~wing/publications/LiskovWing94.pdf)
* **EN** [A Denotational Semantics of Inheritance](https://www.cs.utexas.edu/~wcook/papers/thesis/cook89.pdf)
* **EN** [Representing Type Information in Dynamically Typed Languages](https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.4394&rep=rep1&type=pdf)
* **EN** [Concurrent Object-Oriented Programming in Act 1](https://web.media.mit.edu/~lieber/Lieberary/OOP/Act-1/Concurrent-OOP-in-Act-1.html)
* **EN** [The Power of Interoperability: Why Objects Are Inevitable](http://www.cs.cmu.edu/~aldrich/papers/objects-essay.pdf)
* **EN NEW !** [Law of demeter](https://www2.ccs.neu.edu/research/demeter/papers/law-of-demeter/oopsla88-law-of-demeter.pdf)
* **EN NEW !** [Object Design - Rebecca Wirfs-Brock and Alan McKean](https://www.informit.com/promotions/object-design-142314?utm_source=referral&utm_medium=wrifsbrock&utm_campaign=objectdesign&utm_term=pdf)

À partir de tout cela vous aurez moyen de retrouver plusieurs centaines de références. J’ai lu d’autres ouvrages mais je ne pense pas qu’ils aient une place ici (néanmoins je reste ouvert à la discussion si vous souhaitez vous-même approfondir cela avec mes recommandations).

## Design patterns

Les développeurs de nos jours aiment suivre ces fameux “**design patterns**” pour découvrir diverses façons de régler des problèmes qu’ils rencontrent (seul ou en équipe). Beaucoup des frameworks Node.js sont d’ailleurs bien souvent construits à l’aide d’un ensemble de design patterns (Adapter, Factory, Singleton, Observer etc..).

Les design patterns sont souvent nécessaires pour mieux spécifier, identifier et discuter en équipe de divers problèmes et de comment les résoudre par le biais d’abstractions qui seront clairement identifiables par l’ensemble de l’équipe.

J’utilise et je construis moi-même très régulièrement plusieurs de ces patterns. Quelques liens et ressources pour découvrir et vous documentez sur les différents patterns designs existants :

* **EN** [Patterns for Beginners (Wiki C2)](https://wiki.c2.com/?PatternsForBeginners)
* **EN** [Dive Into Design Patterns](https://sourcemaking.com/design-patterns-ebook)
* **EN** [Node.js design patterns](https://www.nodejsdesignpatterns.com/)
* **FR** [Design Patterns - Tête la première](https://www.eyrolles.com/Informatique/Livre/design-patterns-tete-la-premiere-9782841773503/)
* **EN** [Design Patterns (Wiki C2)](https://wiki.c2.com/?DesignPatterns)

## 

## Compilateur et Interpréteur

En complément de la section V8 engine je pense qu’il est important d’ajouter une section qui parle de notions qui vont aussi un peu plus loin que ce qui est disponible en JavaScript.

Ce sont souvent des notions que beaucoup considèrent comme étant extrêmement avancées mais je pense qu’il faut surtout avoir un certain intérêt pour le sujet (ce n’est pas forcément non plus la mort à appréhender).

En premier lieu je vous recommande vraiment [la chaîne youtube](https://www.youtube.com/channel/UC15UOu9F157NAXGL9dHc4TQ) de Dmitry Soshnikov. Il y a aussi [son site internet](http://dmitrysoshnikov.com/) qui est bourré d’articles expliquant de divers sujets en lien avec les compilateurs modernes (une réelle mine d’or).

Mon github [contient une section dédiée](https://github.com/fraxken/VM-Resources#papers-and-academic-research) à différentes recherches académiques sur des optimisations modernes de compilateurs.

## 

## **NEW !** DDD (Domain Driven Design)

Les logiciels sont un moyen comme un autre d’arriver à une fin / d’atteindre un objectif réel. Les logiciels n’étant pas une fin en soi, ils doivent rester pratiques, utiles et ergonomiques.

Le Domain Driven Design fait référence à la conception pilotée par le métier. C’est une approche de développement de logiciels centrée sur le métier au travers de design patterns de conception (technique), des modèles conceptuels.

* **EN** [What is DDD - Eric Evans - DDD Europe 2019](https://www.youtube.com/watch?v=pMuiVlnGqjk)
* **FR** [Event Storming, une description](https://pablopernot.fr/2019/07/event-storming-description/)
* **EN** [Event Storming - Alberto Brandolini - DDD Europe 2019](https://www.youtube.com/watch?v=mLXQIYEwK24)
* **FR** [DDD, en vrai pour le développeur (Cyrille Martraire)](https://www.youtube.com/watch?v=h3DLKrvp5V8)
* **EN** [Alberto Brandolini - 100,000 Orange Stickies Later | Øredev 2019](https://www.youtube.com/watch?v=fGm62ra_mQ8)

More videos on the [Domain-Driven Design Europe youtube channel](https://www.youtube.com/channel/UC3PGn-hQdbtRiqxZK9XBGqQ).

## Autres

Il y a énormément de choses à apprendre qui ne sont pas forcément en lien avec le code en lui-même (car notre métier ce n’est pas juste pisser du code sans réfléchir).

[Le podcast de l’Artisan Développeur](https://artisandeveloppeur.fr/podcast/) fait intervenir beaucoup d’interlocuteurs autour de sujets métiers en lien avec les sections précédentes, donc je vous recommande d’aller écouter. **Il couvre aussi un tas de sujets que je n’aborde pas ici**.

Autres liens que j’apprécie :

* **EN** [AHA Programming](https://kentcdodds.com/blog/aha-programming)
* **EN** [Free Computer, Mathematics, Technical Books all over the World](https://freecomputerbooks.com/)
* **EN** [Computerphile (Youtube)](https://www.youtube.com/channel/UC9-y-6csu5WGm29I7JiwpnA/videos)
* **EN NEW !** [Being glue](https://noidea.dog/glue)
* **EN NEW !** [Radical Simplicity in Software](https://www.radicalsimpli.city/)

# 

# 🔬 Monitoring

| ⚠️ La section monitoring n’est pas encore très complète. Attendez-vous à une amélioration de plusieurs pages dans les mois à venir. |
| --- |

## Introduction

En tant que développeur il est important de savoir identifier les métriques et pratiques qui vont nous permettre de réfléchir et construire un monitoring pour nos applications (web ou desktop). **Nous parlerons surtout d’APM ici**.

Certains frameworks modernes comme fastify [ont déjà des packages](https://www.npmjs.com/package/fastify-metrics) pour exporter des métriques (format prometheus).

Les métriques les plus importantes en Node.js sont bien souvent celles qui concernent l’event-loop. Je vous recommande les ressources suivantes qui passeront à travers différents moyens de les exploiter ;

* **EN** [NodeConf Remote 2020 - Trevor Norris - Introduction to Event Loop Utilization](https://www.youtube.com/watch?v=WetXnEPraYM&list=PL0CdgOSSGlBalMPxFFycq7OIqQF8cJS28&index=13)
* **EN** [Introduction to Event Loop Utilization in Node.js](https://nodesource.com/blog/event-loop-utilization-nodejs/)

Autres ressources en lien avec le monitoring Node.js :

* **EN** [Don’t Starve the Event Loop: Measuring and Monitoring Node.js for Performance](https://www.youtube.com/watch?v=9RhOGoChGqo)
* **EN** [Node.js Monitoring, Alerting and Reliability 101 Ebook](https://risingstack.com/resources/node-js-monitoring-ebook)
* **EN** [Module core Node.js V8](https://nodejs.org/dist/latest-v15.x/docs/api/v8.html)

## API

Liens et ressources pour du monitoring d’API HTTP :

* [Health Check Response Format for HTTP APIs](https://tools.ietf.org/id/draft-inadarei-api-health-check-01.html)

# 💻 Système

## Introduction

Je vais sûrement en décevoir beaucoup, mais tout ce qui est en lien avec le système est très loin d’être ma tasse de thé. Non pas que le sujet ne m’intéresse pas, mais je préfère très largement me focaliser sur des tâches de développement pures (ce qui se ressent énormément dans mon spectre de compétences).

Il est néanmoins important de savoir au minimum se démerder (la plupart du temps c’est juste une question de bonne volonté). En France nous avons l'excellente chaîne youtube de [Xavki](https://www.youtube.com/c/xavki-linux/videos) qui parle d’énormément de sujets autours de divers OS, Docker, etc… Par ailleurs, en ce qui me concerne je ne suis pas particulièrement un grand fan de tout ce qui est en lien avec la guerre fréquente entre les OS… En tant que développeur de solution de monitoring, j’ai le devoir de mettre les pieds sur les différentes solutions existantes ce qui fait que je suis amené à vivre différentes choses (pour avoir déjà un peu d’expérience en binding bas niveau sur les différents OS, je peux vous garantir qu’il y a toujours des trucs bien et des problèmes sur chaque système).

| **⛔** Je le dis souvent dans ce document mais ne vous enfermez jamais dans une bulle idéologique. |
| --- |

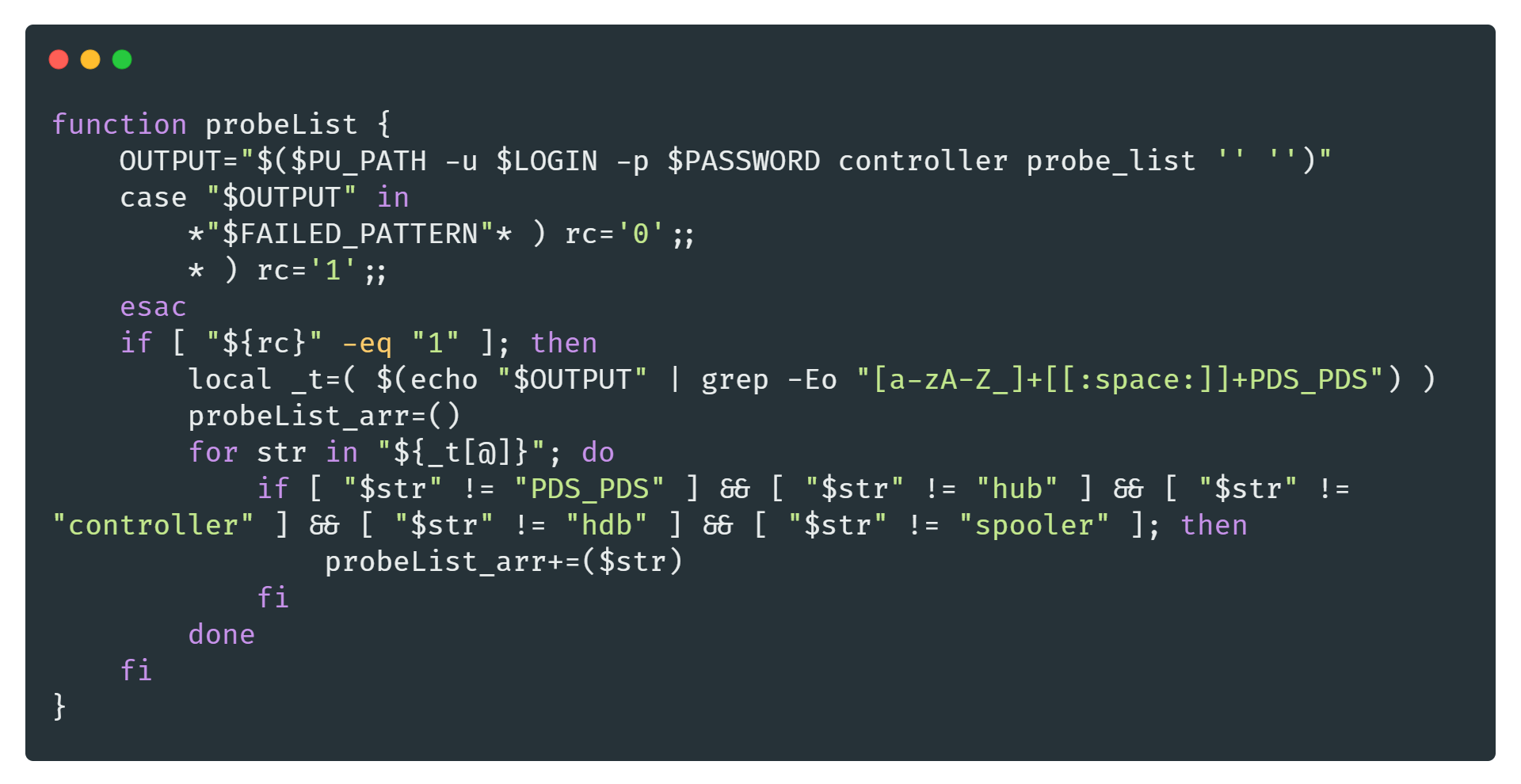
## 

## Scripting: Shell, Bash et Powershell

En ce qui me concerne, j’ai déjà pas mal d’expérience vis-à-vis de diverses notions de scripting système avec des langages comme Powershell. Dans le monde du monitoring c’est une maîtrise assez importante car nous sommes fréquemment confrontés à des situations où il est important de savoir bidouiller des scripts pour récolter des métriques, automatiser des tâches ou tout simplement modifier/configurer un produit X ou Y.

Plus je m’enfonce dans le monde du Software, plus je me rend compte qu’il est vraiment important de maîtriser cela (bien souvent pour écrire et maintenir des scripts d’installation ou de déploiement par exemple).

Cela me permet en tant que développeur de faire le pont avec les membres de mon équipe qui vont gérer la DevOps etc… ; tout en conservant ma passion et mon intérêt car nous restons dans le domaine du développement.



# 

# 💖 Remerciements

C’est le moment des adieux et des larmes de tristesse 😭 (ou de joie ? 😈). Je tiens à remercier toutes les personnes sans qui ce document n’aurait pas été possible (ou du moins pas avec le même niveau de qualité).

Un grand merci aux membres de la communauté [JavaScript et Node.js francophone](https://github.com/ES-Community/Code-of-conduct) sans qui je n’aurais jamais pu acquérir tant de connaissances et références. Quand bien même, je suis le fondateur du Discord, je ne peux qu’être honoré de votre participation quotidienne et ce maintenant depuis plus de cinq ans pour certains d’entre vous.

Merci à ceux qui m’ont conseillé et aidé dans l’écriture et la correction du document;

* [Tony Gorez](https://www.linkedin.com/in/tonygorez/) (sa chaîne youtube [tonygo](https://www.youtube.com/channel/UC0yiy-XPDRVAgLaAiA8kvrQ)).
* [Alexandre Malaj](https://www.linkedin.com/in/alexandre-malaj-6062b0a6/) (mon associé et collègue).
* [Nadia](https://www.linkedin.com/in/nadiamedkouri/)
* Koko (mentor ES-Community).
* [Vincent Dhennin](https://www.linkedin.com/in/vincentdhennin/).
* [Vladimir de Turckheim](https://twitter.com/poledesfetes)