Dev Web

Côté Serveur / Backend

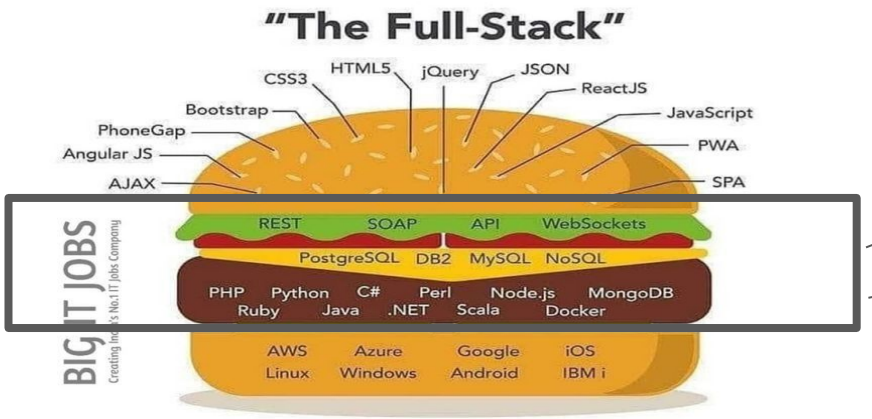
S3 - R3.01 - 2025/2026

UNIVERSITÉ MARIE & LOUIS PASTEUR

IUT Nord Franche-Comté BELFORT - MONTBÉLIARD

Qu'allons nous faire?

- Dev Web back-end (Joseph Azar)
  - 8 semaines
- Dev Web front-end (Stephane Domas)

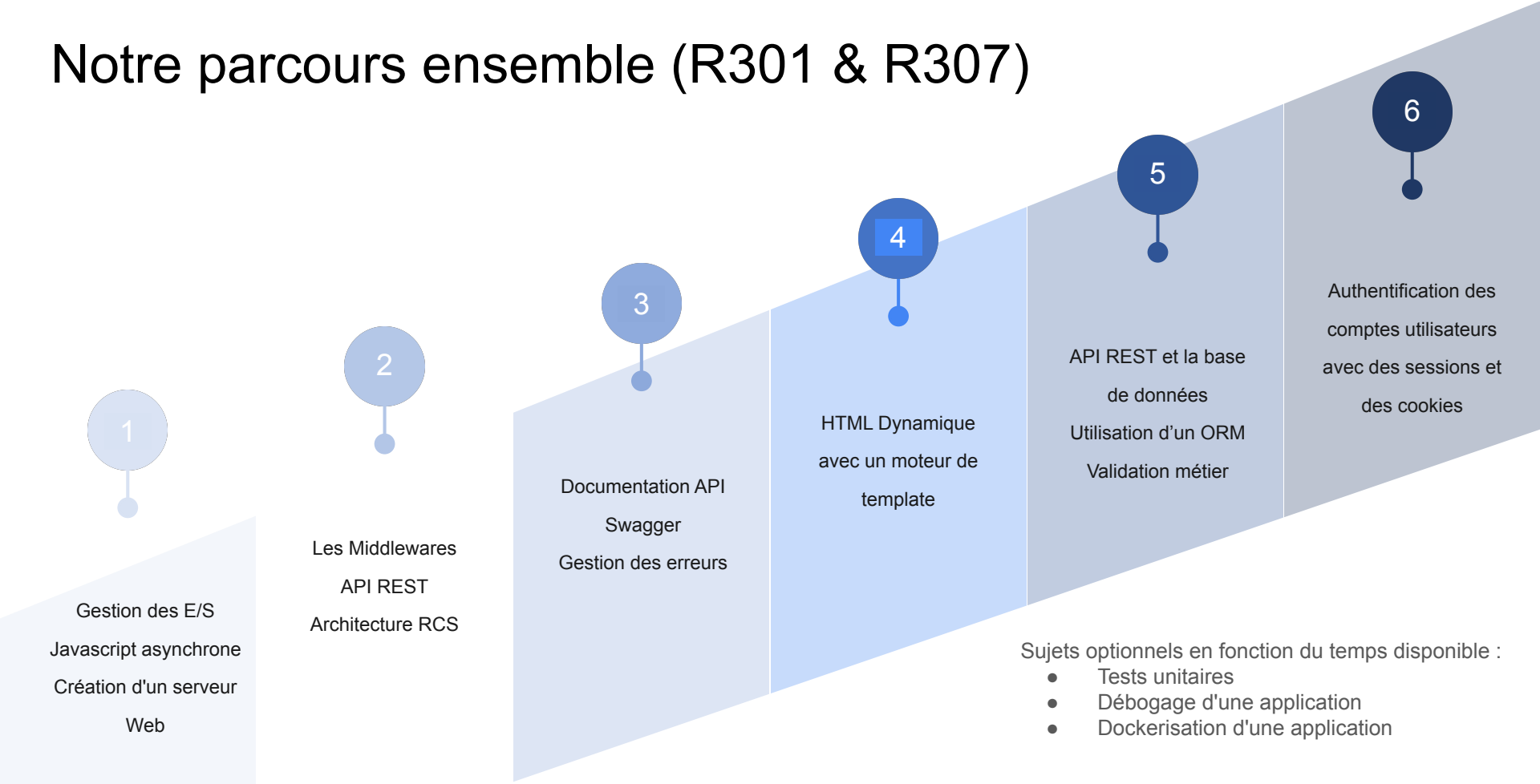


API REST, Serveur Web

Javascript, NodeJS

JSON

Notre parcours ensemble (R301 & R307)



Sujets optionnels en fonction du temps disponible :

- Tests unitaires
- Débogage d'une application
- Dockerisation d'une application

Points importants à savoir :

- Je corrigerai le code de votre SAE et je note le backend et la base de données (50% du SAE)
- Je m'attends à ce que le code SAE couvre ce que j'explique dans mes cours :
  - Bonne architecture API
  - Documentation avec Swagger
  - Connexion à une base de données avec un bon modèle de base de données
  - Authentification avec sessions et cookies
  - Vous comprenez ce que vous faites
- Toute partie du code qui n'est pas couverte dans le cours et dont vous ne savez pas comment répondre dans la soutenance ==> 0
  - Ex: L'utilisation de l'authentification basée sur des jetons sans comprendre son fonctionnement entraînera un score de 0/2,5 sur la partie authentification.
- Nous continuerons ensemble en S4 avec les R401 et R403 où nous aborderons des sujets plus avancés (authentification basée sur des jetons, sécurité, etc.). Ces sujets seront inclus dans la SAE S4.
- Vous aurez un seul examen. La note sur la partie pratique sera tirée de votre code SAE.
- Des points supplémentaires peuvent être ajoutés en fonction de votre travail pendant les cours (des points peuvent également être supprimés).
- Approche: Codelab + Slides + Live Code

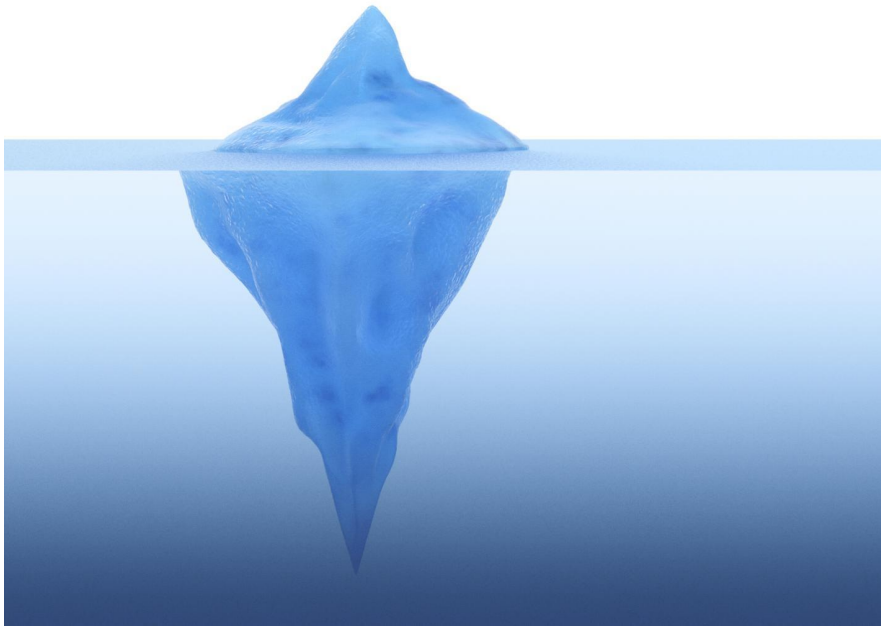
Quels outils/langages allons-nous utiliser ?

Semaine 1

- Qu'est-ce que Node.js ?
- Fonctionnalités
- Pourquoi choisir Node.js ?
- Programmation événementielle asynchrone & Callbacks
- Système de fichiers NodeJS

Fondements du développement backend

Définition et importance du développement backend



Définition du développement backend

Le développement backend concerne la partie serveur des applications, invisible mais essentielle à leur fonctionnement.

Rôle clé du backend

Le backend stocke les données, traite les requêtes et assure la fluidité des interactions utilisateur.

Analogie de l’iceberg

Le frontend est la partie visible, tandis que le backend est la base massive sous la surface qui soutient tout.

Exemples d'interactions backend dans des applications courantes

Connexion utilisateur sécurisée
Le backend vérifie les identifiants et crée une session sécurisée pour l'accès utilisateur sur les réseaux sociaux.

Mise à jour du panier en ligne
Le backend met à jour la base de données du panier, calcule les totaux et prépare la commande pour le paiement.

Transmission des messages chat
Le backend stocke et transmet les messages aux destinataires pour une communication en temps réel.

Rôles essentiels du backend dans une application web

Gestion des données
Le backend gère le stockage, la récupération, la mise à jour et la suppression des données dans les bases de données.

Logique métier
Il applique les règles métier qui définissent le fonctionnement d'une application, comme les calculs de coûts ou la gestion des stocks.

Sécurité
Le backend assure l'authentification, l'autorisation, la validation des entrées et la protection des données sensibles.

Fourniture d'API
Le backend offre des API qui permettent la communication entre le frontend et d'autres applications.

Gestion des serveurs
Les développeurs backend gèrent le déploiement, la montée en charge, la surveillance et la disponibilité des serveurs.

Architecture client-serveur et composants clés du backend

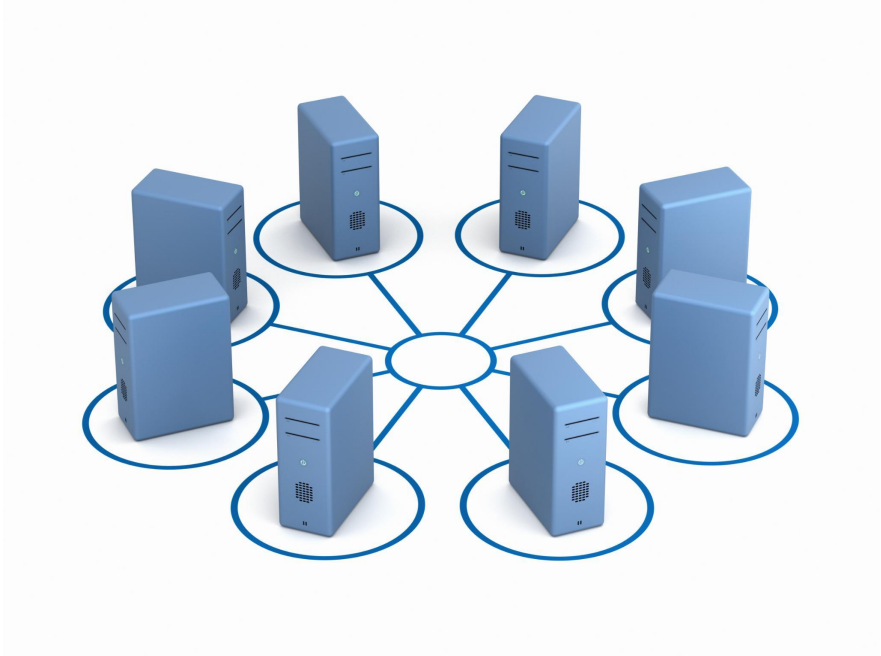
Principe de l’architecture client-serveur et son fonctionnement

Rôle du client
Le client est l'interface utilisateur qui envoie des requêtes au serveur et affiche les réponses reçues.

Fonctionnement du serveur
Le serveur traite les requêtes, interagit avec la base de données et prépare les réponses à envoyer au client.

Échange requêtes-réponses
Le client envoie une requête, le serveur répond avec les données ou confirmations appropriées, assurant une communication fluide.

Affichage des données
Le client reçoit et affiche les données de manière lisible et interactive pour l'utilisateur final.



Serveurs et logiciels serveurs dans le développement backend

Matériel du serveur
Le matériel serveur est une machine physique ou virtuelle puissante qui exécute le code backend et gère de nombreuses requêtes simultanées.

Logiciel serveur
Le logiciel serveur écoute les requêtes des clients, les achemine vers le code backend approprié, et renvoie les réponses, incluant Apache, Nginx, Node.js, Gunicorn.

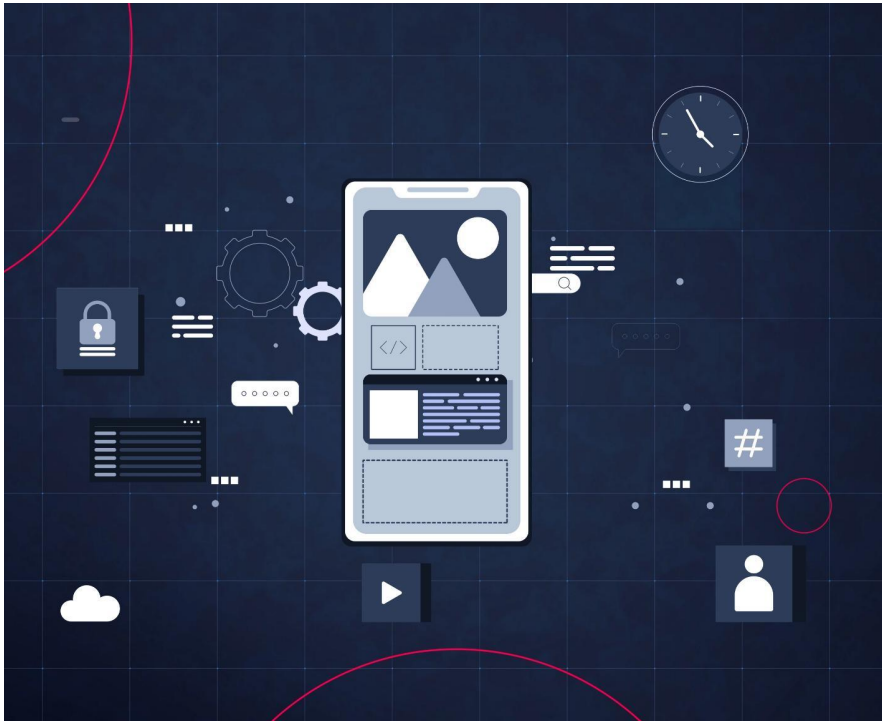
APIs : RESTful et GraphQL, ponts de communication

Rôle des APIs
Les APIs servent de ponts de communication entre les composants logiciels, facilitant l'interaction front-end et back-end.

APIs RESTful
Les APIs RESTful utilisent des méthodes HTTP standard pour gérer les ressources de manière simple et efficace.

APIs GraphQL
GraphQL permet aux clients de demander précisément les données requises, optimisant ainsi les performances des requêtes complexes.

Triade fondamentale du backend
Bases de données, serveurs et APIs forment ensemble la structure essentielle de tout système backend moderne.



Choix du langage et du framework backend

Comparatif des principaux langages et frameworks backend

Python avec Django et Flask
Python offre une syntaxe lisible avec un large soutien communautaire, idéal pour les applications web et l'intégration IA.

JavaScript avec Node.js et Express.js
Node.js permet le développement full-stack JavaScript, excellent pour les applications en temps réel et les microservices.

Ruby on Rails
Ruby on Rails favorise un développement rapide grâce à sa philosophie convention sur configuration, adapté aux prototypes rapides.

PHP avec Laravel et Symfony
PHP alimente une grande partie du web, avec Laravel offrant un cadre moderne pour le développement web traditionnel.

Java avec Spring Boot
Java est robuste et scalable, adapté aux applications d'entreprise et aux systèmes performants avec un bon outillage.

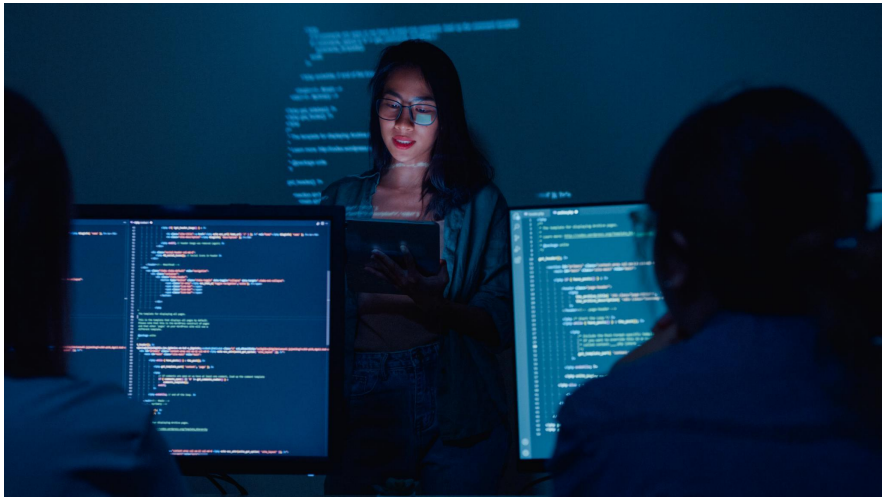
Go (Golang)
Go offre d'excellentes performances et une syntaxe simple, idéal pour les services à haute performance et les microservices.

Recommandations pour débuter en backend selon le profil

Choix selon expérience frontend
Si vous venez du développement frontend avec JavaScript, Node.js avec Express.js est idéal pour réutiliser vos connaissances existantes.

Débutants en programmation
Pour les novices, Python avec Flask est un excellent choix, car il est simple et léger, facilitant l'apprentissage des concepts backend.

Conseil pour choisir
Essayez plusieurs tutoriels dans différents langages pour trouver celui qui vous convient, les concepts backend sont transférables.



Mise en place de l’environnement de développement

Choix du système d’exploitation pour le développement backend

Windows pour le développement
Windows est convivial avec une large compatibilité logicielle. WSL améliore l'expérience de développement backend.

macOS et son noyau Unix
macOS offre une interface soignée et un terminal Unix puissant idéal pour les développeurs backend professionnels.

Linux pour contrôle maximal
Linux est open-source, personnalisable et stable. Parfait pour ceux voulant une expérience serveur native.

Installation des outils essentiels : Git, Node.js, Python, Java, bases de données

Installation de Git
Git est essentiel pour le contrôle de version, la collaboration et la gestion des modifications dans le code source.

Environnements d’exécution : Node.js et Python
Node.js et Python sont des environnements populaires pour le développement backend, chacun nécessitant une installation spécifique.

Java Development Kit (JDK)
Le JDK est requis pour développer en Java, fournissant les outils nécessaires pour compiler et exécuter le code Java.

Systèmes de bases de données
Les bases de données relationnelles comme PostgreSQL et NoSQL comme MongoDB stockent les données applicatives efficacement.

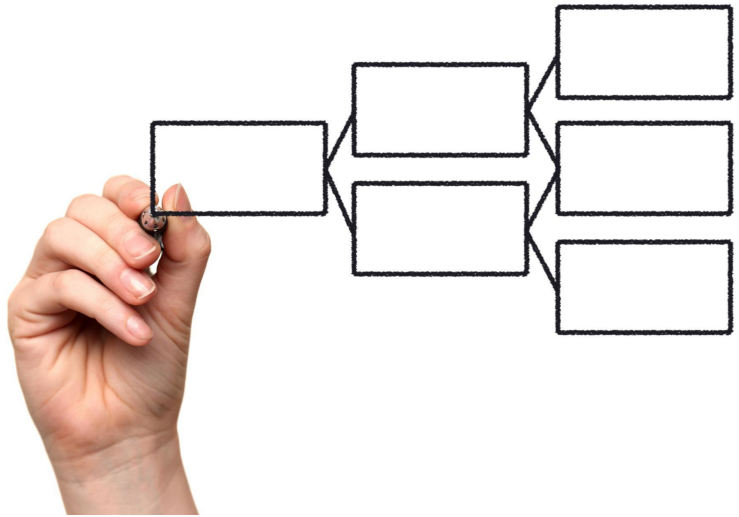
Gestion des dépendances avec les gestionnaires de paquets

Rôle des gestionnaires de paquets
Les gestionnaires de paquets automatisent l'installation, la mise à jour et la gestion des bibliothèques externes de projets logiciels.

npm pour JavaScript
npm est livré avec Node.js et facilite l'installation et l'initialisation des projets JavaScript.

pip pour Python
pip est le gestionnaire standard de paquets Python, utilisé pour installer et enregistrer les dépendances du projet.

Maven et Gradle pour Java
Maven et Gradle sont des outils d'automatisation de build gérant aussi les dépendances via des fichiers de configuration.



Contrôle de version avec Git et GitHub

- Git est un système local de contrôle de version essentiel pour suivre les modifications du code.
- Il permet de revenir à des versions précédentes, de créer des branches et de fusionner des changements.
- GitHub est une plateforme cloud pour héberger des dépôts Git et faciliter la collaboration en équipe.
- La synchronisation s’effectue via les opérations push et pull entre dépôts locaux et distants.
- Commandes Git essentielles : git init, add, commit, status, log, clone, push et pull.
- Maîtriser Git dès le début évite de nombreux problèmes et améliore la collaboration des développeurs.

Environnements de développement intégrés (IDE) recommandés

Fonctions clés des IDE
Les IDE offrent des éditeurs de code avec coloration syntaxique, débogueurs intégrés et automatisation de la construction.

Visual Studio Code
Un éditeur léger, rapide et personnalisable, idéal pour débutants et développeurs polyvalents.

IDE JetBrains
Des IDE puissants avec analyse approfondie, refactorisation et support avancé pour plusieurs langages.

Éditeurs légers Sublime et Atom
Éditeurs rapides et minimalistes, extensibles mais nécessitant plus de configuration manuelle.

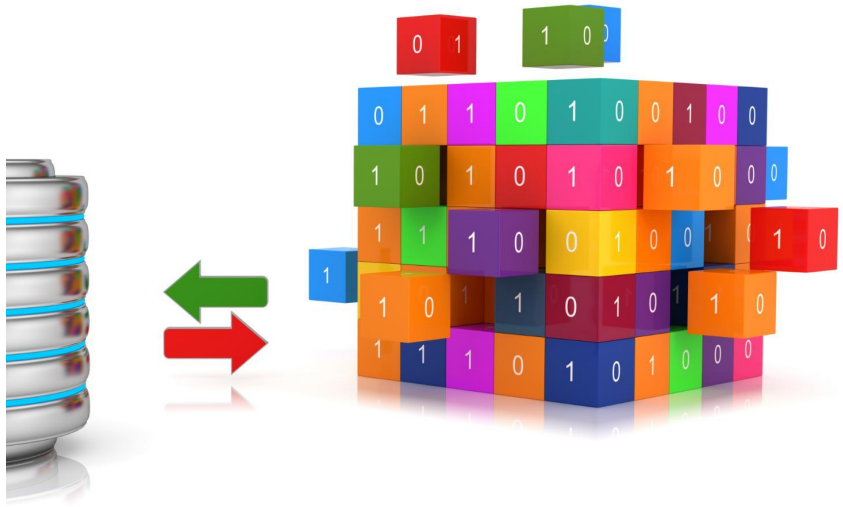
Échange de données : JSON et XML

Format JSON : structure, avantages et exemple

Structures principales de JSON
JSON est basé sur deux structures : paires nom/valeur et listes ordonnées de valeurs, faciles à comprendre et utiliser.

Avantages de JSON
JSON est simple, lisible par l'humain, léger et facile à analyser pour les machines, idéal pour l' échange de données web.

Exemple de données JSON
Un exemple JSON typique pour un article de blog montre des objets imbriqués, des tableaux et des chaînes de caractères organisées.

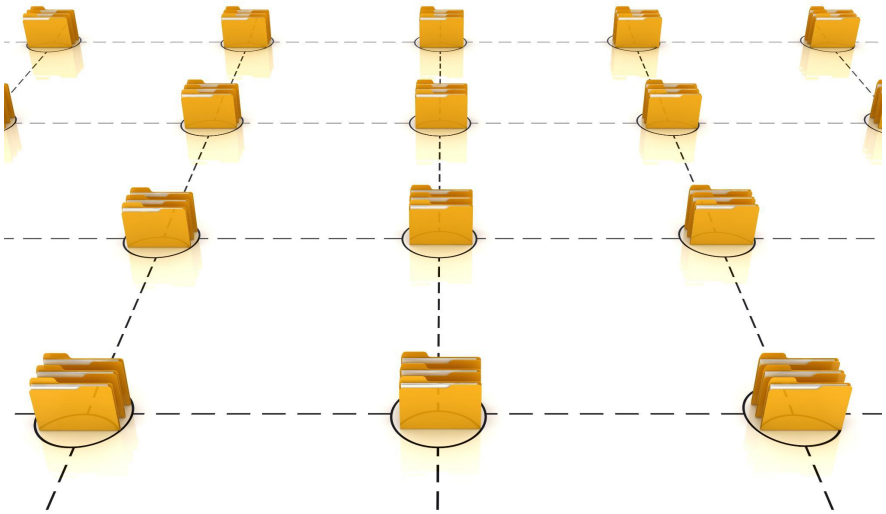


Format XML : structure et exemple comparatif

Nature et usage de XML
XML est un format de données basé sur des balises, largement utilisé dans les systèmes d'entreprise et les services SOAP.

Comparaison avec JSON
XML est plus verbeux et complexe à analyser que JSON, ce qui a réduit son usage dans les nouvelles API RESTful.

Exemple de données XML
Un exemple XML montre une structure en arbre avec des balises imbriquées représentant un article de blog.



Pourquoi JSON est privilégié pour les APIs REST

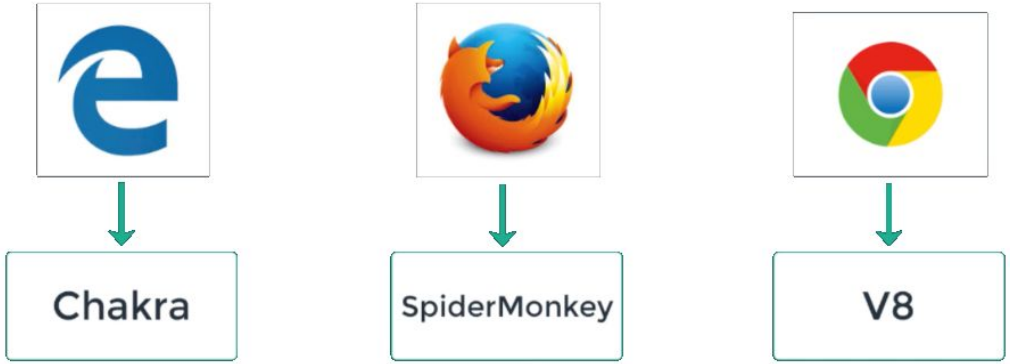
Conciseness et efficacité
JSON est moins verbeux que XML, ce qui réduit la taille des données et accélère leur transmission.

Lisibilité humaine
JSON est plus facile à lire et à écrire, facilitant la compréhension et la maintenance du code.

Analyse et génération simple
JSON est facile à analyser et à générer dans la plupart des langages de programmation.

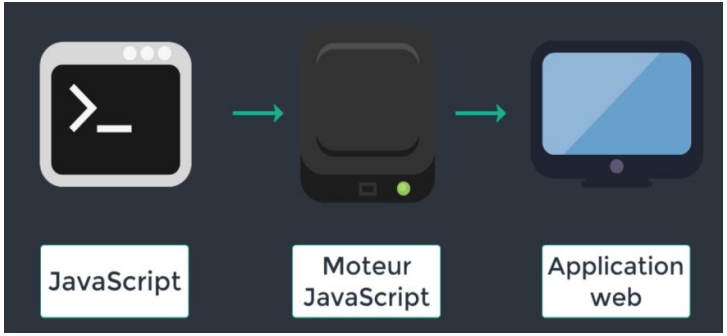
Correspondance directe avec JavaScript
JSON se mappe directement aux objets JavaScript, rendant son utilisation naturelle pour frontends et backends.

Navigateur vs Serveur



NodeJS ne se situe pas là dedans !

Environnement Web
=
Navigateur + Moteur JavaScript



Navigateur vs Serveur

Environnement Serveur
=
Node.js + Moteur V8
Chrome + Modules internes

Qu'est-ce que Node.js ?

Amplication
@amplication

1 What is Node.js?

- It is an open-source asynchronous event-driven JavaScript runtime.

- It is built on Chrome's JavaScript Engine (V8 Engine).

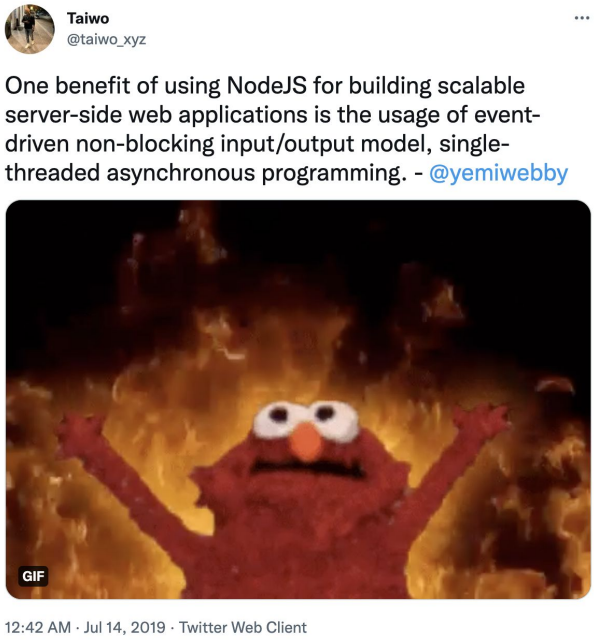
- It allows developers to use Javascript on the server-side.

11:56 AM · Dec 5, 2021 · FeedHive.io

- Il s'agit d'un environnement d'exécution JavaScript open source asynchrone piloté par les événements.
- Il est construit sur le moteur JavaScript de Chrome (moteur V8).
- Il permet aux développeurs d'utiliser Javascript côté serveur.

Qu'est-ce que Node.js ?

Asynchronous event-driven JavaScript

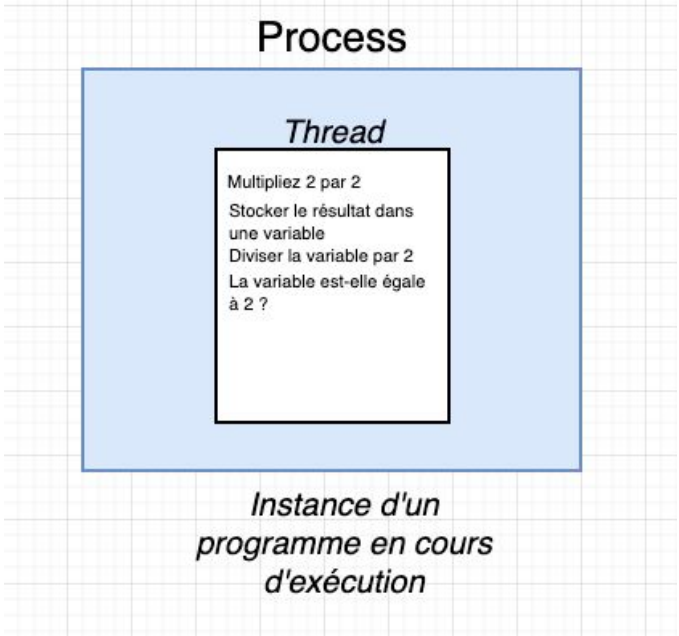


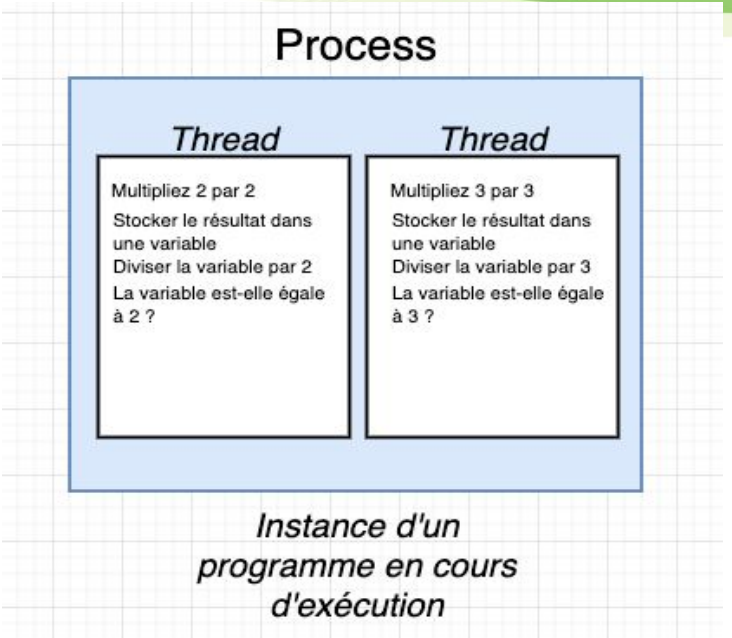
“L'un des avantages de l'utilisation de NodeJS pour la création d'applications Web évolutives côté serveur est l'utilisation d'un modèle d'entrée/sortie non bloquant piloté par les événements et d'une programmation asynchrone à un seul thread.”

“L'élimination des processus de blocage grâce à l'utilisation d'E/S asynchrones pilotées par les événements est le principal principe organisationnel de Node.” – Mastering Node.js - Second Edition

Qu'est-ce que Node.js ?

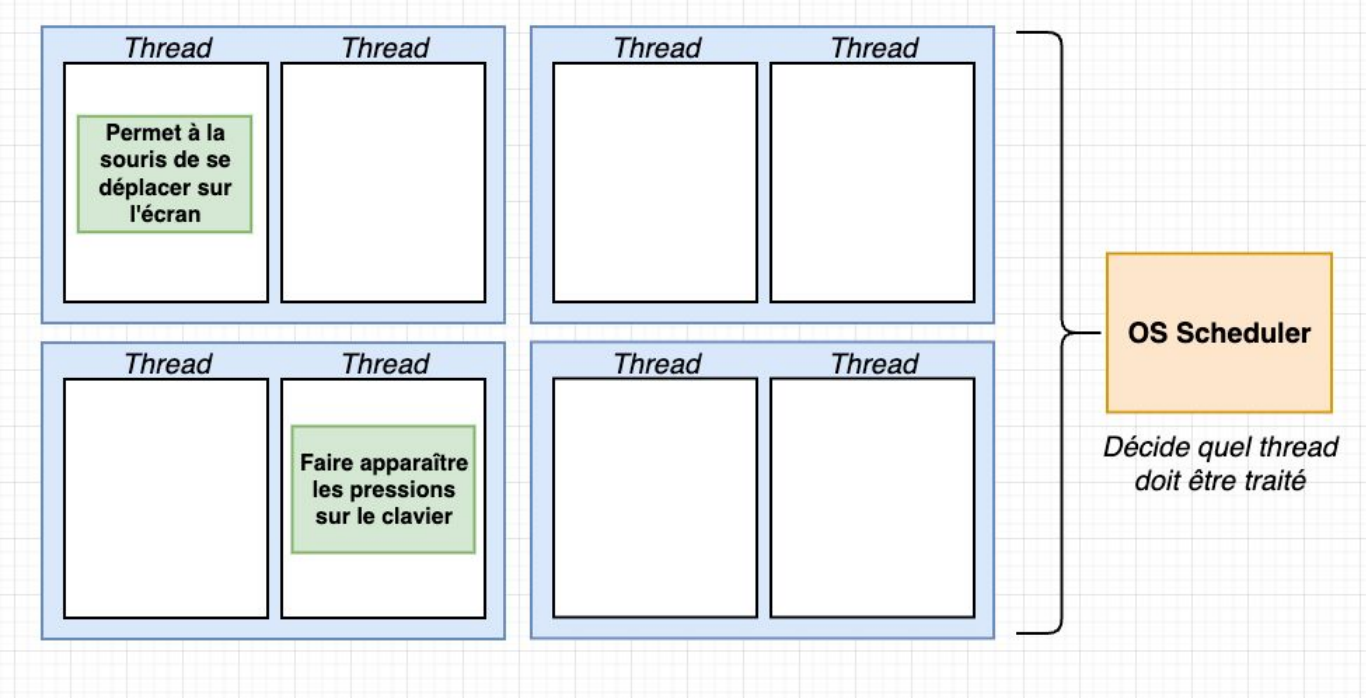
Threads





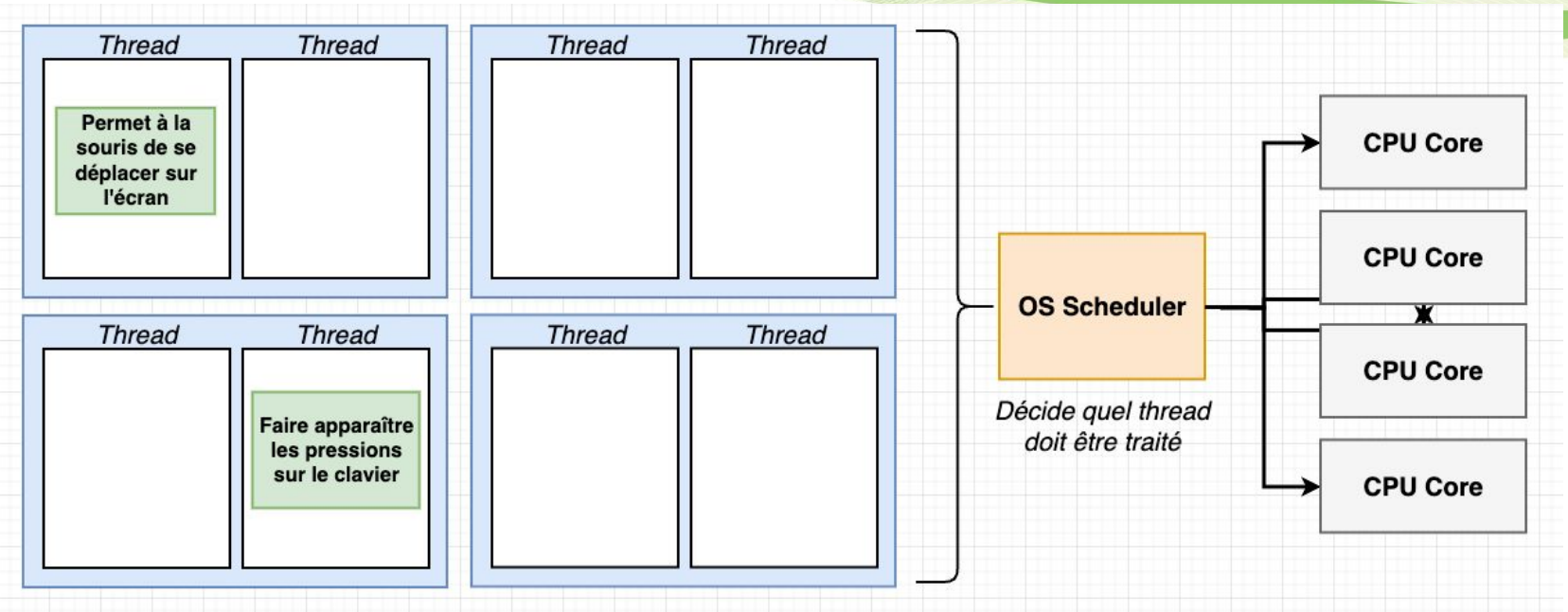
Qu'est-ce que Node.js ?

Threads



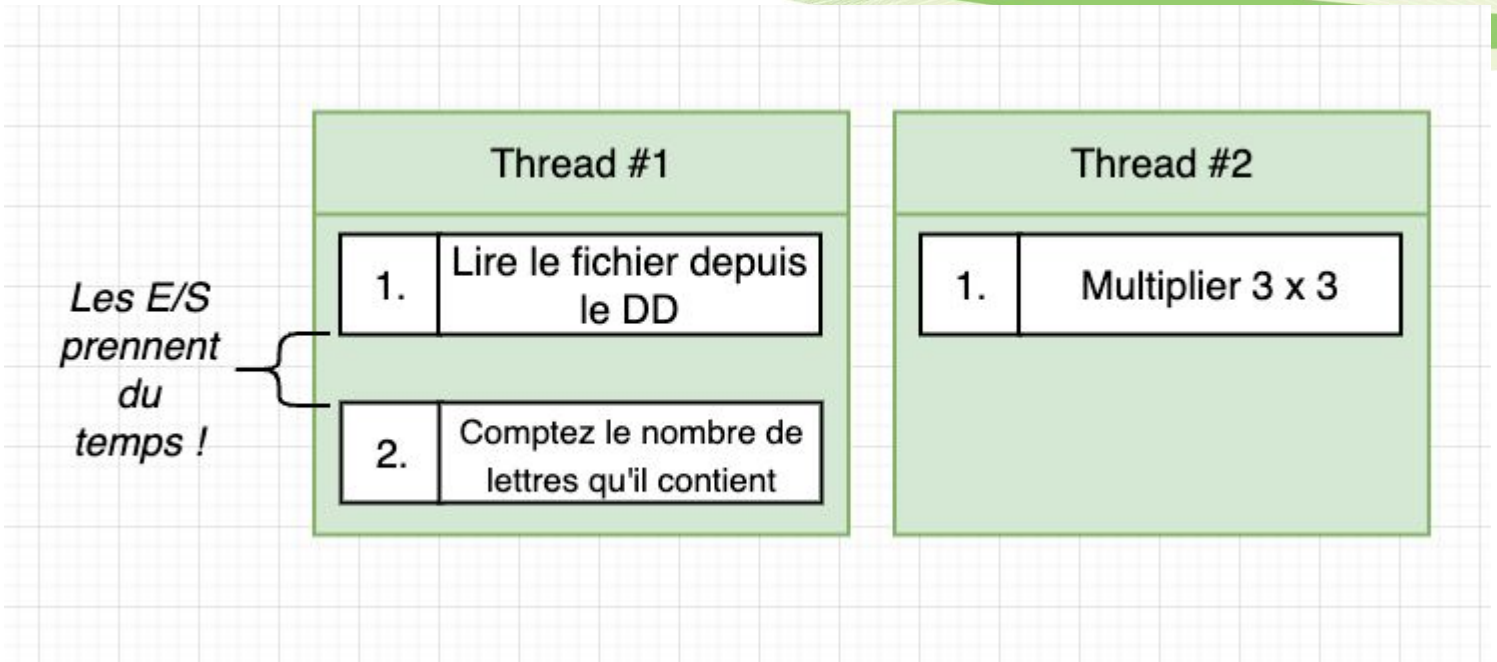
Qu'est-ce que Node.js ?

Threads



Qu'est-ce que Node.js ?

Threads



Qu'est-ce que Node.js ?

Threads

- Les threads sont des unités d'instruction en attente d'exécution par le CPU.
- La décision de l'ordre d'exécution de ces threads est appelée ordonnancement (**Scheduling**).
- L'ordonnancement est géré par le système d'exploitation.
- Pour améliorer la vitesse de traitement des threads, on peut :
  - soit ajouter plus de cœurs CPU à notre machine,
  - soit permettre à notre ordonnanceur OS de détecter de longues pauses dues à des opérations d'entrée/sortie coûteuses.

Qu'est-ce que Node.js ?

Asynchronous event-driven JavaScript

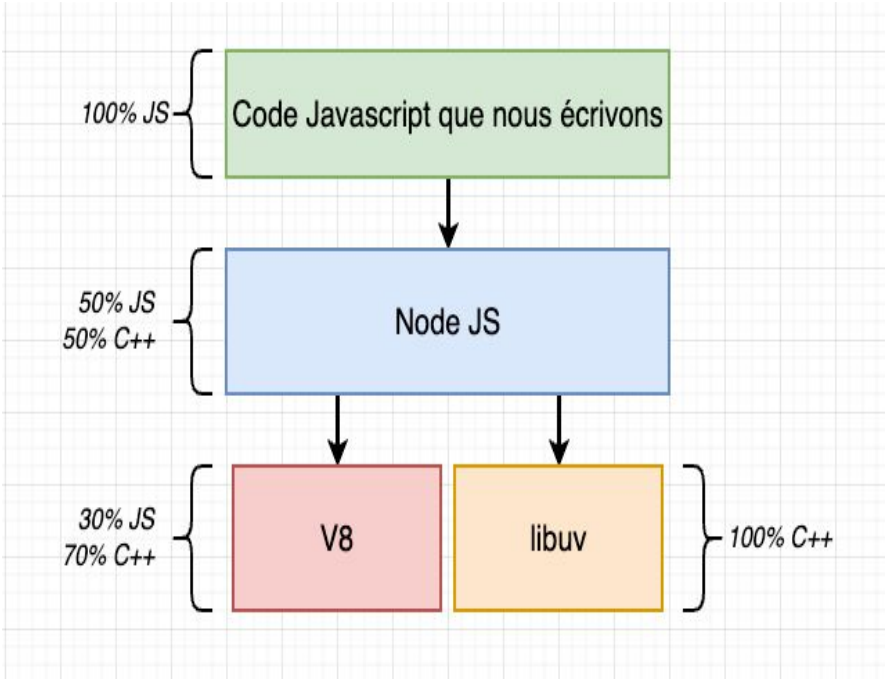
Threads versus événements utilisant des rappels (Callbacks)

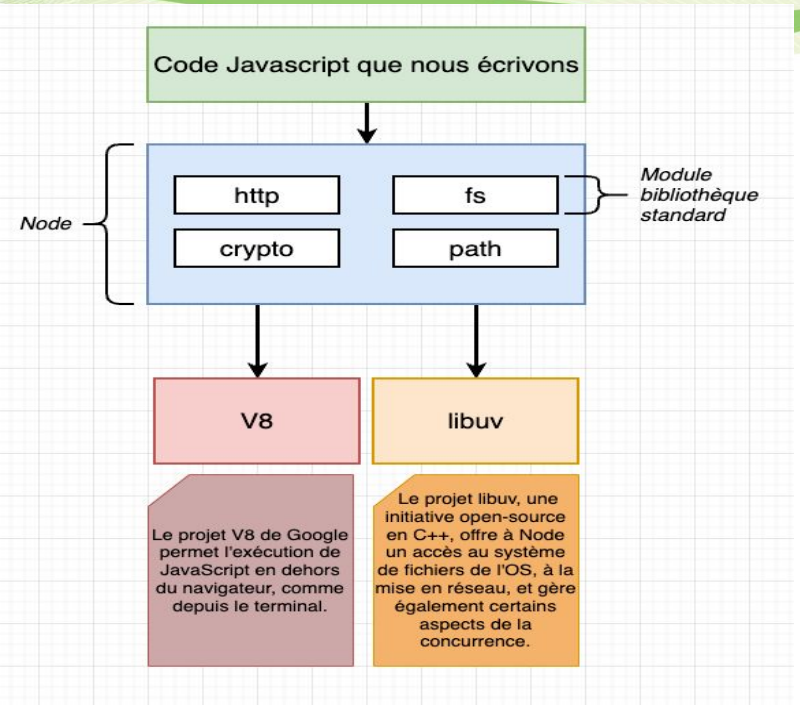
```
request = readRequest(socket);
reply = processRequest(request);
sendReply(socket, reply);
moreWork(); // will run after sendreply()
```

```
readRequest(socket, function(request) {
 processRequest(request,
    function (reply) {
        sendReply(socket, reply);
    });
});
moreWork(); // will run before readRequest
```

Qu'est-ce que Node.js ?

Éléments internes de NodeJS





Qu'est-ce que Node.js ?

Moteur Javascript V8

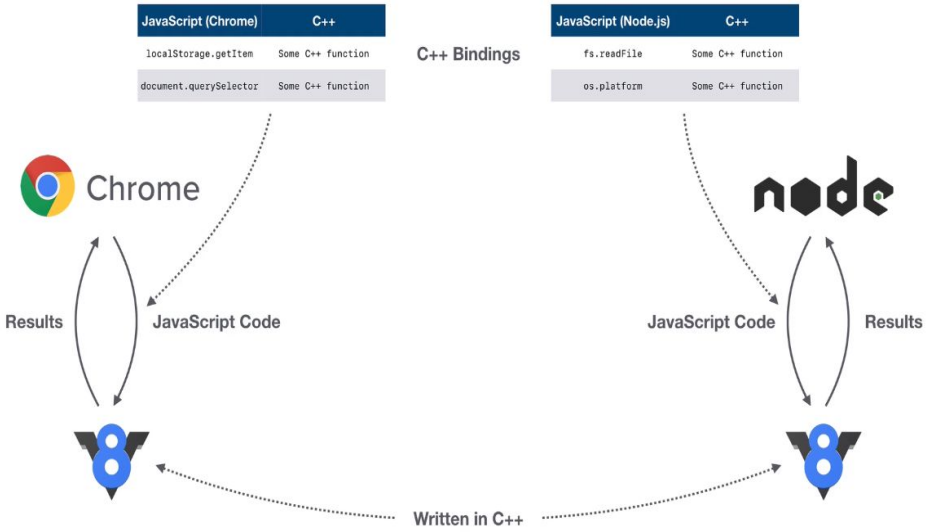
La bibliothèque V8 fournit à Node.js un moteur JavaScript (un programme qui convertit le code Javascript en code machine ou de niveau inférieur que les microprocesseurs peuvent comprendre), que Node.js contrôle via l'API V8 C++. V8 est maintenu par Google, pour une utilisation dans Chrome.

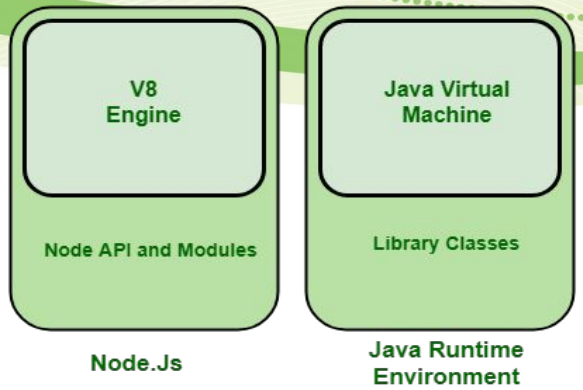
Le moteur Chrome V8 :

- Le moteur V8 est écrit en C++ et utilisé dans Chrome et Nodejs.
- Il implémente ECMAScript comme spécifié dans ECMA-262.
- Le moteur V8 peut fonctionner de manière autonome, nous pouvons l'intégrer à notre propre programme C++.

Qu'est-ce que Node.js ?

Moteur Javascript V8





- V8 est essentiellement une bibliothèque C++ autonome.
- Vous pouvez intégrer V8 dans votre programme C++ et exécuter un programme JavaScript à partir de celui-ci.

Disons que nous voulons ajouter la possibilité d'avoir des déclarations comme print('hello world') en plus de console.log('Hello World') dans notre code JavaScript. Avec le moteur V8, qui est déjà open source, nous pouvons ajouter notre propre implémentation C++ de la fonction d'impression.

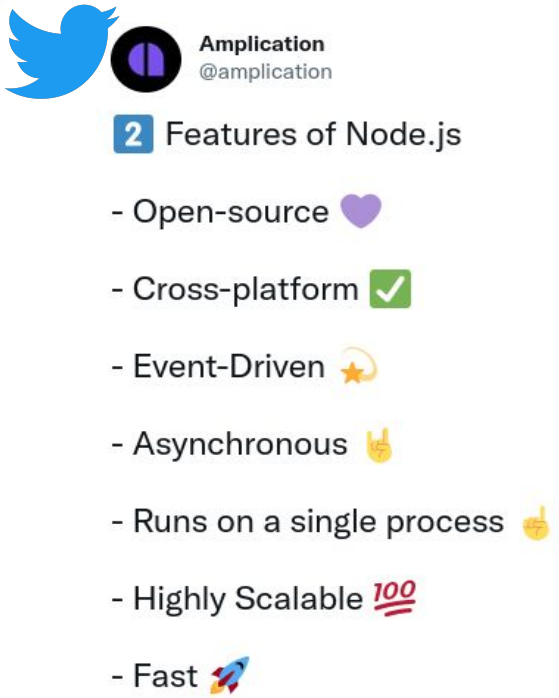
Qu'est-ce que Node.js ?

Javascript côté serveur

“Lorsqu'il a conçu Node, JavaScript n'était pas le choix de langue d'origine de Ryan Dahl. Pourtant, lors de l'exploration, il a trouvé un bon langage moderne sans opinions sur les **flux**, le **système de fichiers**, la **gestion des objets binaires**, les **processus**, la **mise en réseau** et d'autres capacités que l'on s'attendrait à voir exister dans un langage système. JavaScript, strictement limité au navigateur, n'avait pas implémenté, ces fonctionnalités.”

— *Mastering Node.js - Second Edition*

Fonctionnalités



Les capacités de Node.js

Node.js est une plate-forme pour écrire des applications JavaScript en dehors des navigateurs Web. Ce n'est pas l'environnement JavaScript que nous connaissons dans les navigateurs Web ! Bien que Node.js exécute le même langage JavaScript que nous utilisons dans les navigateurs, il ne possède pas certaines des fonctionnalités associées au navigateur. Par exemple, il n'y a pas de DOM HTML intégré dans Node.js.

Au-delà de sa capacité native à exécuter JavaScript, les modules intégrés offrent les fonctionnalités suivantes :

- Outils de ligne de commande (dans le style de script shell)
- Un programme de type terminal interactif , c'est-à-dire une **boucle de lecture-évaluation-impression (REPL)**
- Excellentes fonctions de contrôle de processus pour superviser les processus enfants
- Un objet tampon pour traiter les données binaires
- Sockets TCP ou UDP avec rappels complets basés sur les événements
- Recherche DNS
- Un serveur client HTTP, HTTPS et HTTP/2 superposé à l'accès au système de fichiers de la bibliothèque TCP
- Prise en charge intégrée des tests unitaires rudimentaires via des assertions

– Node.js Web Development, Fifth Edition

Pourquoi choisir Node.js ?

Node.js, un langage utile et efficace

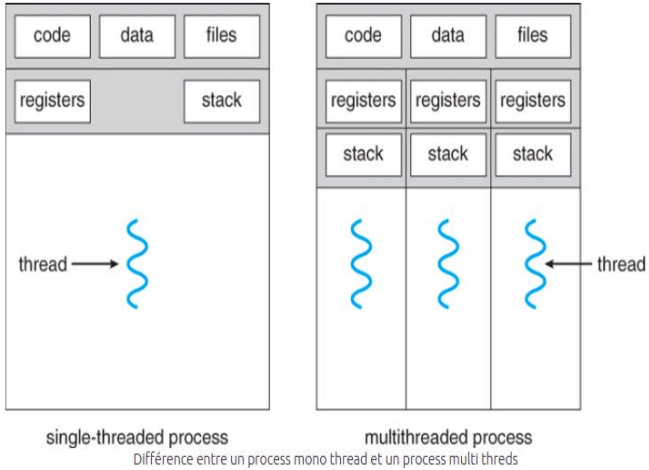
- Le moteur JavaScript V8 développé par Google, qui permet d'exécuter du code JavaScript à l'intérieur de Google Chrome et, grâce à Node, directement sur le serveur
- Open-Source avec une communauté de développeurs active
- Conçu pour une programmation efficace et asynchrone côté serveur
- Une boucle d'événements, appelée aussi EVENT LOOP NodeJS, permettant d'exécuter plusieurs opérations simultanées de façon asynchrone et non bloquante en tirant profit des multiples fils d'exécution (multithreading) des noyaux des processeurs modernes

Pourquoi choisir Node.js ?

Node est très adapté pour les serveurs web

Node JS est single-threaded: il ne s'exécute que sur une seule instance (un thread ou fil d'exécution) du système d'exploitation, à l'inverse d'un serveur fonctionnant en PHP (avec Apache HTTP Server par exemple) qui est, lui, multi-threaded

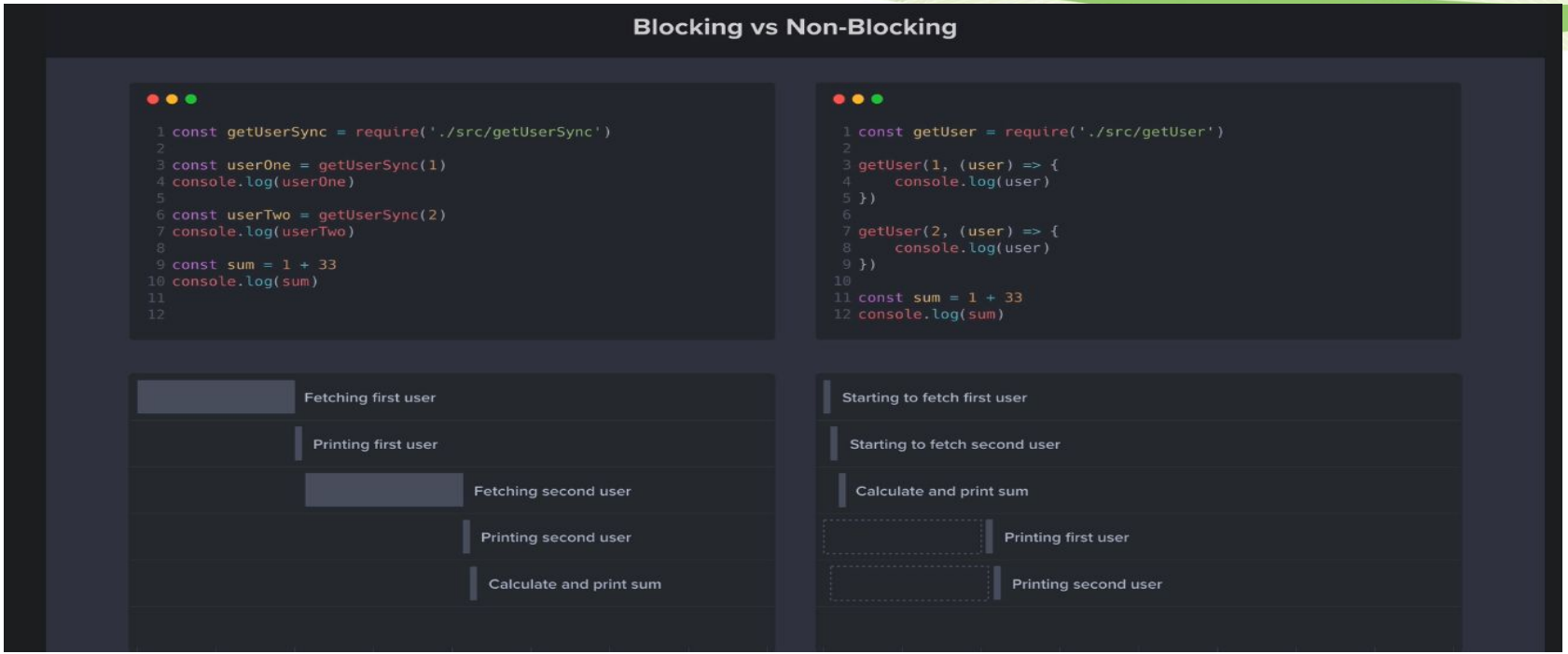
Cette caractéristique de Node lui permet de gérer les requêtes de façon non bloquantes. Là où un serveur PHP mobilise un thread pour réceptionner une requête et attendre qu'elle passe à travers tout le code pour retourner une réponse au client, le serveur Node va gérer chaque requête de manière asynchrone au sein d'un seul et unique thread. La requête est ainsi réceptionnée puis envoyée dans la callback queue où le thread de Node sert les réponses une par une



Différence entre un process mono thread et un process multi threds

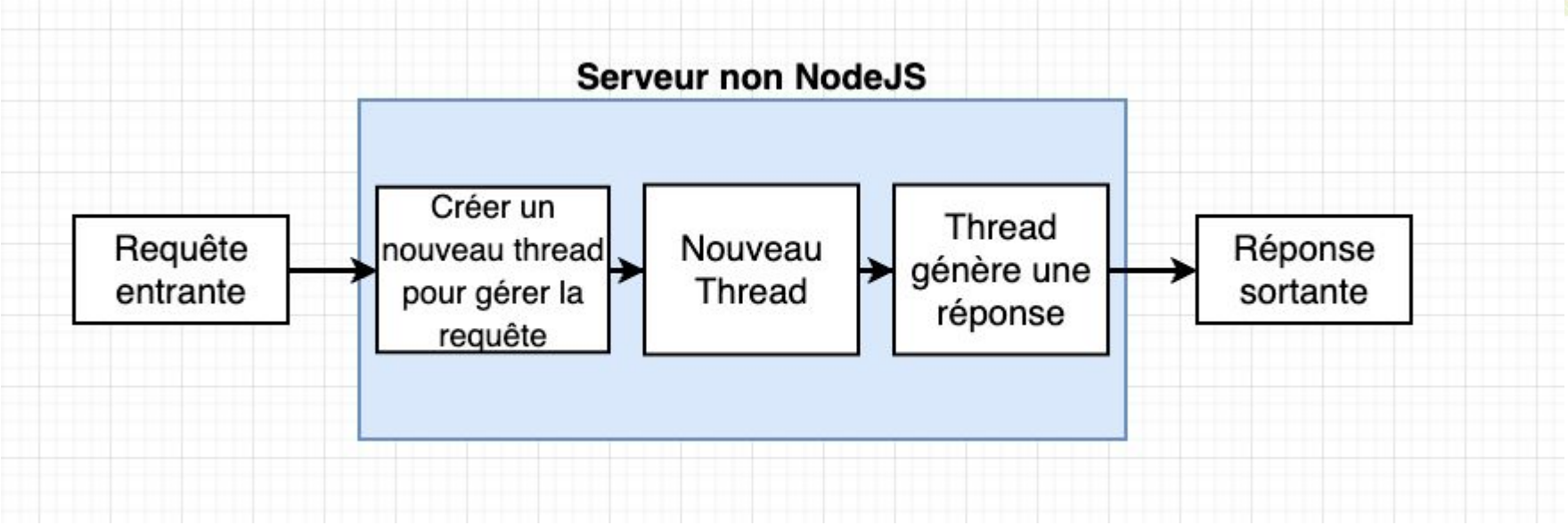
Pourquoi choisir Node.js ?

Bloquant vs Non bloquant



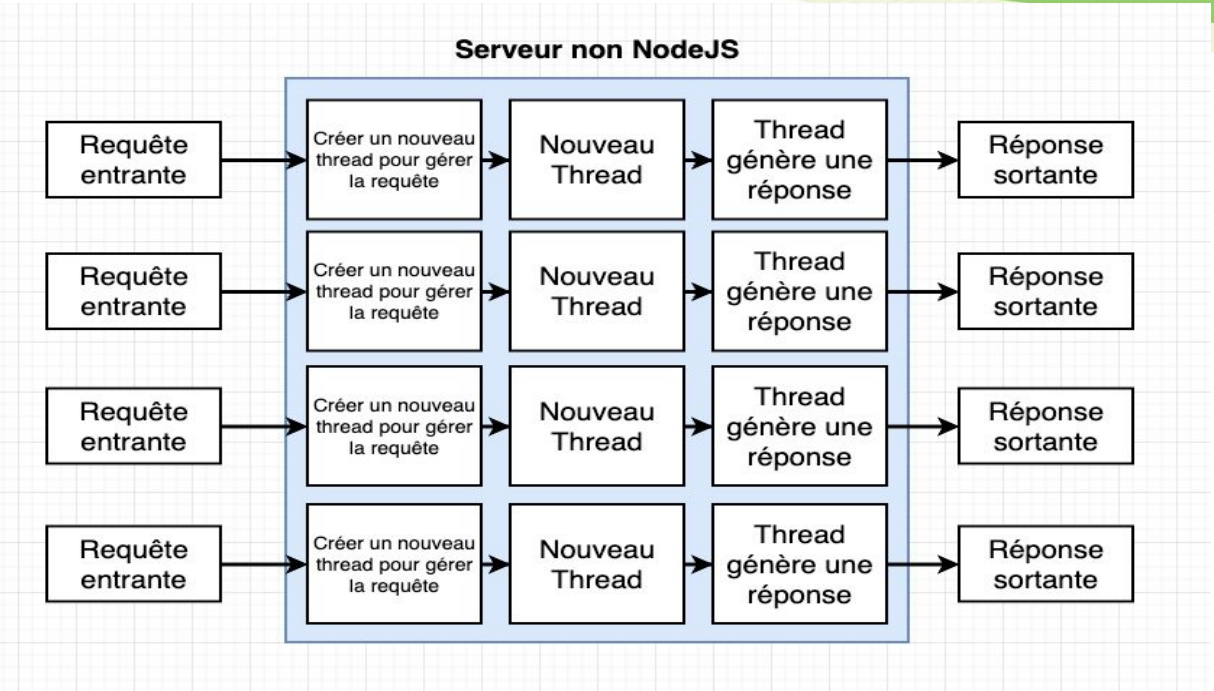
Pourquoi choisir Node.js ?

Bloquant vs Non bloquant



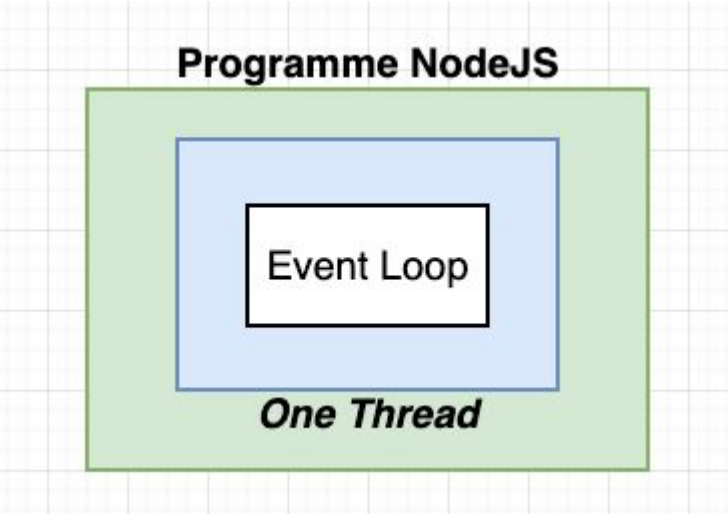
Pourquoi choisir Node.js ?

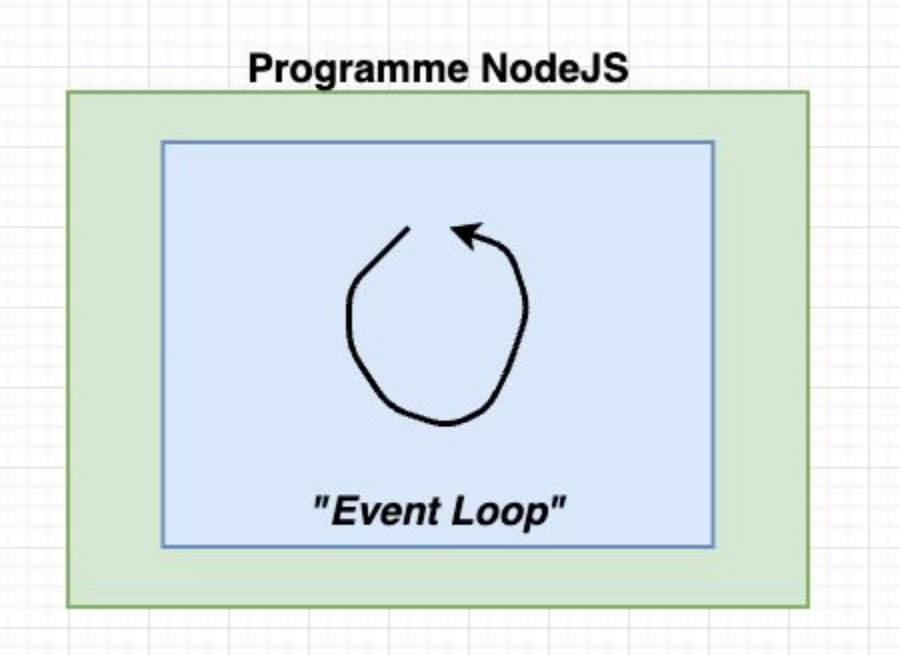
Bloquant vs Non bloquant



Pourquoi choisir Node.js ?

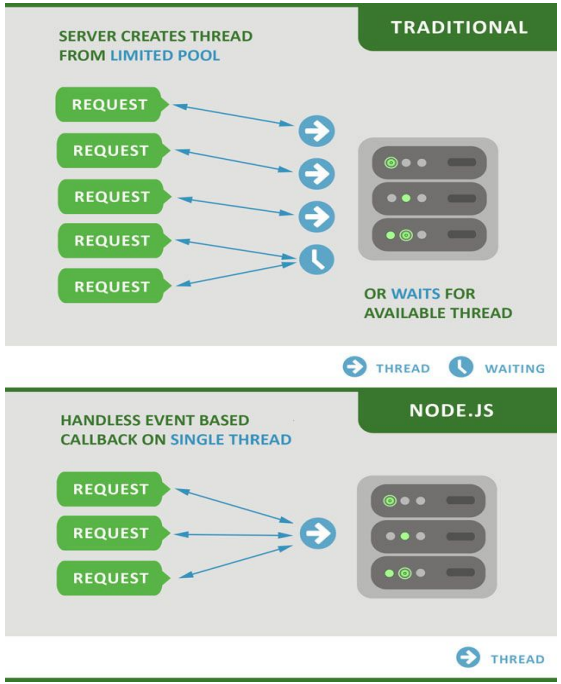
Bloquant vs Non bloquant

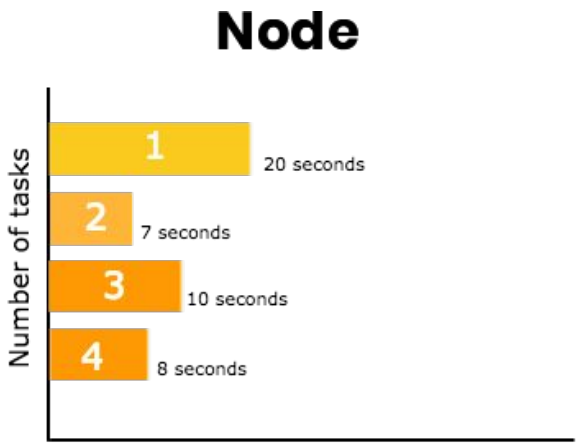


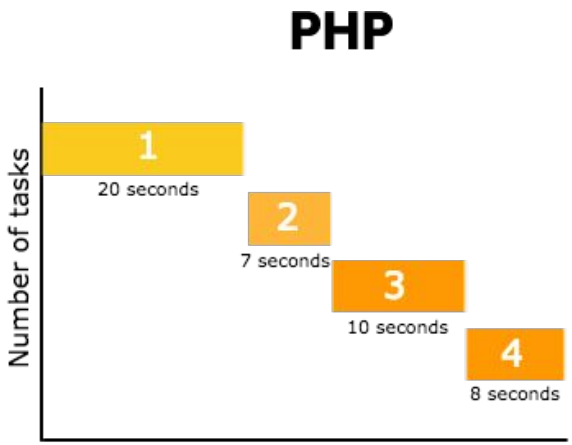


Pourquoi choisir Node.js ?

Bloquant vs Non bloquant







Pourquoi choisir Node.js ?

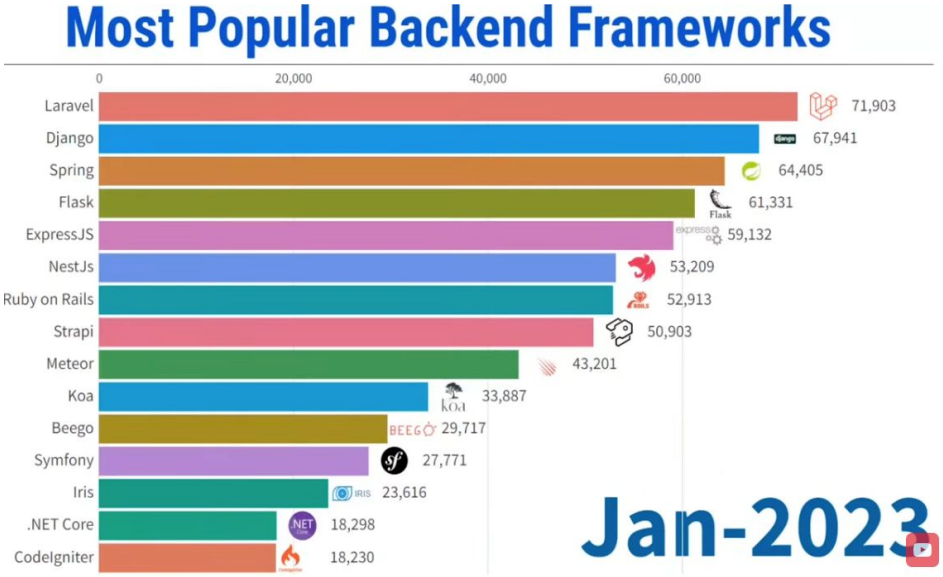
Node est basé sur JavaScript

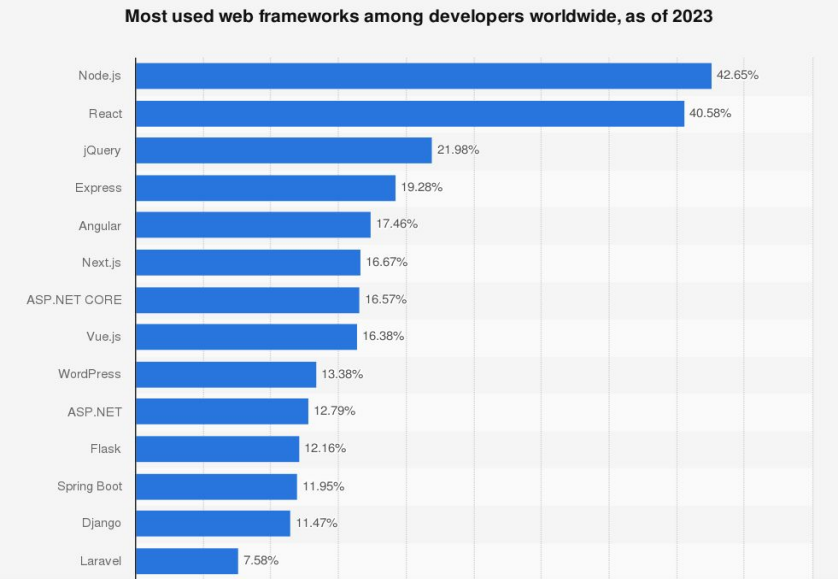
Créé en 1995 par Netscape et longtemps cantonné à réaliser de petites animatio
les sites web, JavaScript est aujourd'hui le langage de développement web le plus populaire au monde. Cela peut s'illustrer par plusieurs statistiques dont les suivantes :

- JavaScript est le langage le plus utilisé par les répondants au Stack Overflow Developers Survey de 2019.
- C'est également le langage le plus présent dans les contributions GitHub depuis au moins cinq ans.
- L'utilisation de JavaScript comme langage de programmation côté serveur permet également d'abolir les barrières entre front-end et back-end, permettant de monter une équipe fullstack plus facilement.
- Enfin, Javascript est le langage utilisé par une écrasante majorité de sites web pour le front-end mais est également de plus en plus populaire pour le back-end.

Pourquoi choisir Node.js ?

Le marché de l'emploi est en demande





Référence : https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2023/

Pourquoi choisir Node.js ?

Succès rapide et fulgurant de Node.js : Node.js s'est rapidement imposé dans l'industrie, justifiant une exploration des raisons de son adoption.
JavaScript en tant que langage principal : Node.js utilise JavaScript, un langage universellement répandu parmi les développeurs web et interprété par tous les navigateurs, simplifiant ainsi le développement full-stack.
Unification du langage dans l'équipe : Utiliser un seul langage pour le développement frontend et backend facilite la communication au sein des équipes, réduisant les problèmes liés à la maîtrise de plusieurs langages.
Simplification et accélération des projets : Node.js permet de simplifier les projets et d'accélérer le développement en utilisant un langage unique, déjà familier aux développeurs.
Rapidité d'exécution : Node.js est extrêmement rapide grâce au moteur JavaScript V8 de Chrome et à son architecture non bloquante, idéale pour les applications en temps réel comme les messageries instantanées.
Environnement d'exécution léger : Node.js est très léger par défaut, permettant d'ajouter uniquement les modules nécessaires, ce qui optimise la performance.
Large écosystème de librairies open source : Node.js bénéficie d'un vaste écosystème de modules et de librairies disponibles via NPM, facilitant l'intégration de fonctionnalités dans les projets.
Popularité et support communautaire : Node.js est très populaire, ce qui assure une abondance de ressources d'aide et une grande disponibilité de modules prêts à l'emploi, utilisés par de grandes entreprises comme Microsoft, PayPal, et LinkedIn.
Technologie pérenne : La popularité et l'adoption généralisée de Node.js indiquent qu'il est une technologie fiable, susceptible de rester pertinente dans les années à venir.

Qui utilise Node.js?

- Paypal – De nombreux sites au sein de Paypal ont également commencé la transition vers Node.js.
- LinkedIn - LinkedIn utilise Node.js pour alimenter ses serveurs mobiles, qui alimentent les produits iPhone, Android et Web mobile.
- Mozilla a implémenté Node.js pour prendre en charge les API de navigateur qui compte un demi-milliard d'installations.
- Ebay héberge son service d'API HTTP à l'aide de Node.js .

Différents types de programme Node

- Des applications Web
- Outils de ligne de commande
- Applications de bureau
- Applications IoT (Node-RED)

Express

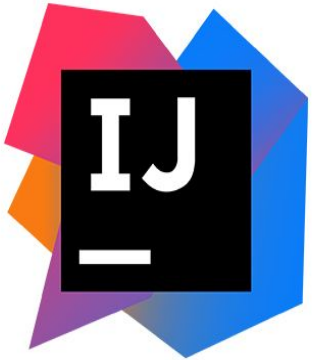
yargs

ELECTRON

Node-RED

Environnement de développement

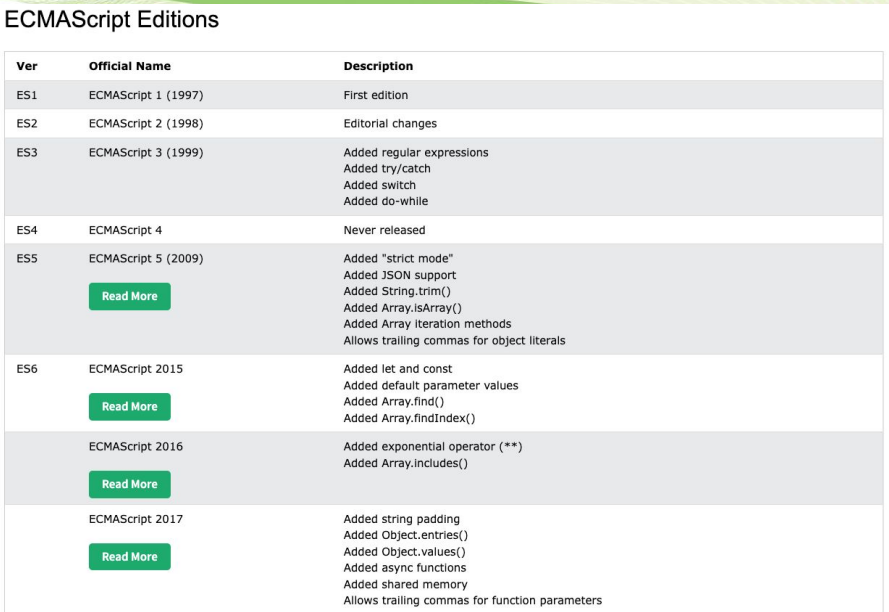
Visual Studio Code



Rappel sur les notions clés de Javascript

Javascript et ECMAScript

- **JavaScript et ECMAScript :** JavaScript est une implémentation du standard ECMAScript, avec différentes versions apportant des fonctionnalités variées.
- **Évolution des versions ECMAScript :** Le langage évolue avec le temps, les versions post-2015 étant nommées par année (2016, 2017, etc.), chacune introduisant des nouveautés.
- **Focus sur les syntaxes clés :** Le chapitre se concentre sur des syntaxes et fonctionnalités essentielles d'ECMAScript 2015-2018, comme let, const, les boucles for...of, et les fonctions fléchées, importantes pour la suite.



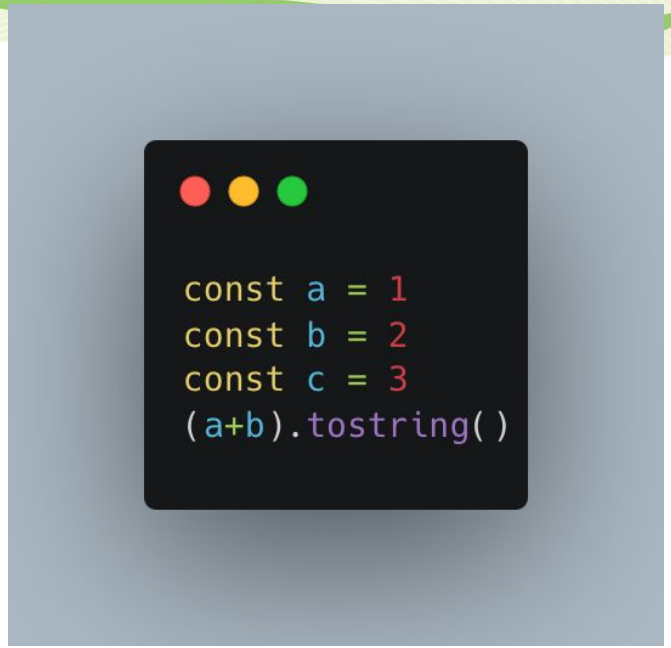
ECMAScript Editions

Ver	Official Name	Description
ES1	ECMAScript 1 (1997)	First edition
ES2	ECMAScript 2 (1998)	Editorial changes
ES3	ECMAScript 3 (1999)	Added regular expressions Added try/catch Added switch Added do-while
ES4	ECMAScript 4	Never released
ES5	ECMAScript 5 (2009) Read More	Added "strict mode" Added JSON support Added String.trim() Added Array.isArray() Added Array iteration methods Allows trailing commas for object literals
ES6	ECMAScript 2015 Read More	Added let and const Added default parameter values Added Array.find() Added Array.findIndex()
	ECMAScript 2016 Read More	Added exponential operator (**) Added Array.includes()
	ECMAScript 2017 Read More	Added string padding Added Object.entries() Added Object.values() Added async functions Added shared memory Allows trailing commas for function parameters

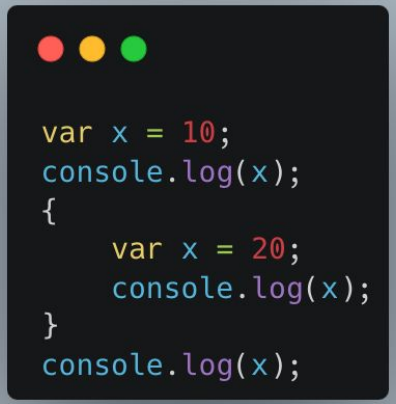
https://www.w3schools.com/js/js_versions.asp

Point-virgule?

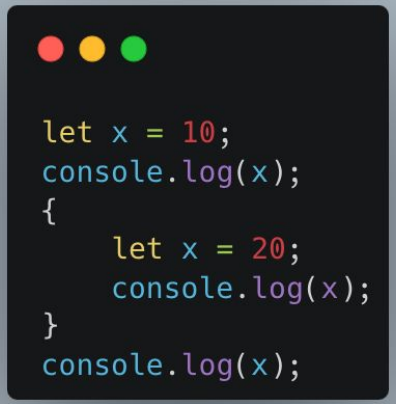
Points-virgules en JavaScript : Optionnels mais débattus : Les points-virgules en JavaScript sont optionnels, suscitant des débats parmi les développeurs sur leur nécessité.
Insertion automatique des points-virgules : JavaScript ajoute automatiquement des points-virgules là où ils sont omis, pour interpréter correctement le code.
Risques de comportements inattendus : L'absence de points-virgules peut entraîner des erreurs d'interprétation, notamment dans des cas spécifiques où JavaScript place les points-virgules au mauvais endroit.
Sécurité et habitudes de codage : Ajouter systématiquement des points-virgules en fin d'instruction peut prévenir les erreurs et garantir un comportement prévisible du code, bien que certains préfèrent s'en passer pour des raisons esthétiques ou par habitude d'autres langages.



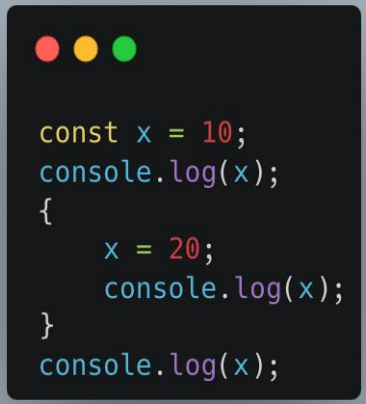
Les variables et constantes



```
var x = 10;
console.log(x);
{
    var x = 20;
    console.log(x);
}
console.log(x);
```



```
let x = 10;
console.log(x);
{
    let x = 20;
    console.log(x);
}
console.log(x);
```



```
const x = 10;
console.log(x);
{
    x = 20;
    console.log(x);
}
console.log(x);
```

Les boucles for … of

```
const tab = ["python", "java", "javascript"]

for (let i = 0; i < tab.length; i++) {
    console.log(tab[i])
}
```

```
const tab = ["python", "java", "javascript"]

for (const langage of tab) {
    console.log(langage)
}
```

Les fonctions "fléchées"

```
function sum(a,b){
    return a+b
}

const result = sum(1,2)
console.log(result)
```

```
const sum2 = (a,b) => {
    return a+b
}
const result2 = sum2(1,2)
console.log(result2)
```

Les paramètres de fonction par défaut

```
const direSalut = (formule = "Salut", prenom = "Bob") => {
  console.log(`${formule} ${prenom}`);
}

direSalut("Bonjour", "Alice");
direSalut();
direSalut("Hey");
direSalut(prenom = "Alice");
direSalut(prenom = "Alice", formule = "Hey");
```

```
(base) josephazar@Josephs-MacBook-Pro-2 fundamentals % node params_fonctions.js
Bonjour Alice
Salut Bob
Hey Bob
Alice Bob
Alice Hey
```

Rappel sur les objets (JSON)

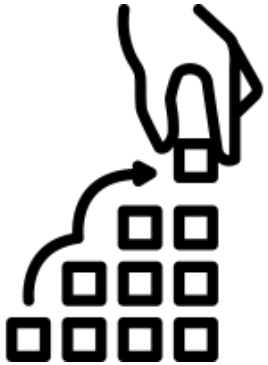
```
const pc = {
    ram: 16,
    stockage: 512,
    processeur: 'i7',
    type: 'portable',
    annee: 2020,
    logiciels: ['word', 'excel', 'powerpoint'],
    upgrade: function() {
        this.ram += 16;
        this.stockage += 512;
    },
    upgrade2: () => {
        console.log(this);
        this.ram += 16;
        this.stockage += 512;
    }
}
```

```
console.log(pc);
console.log(typeof pc);
console.log(pc.stockage);
console.log(pc.logiciels);

for (let logiciel of pc.logiciels) {
    console.log(logiciel);
}

pc.upgrade();
console.log(pc);

const pcJSON = JSON.stringify(pc);
console.log(pcJSON);
console.log(typeof pcJSON);

const pc2 = JSON.parse(pcJSON);
console.log(pc2);

pc.upgrade2();
console.log(pc);
```

Les classes

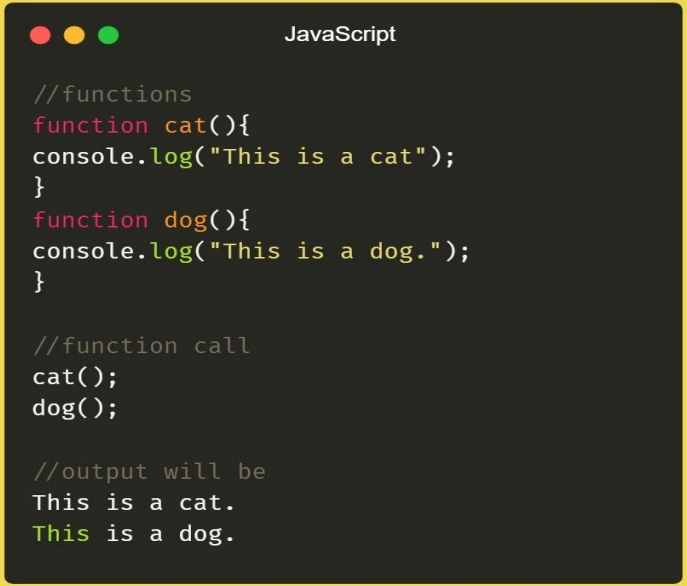
```
class Voiture {
  constructor(marque, modele, couleur) {
    this.marque = marque;
    this.modele = modele;
    this.couleur = couleur;
  }

  getInfos() {
    return `La voiture est une ${this.marque} ${this.modele} de couleur ${this.couleur}`;
  }
}

const voiture1 = new Voiture('Peugeot', '208', 'rouge');
console.log(voiture1);
console.log(voiture1.getInfos());

voiture1.marque = 'Renault';
console.log(voiture1.getInfos());
```

Fondamentaux de Node JS



Qu'est-ce que la fonction de rappel (Callback) ? 📞🔙

En JavaScript, les fonctions sont généralement exécutées de manière séquentielle. Une fonction s'exécute dans l'ordre dans lequel vous l'appelez, et non dans l'ordre dans lequel vous la définissez. Exemple 👉



Qu'est-ce que la fonction de rappel (Callback) ? 📞🔙

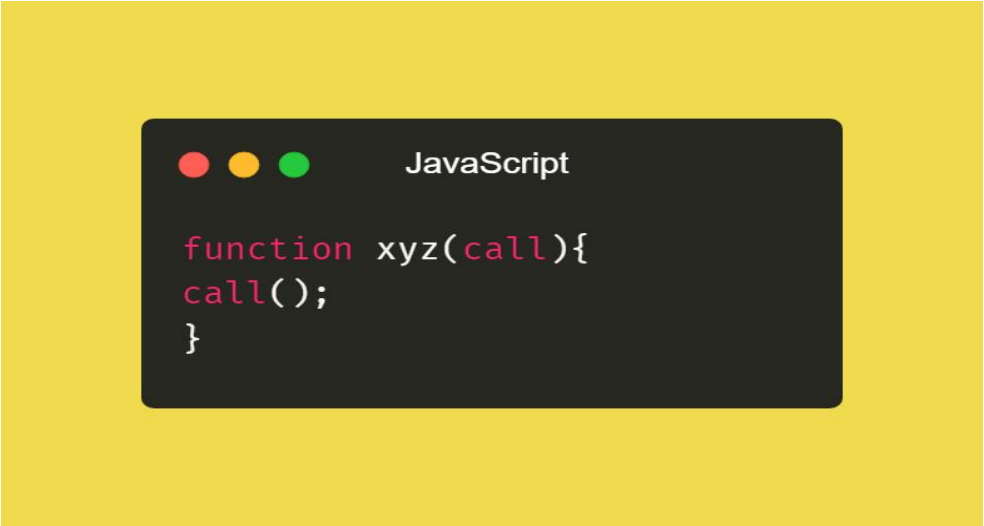
Par exemple, si nous utilisons la fonction '**setTimeout()**' sur la fonction **pet1**, **setTimeout()** est une fonction qui exécute une fonction après un temps donné. En attendant, JavaScript exécute d'autres fonctions, quelle que soit la séquence. Exemple 👉

JavaScript

```
function pet1(){
console.log("This is a cat");
}
function pet2(){
console.log("This is a dog.");
}
setTimeout(pet1, 5000);
pet2();

//output will be
This is a dog.
This is a cat.
```

Qu'est-ce que la fonction de rappel (Callback) ? 📞🔙

La fonction que nous passons en argument à une autre fonction s'appelle une fonction de rappel. Exemple 👇



Qu'est-ce que la fonction de rappel (Callback) ? 📞🔙

Contrôle de séquence

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2) {
  let sum = num1 + num2;
  return sum;
}

let result = myCalculator(5, 5);
myDisplayer(result);
```

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2) {
  let sum = num1 + num2;
  myDisplayer(sum);
}

myCalculator(5, 5);
```

Qu'est-ce que la fonction de rappel (Callback) ? 📞🔙

Contrôle de séquence

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2) {
  let sum = num1 + num2;
  return sum;
}

let result = myCalculator(5, 5);
myDisplayer(result);
```

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2) {
  let sum = num1 + num2;
  myDisplayer(sum);
}

myCalculator(5, 5);
```

Le problème avec le premier exemple ci-dessus, c'est que vous devez appeler deux fonctions pour afficher le résultat.
Le problème avec le deuxième exemple, c'est que vous ne pouvez pas empêcher la fonction myCalculator d'afficher le résultat.

Qu'est-ce que la fonction de rappel (Callback) ? 📞🔙

Contrôle de séquence avec 📞🔙

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2, Callback) {
  let sum = num1 + num2;
  Callback(sum);
}

myCalculator(5, 5, myDisplayer);
```

Dans l'exemple ci-dessus, **myDisplayer** est le nom d'une fonction. Il est passé à **myCalculator()** en tant qu'argument.

Lorsque vous passez une fonction en argument, n'oubliez pas de ne pas utiliser de parenthèses.

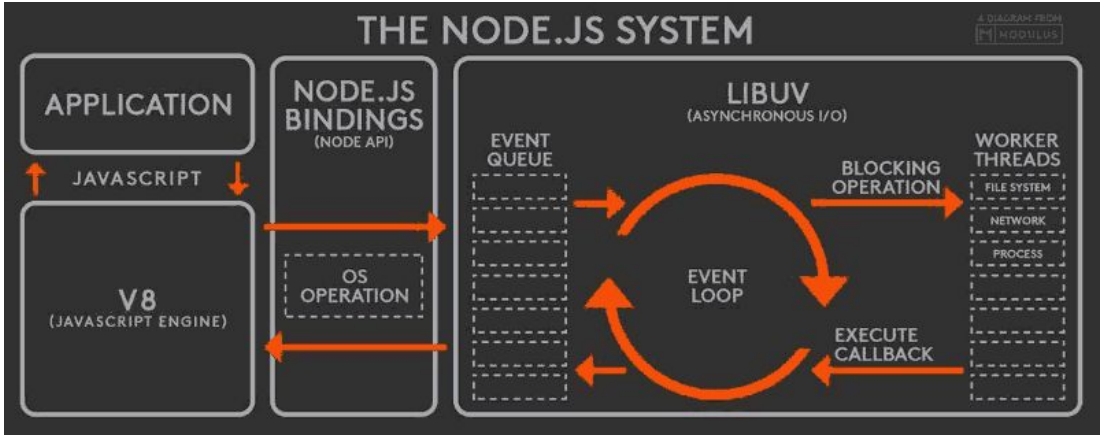
Correcte : myCalculator(5, 5, myDisplayer) ;
Faux : myCalculator(5, 5, myDisplayer());

Là où les rappels brillent vraiment, ce sont dans les fonctions asynchrones, où une fonction doit attendre une autre fonction (comme attendre le chargement d'un fichier).

Vue d'ensemble sur NodeJS Event Loop

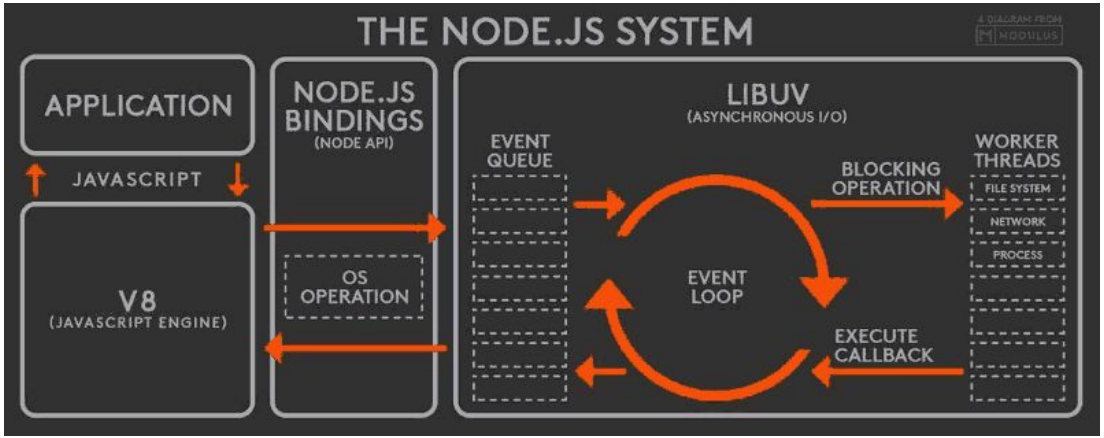
Q: Qu'est-ce que la boucle d'événement **Event Loop** ?

R: La boucle d'événements est ce qui permet à Node.js d'effectuer des opérations d'E/S non bloquantes - malgré le fait que JavaScript est à thread unique - en déchargeant les opérations sur le noyau du système chaque fois que possible.



Vue d'ensemble sur NodeJS Event Loop

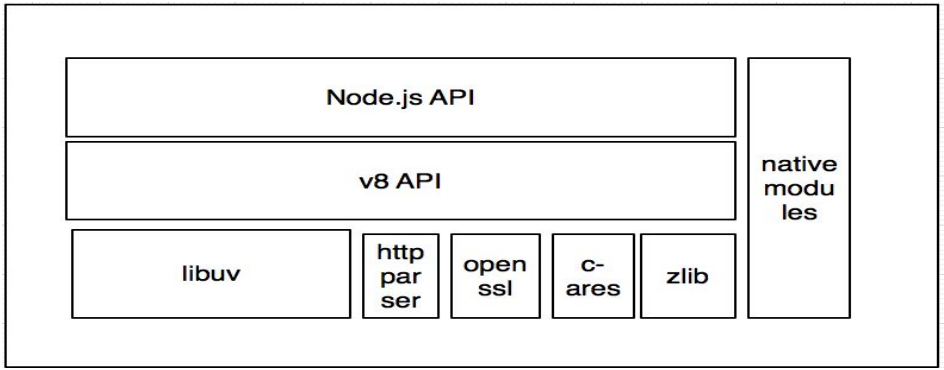
Chaque E / S nécessite un rappel - une fois qu'elles sont terminées, elles sont poussées dans la boucle d'événements pour exécution. Étant donné que la plupart des noyaux modernes sont multithreads, ils peuvent gérer plusieurs opérations exécutées en arrière-plan. Lorsque l'une de ces opérations est terminée, le noyau indique à Node.js que le rappel approprié peut être ajouté à la file d'attente d'interrogation pour être éventuellement exécuté.

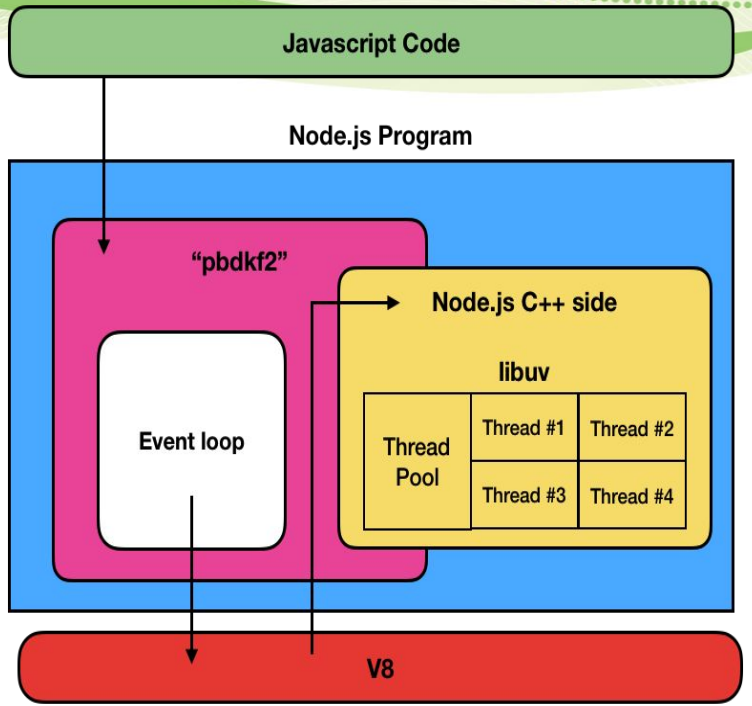


Vue d'ensemble sur NodeJS Event Loop

Q: Qu'est-ce que **libuv** ?

R: libuv est une bibliothèque C utilisée pour résumer les opérations d'E/S non bloquantes en une interface cohérente sur toutes les plates-formes prises en charge. libuv fournit des mécanismes pour gérer le système de fichiers, le DNS, le réseau, les processus enfants, la gestion du signal, et le streaming. Elle comprend également un pool de threads pour décharger le travail de certaines choses qui ne peuvent pas être effectuées de manière asynchrone au niveau du système d'exploitation.

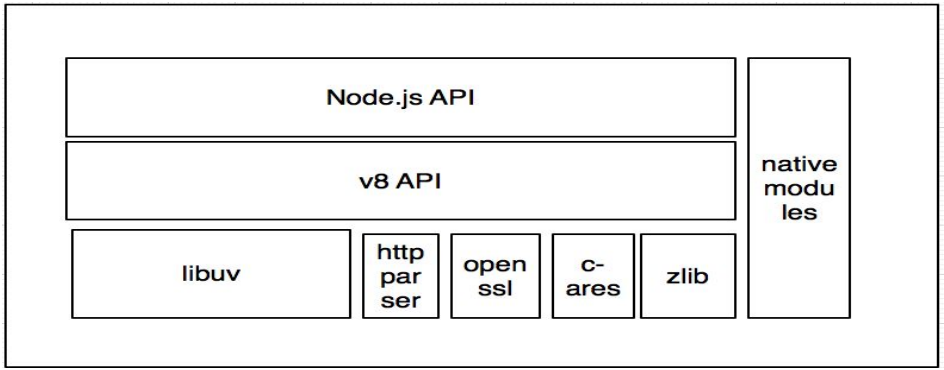


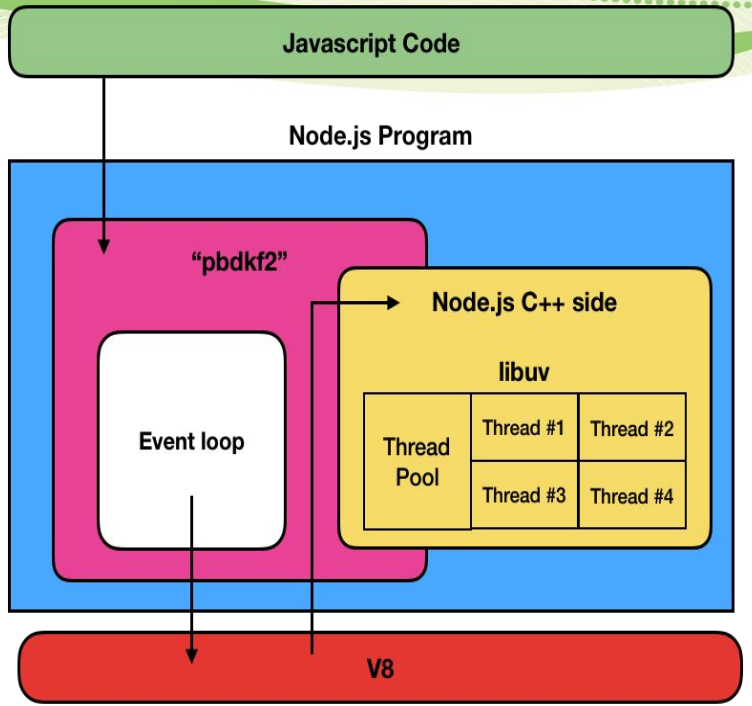


Vue d'ensemble sur NodeJS Event Loop

Q: Quelle est la différence entre **libuv** et **v8** ?

R: **V8** est responsable de l'exécution du code JavaScript en dehors du navigateur, tandis que **libuv** est responsable de l'accès au système d'exploitation et au système de fichiers sous-jacents de notre système et résout également une certaine mesure de la concurrence.

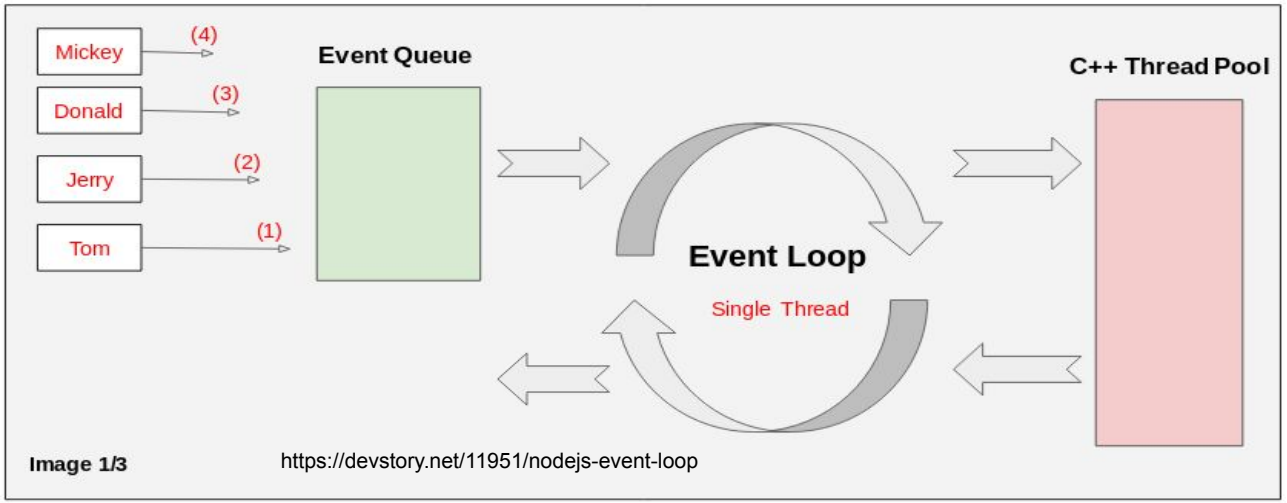




Vue d'ensemble sur NodeJS Event Loop

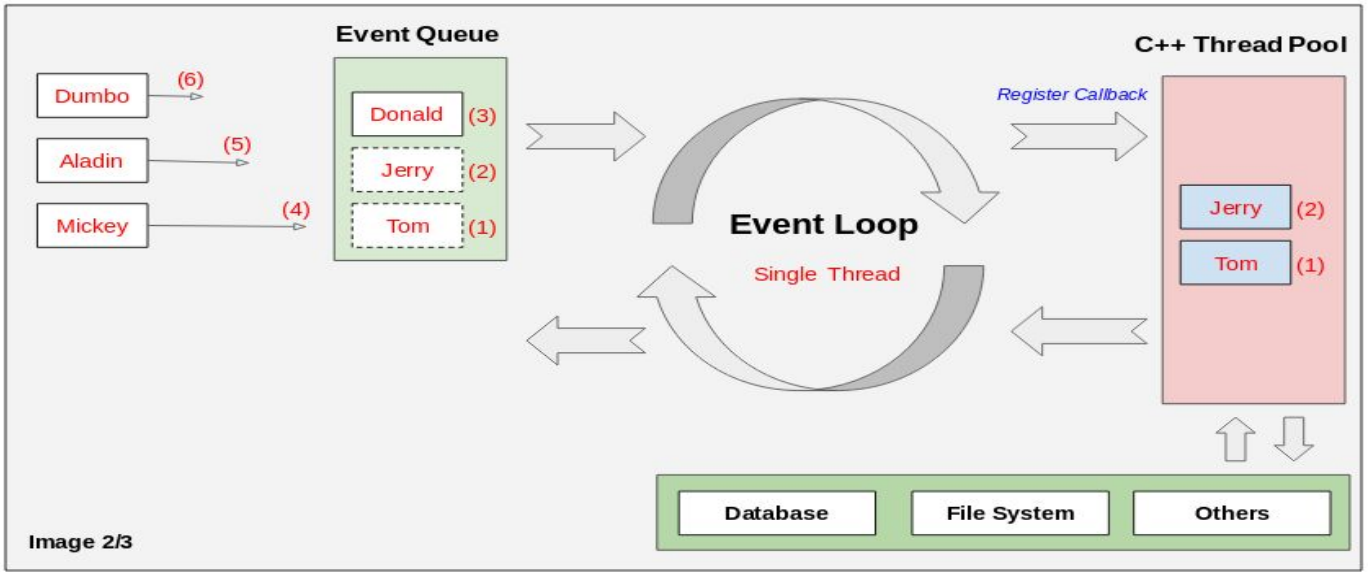
NodeJS est une application de thread unique (Single Thread), qui fonctionne sur une plateforme écrite en C++. Cette plateforme utilise le multi-thread pour effectuer des tâches en même temps.

La figure suivante illustre des demandes (request) envoyés à part des utilisateurs au serveur de NodeJS.



Vue d'ensemble sur NodeJS Event Loop

Chaque demande (request) de l'utilisateur est traitée comme un événement (event) par le NodeJS. Elles sont placées dans un Event Queue (La liste des événements). Le NodeJS utilise le principe FIFO (First In First Out), ce qui signifie que les premières demandes arrivant sera traités avant tout.

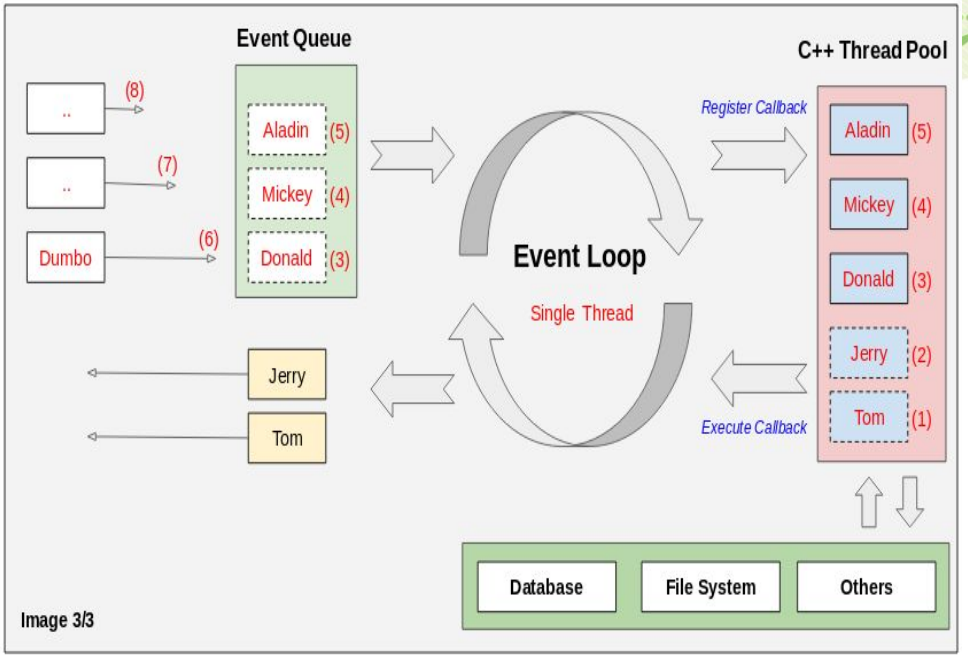


Vue d'ensemble sur NodeJS Event Loop

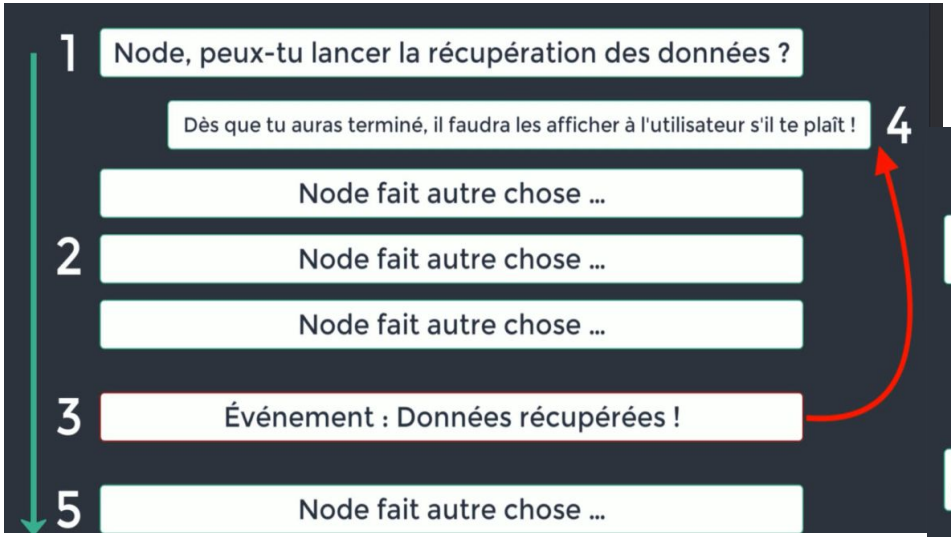
Comme une boucle infinie, elle transmet les demandes au Thread Pool (Pool de threads) et chaque requête est enregistrée une fonction Callback. Quand une requête est terminée, la fonction Callback correspondante sera appelée à être exécuté.

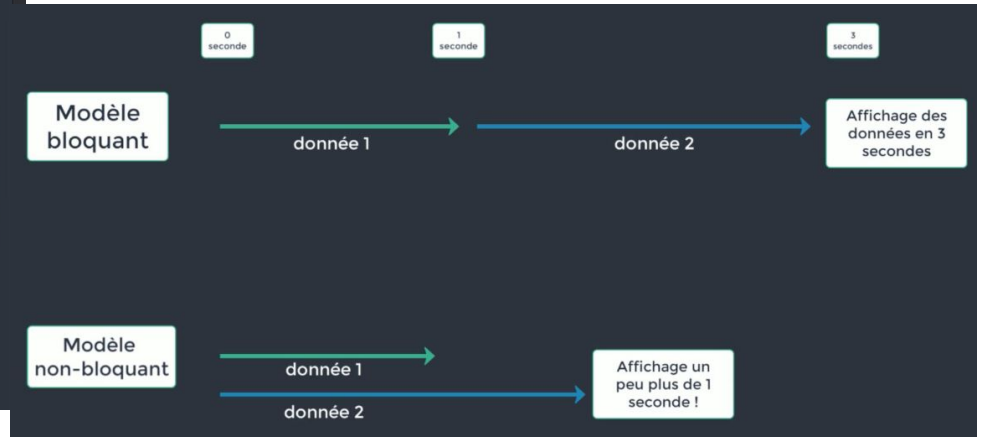
Lorsque le traitement d'une demande est terminé, le NodeJS appelera la fonction Callback (Enregistré pour cette demande) pour l'exécuter.

Démo : https://www.jsv9000.app/

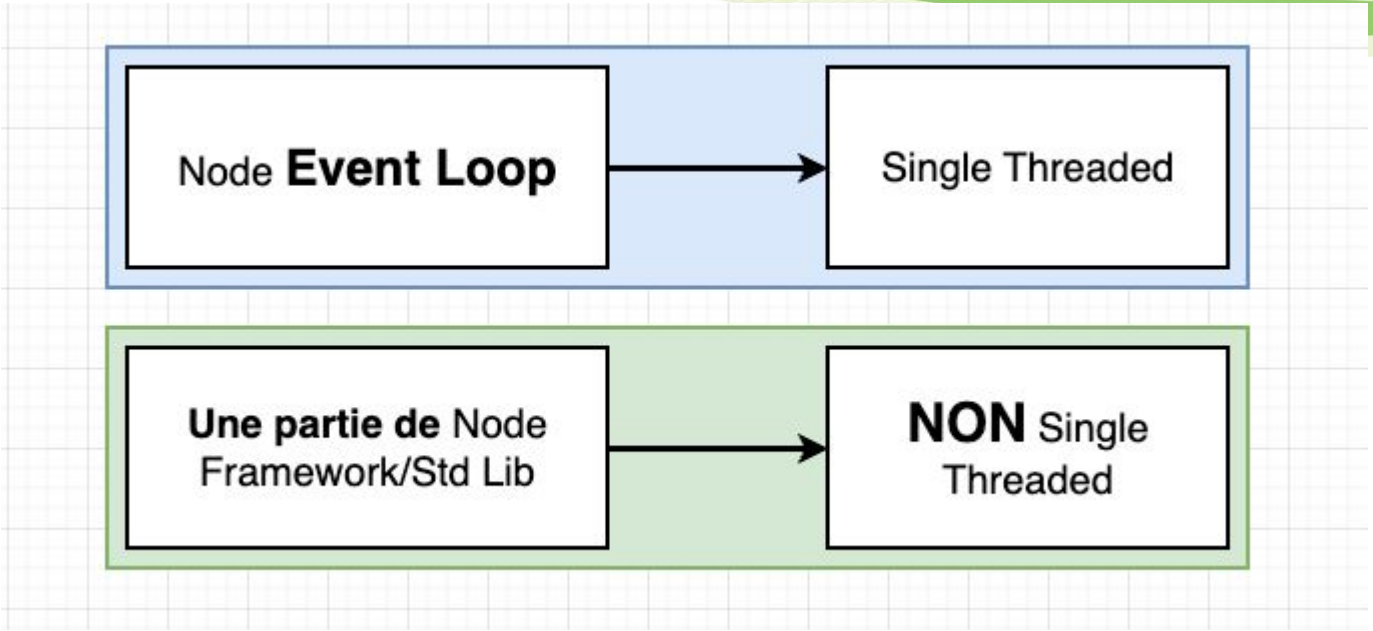


Vue d'ensemble sur NodeJS Event Loop



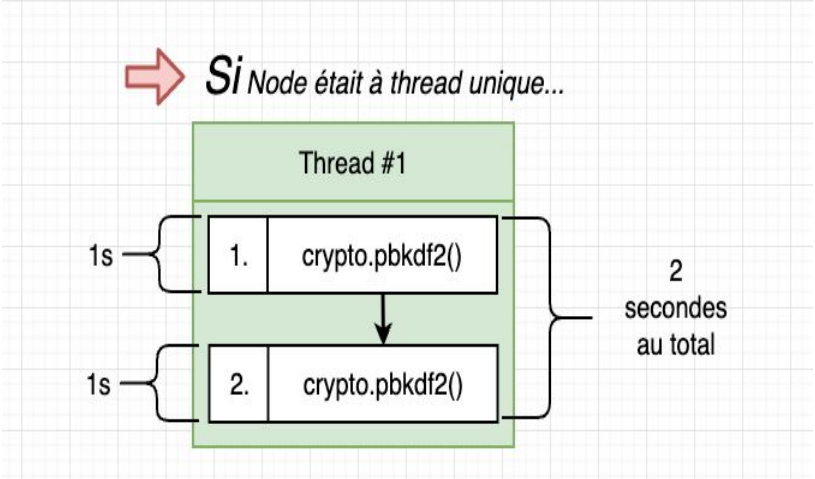


Vue d'ensemble sur NodeJS Event Loop



Vue d'ensemble sur NodeJS Event Loop

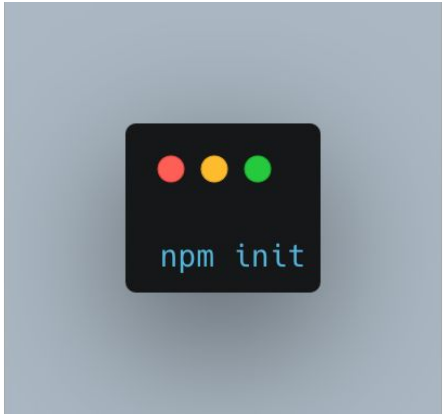
```
const crypto = require("crypto");

const start = Date.now();
crypto.pbkdf2('a','b',100000,512,'sha512', ()=>{
   console.log('1: ', Date.now()-start);
});

crypto.pbkdf2('a','b',100000,512,'sha512', ()=>{
    console.log('2: ', Date.now()-start);
});
```



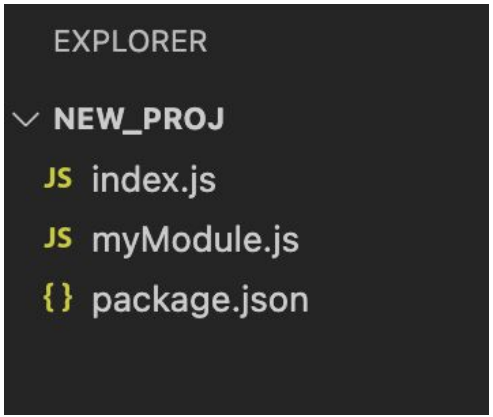
NodeJS Modules

Créer une application NodeJS

- Initialisez un projet Node.js : créez un nouveau répertoire pour votre projet et exécutez npm init dans votre terminal pour créer un fichier package.json.
- Créer un fichier principal : Créez un fichier nommé index.js.
- Créer un fichier de module : créez un fichier appelé myModule.js




```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE  COMMENTS

(base) josephazar@Josephs-MacBook-Pro-2 new_proj % nvm use 20
Now using node v20.14.0 (npm v10.7.0)
(base) josephazar@Josephs-MacBook-Pro-2 new_proj % npm --version
10.7.0
(base) josephazar@Josephs-MacBook-Pro-2 new_proj % node --version
v20.14.0
(base) josephazar@Josephs-MacBook-Pro-2 new_proj % npm init --y
Wrote to /Users/josephazar/Desktop/NodeJs/Web_Dev_2024_2025/new_proj/package.json:

{
  "name": "new_proj",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}



(base) josephazar@Josephs-MacBook-Pro-2 new_proj % touch index.js
(base) josephazar@Josephs-MacBook-Pro-2 new_proj % touch myModule.js
(base) josephazar@Josephs-MacBook-Pro-2 new_proj %
```



Utilisation des modules dans NodeJS

Pour installer un module à l'aide de NPM : *npm install chalk@4*

package.json

```
{
  "name": "new_proj",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "chalk": "^4.1.2"
  }
}
```

Utilisation des modules dans NodeJS

myModule.js

```
1   module.exports.title = "mon premier module JS";
2   module.exports.add = function (a,b){return a + b};
3   module.exports.subt = (a,b) => {return a-b};
```

index.js

```
1   const {title, add, subt} = require("./myModule");
2   const chalk = require("chalk");
3
4   console.log(chalk.red(title))
5   console.log(chalk.green(add(1,2)));
6   console.log(chalk.blue(subt(3,1)));
```

```
(base) josephazar@Josephs-MacBook-Pro-2 new_proj % node index.js
mon premier module JS
3
2
```

Installation de NodeJS & Codelab