**Problématiques**

Faciall est une plateforme permettant l’accès à des informations sur l’identité d’une personne grâce à l’intelligence artificielle, notamment la **reconnaissance faciale.**

Premièrement, les problématiques résolues par totalement ou partiellement **faciall** sont les suivants :

* L’usurpation d’identité
* La recherche de personne (tracking)
* Informations décentralisées

**1. L’usurpation d’identité**

L’usurpation d’identité peut être un problème pour beaucoup de cas, comme :

* La police (des personnes se faisant passer pour des éléments des forces de l’ordre)
* Les élections (des personnes usant de fausses identités pour pouvoir voter plusieurs fois ou pour voter à la place d’une autre personne)
* Sécurité d’un lieu (des personnes usant de pièce justificatif frauduleuse ou volée, par exemple un badge, pour pouvoir accéder à un lieu dont la visite est restreinte)
* Etc…

**2. La recherche de personne**

La recherche de personne est souvent fastidieux, pourtant grâce à la reconnaissance faciale il suffit juste qu’une caméra équipée de cette technologie capte la personne et on pourra déterminer où une personne se trouve.

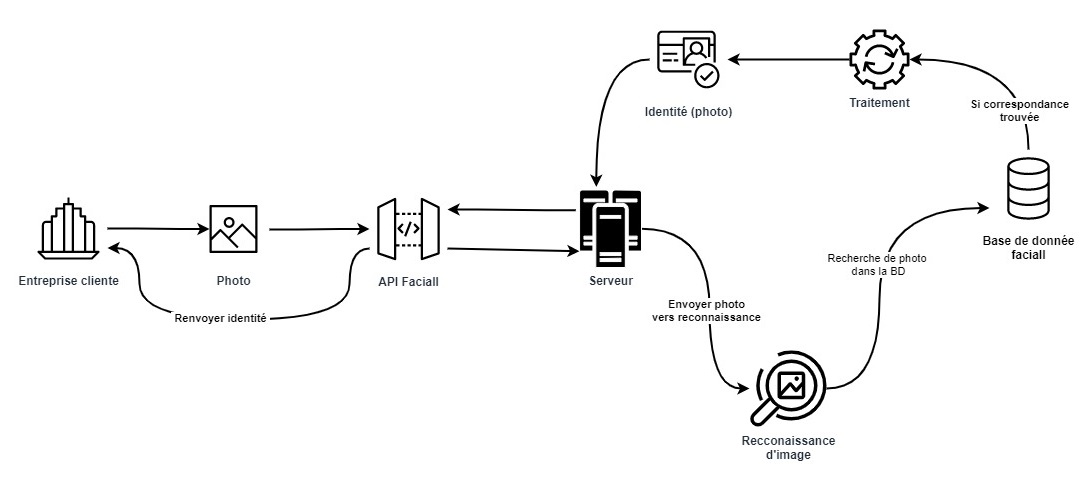
Le **tracking** dont on parle est le **traçage** des déplacements des personnes vues par une caméra de surveillance et utiliser cette information pour des usages futurs, comme des enquêtes criminels (pour servir d’alibi) ou bien pour retrouver une personne disparue et l’endroit où elle a été vue pour la dernière fois.

**3. Informations décentralisées**

Les informations décentralisées (nous parlons ici de la structure actuelle de la base de donnée à Madagascar et non de base de donnée informatique décentralisé en blockchain), l’état actuel de la base de donnée à Madagascar n’est pas encore centralisé dans une base de donnée informatique mais physique ou dans certains endroits numérisé mais pas mis en relation avec les autres bases de données. En d’autre termes, pour pouvoir se certifier par exemple à la commune, on a besoin de papier certifié depuis le quartier (certificat de résidence) il est encore impossible à Madagascar de venir à la commune, et que toutes nos informations soient tous directement disponibles depuis là-bas.

**Comment faciall résout ces problématiques ?**

Pour mieux comprendre les solutions apportées par **faciall** il est nécessaire de savoir comment la plateforme fonctionne et comprendre ses fonctionnalités. Elle fonctionne généralement ainsi ;



La plateforme est structurée ainsi, une entreprise cliente crée une requête en envoyant une photo d’une personne vers l’API de faciall, l’API étant relié au serveur celui-ci traite la photo en la réduisant pour une meilleure performance. Une fois réduite, l’image est envoyée dans le module de reconnaissance faciale, celui-ci calcule la description du visage dans la photo que l’on a formaté sous le format **JSON (JavaScript Object Notation)** ce qui réduit la taille d’une image venant du serveur étant communément entre 400Ko et 300Ko) à une chaine de caractère pesant entre 4Ko et 3Ko, en d’autre terme l’image a été réduit de 99% ce qui améliore considérablement les performances.

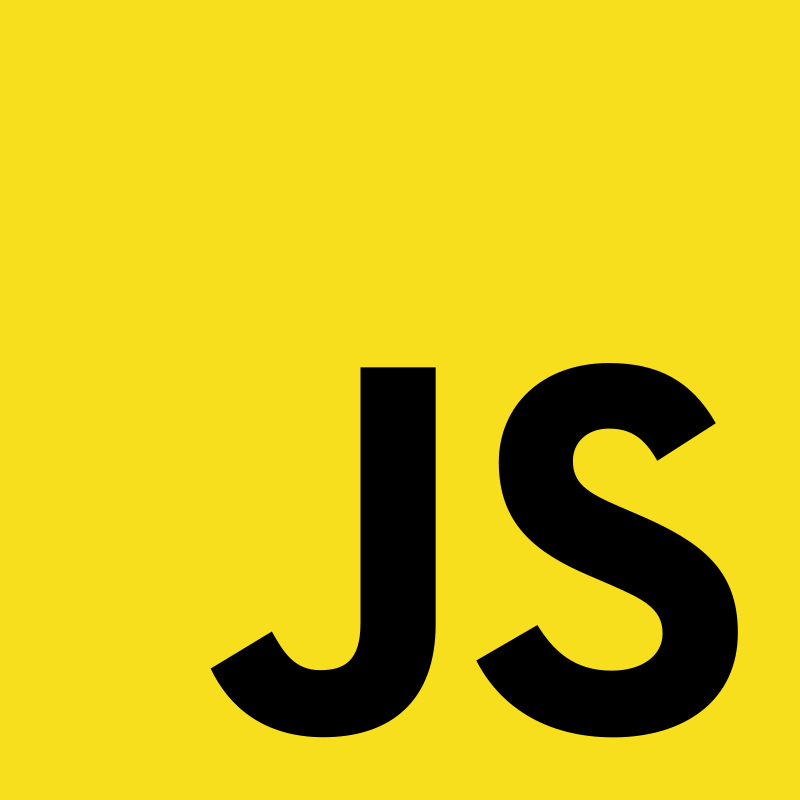
Une fois la description du visage terminée, ce même module procède à la recherche de la personne dans la base de donnée, la recherche se poursuit comme suit, la description de visage à rechercher est comparée aux descriptions faciales stockées dans la base de donnée. Mais en procédant ainsi la recherche pourrait prendre plus de temps d’où dans la description du visage calculé précédemment le genre et l’âge est aussi calculé ce qui nous permet de filtrer la base de donnée pour une recherche plus performante (la base de donnée filtre les éléments à recherche, si on lui demande des visages de femme il ne servira que des visage féminin). Si par exemple la personne à rechercher est un homme de 20 ans, tout d’abord on filtre le genre, c’est-à-dire que s’il y à 50% d’homme et 50% de femme dans la base de donnée l’on ne fera des comparaisons qu’avec 50% de personnes ce qui à priori réduit le temps de recherche de moitié. Par rapport à l’âge, certains visages ne correspondent pas vraiment à leurs âges, si un visage a 20 ans, il se peut qu’il appartient à une personne de 25ans ce qui est un problème, c’est là que la logique floue opère, il définit pour chaque âge de visage une plage d’âge et ainsi une quantité réduite de visage à comparer. Par exemple, si l’âge calculé est de 20 ans, la logique floue pourrait donner une plage de 15 à 25 ans par exemple et la base de donnée ne renverra plus que les hommes entre 15 et 25 ans, une fois le nombre de personne à comparer réduit, la recherche devient de plus en plus rapide.

Une fois une correspondance trouvée, on obtient l’identifiant de la personne recherchée, cette identifiant permettra par la suite de retirer les informations correspondant à la personne, ces informations seront renvoyées à l’entreprise qui a fait la requête et ce dernier pourra s’en servir.

Pour résoudre alors les problématiques, il faut comprendre que **faciall** est finalement une plateforme recueillant des informations exactes de personnes et proposant à des entreprises ou au gouvernement des accès à ces informations en utilisant la reconnaissance faciale, ces derniers peuvent avoir leurs propres applications ou logiciel mais ont une base de donnée « personne » commune, ils pourront aussi avoir leurs propres bases de données via leurs application ce qui n’entre pas en contact avec l’environnement de **faciall**.

**Les technologies utilisées**

**I. JavaScript**



**JavaScript** est un langage de programmation de scripts principalement employé dans les pages web interactives et à ce titre est une partie essentielle des applications web. Avec les langages HTML et CSS, JavaScript est au cœur des langages utilisés par les développeurs web. Une grande majorité des sites web l’utilisent, et la majorité des navigateurs web disposent d’un moteur JavaScript pour l’interpréter.

JavaScript est aussi employé pour le **back end** ou coté serveur avec l'utilisation de **Deno** ou aussi de[**Node.js**](https://fr.wikipedia.org/wiki/Node.js)dont nous parlerons un peu plus tard.

JavaScript a été créé en 1995 par [Brendan Eich](https://fr.wikipedia.org/wiki/Brendan_Eich) et intégré au [navigateur web](https://fr.wikipedia.org/wiki/Navigateur_web) [Netscape Navigator](https://fr.wikipedia.org/wiki/Netscape_Navigator) 2.0. L'implémentation concurrente de JavaScript par [Microsoft](https://fr.wikipedia.org/wiki/Microsoft) dans [Internet Explorer](https://fr.wikipedia.org/wiki/Internet_Explorer) jusqu'à sa version 9 se nommait **[JScript](https://fr.wikipedia.org/wiki/JScript" \o "JScript)**, tandis que celle d'[Adobe System](https://fr.wikipedia.org/wiki/Adobe_Systems) se nommait **[ActionScript](https://fr.wikipedia.org/wiki/ActionScript" \o "ActionScript)**. **JavaScript** a été standardisé sous le nom **d'[ECMAScript](https://fr.wikipedia.org/wiki/ECMAScript" \o "ECMAScript)** en juin 1997 par **[Ecma International](https://fr.wikipedia.org/wiki/Ecma_International" \o "Ecma International)** dans le standard **ECMA-262**. La version en vigueur de ce standard depuis juin 2020 est l’édition numéro 11.

C'est un langage [orienté objet](https://fr.wikipedia.org/wiki/Orient%C3%A9_objet) à [prototype](https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_prototype) : les bases du langage et ses principales [interfaces](https://fr.wikipedia.org/wiki/Interface_de_programmation) sont fournies par des [objets](https://fr.wikipedia.org/wiki/Objet_(informatique)). Cependant, à la différence d'un langage orienté objets, les objets de base ne sont pas des [instances](https://fr.wikipedia.org/wiki/Instance_(programmation)) de [classes](https://fr.wikipedia.org/wiki/Classe_(informatique)). En outre, les [fonctions](https://fr.wikipedia.org/wiki/Fonction_informatique) sont des [objets de première classe](https://fr.wikipedia.org/wiki/Objet_de_premi%C3%A8re_classe). Il supporte le [paradigme](https://fr.wikipedia.org/wiki/Paradigme_de_programmation) [objet](https://fr.wikipedia.org/wiki/Programmation_objet), [impératif](https://fr.wikipedia.org/wiki/Programmation_imp%C3%A9rative) et [fonctionnel](https://fr.wikipedia.org/wiki/Programmation_fonctionnelle).

JavaScript est le langage possédant le plus large [écosystème](https://fr.wikipedia.org/wiki/%C3%89cosyst%C3%A8me) notamment grâce à son gestionnaire de dépendances **[npm](https://fr.wikipedia.org/wiki/Npm" \o "Npm)**, avec environ 500 000 paquets en août 2017.

**1. Utilisation dans une page web**

Le propos de JavaScript est de manipuler de façon simple des [objets](https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_objet), au sens informatique, fournis par une application hôte. Par exemple dans un navigateur web, un script écrit en **JavaScript** peut être utilisé pour apporter une touche interactive ou dynamique à un applicatif (page ou site web), qui sans cela serait une page statique figée. Le langage **JavaScript** permet par exemple d'écrire des scripts pour afficher ou cacher un paragraphe, une image, selon les interactions de l'utilisateur, ou d'informer le serveur du temps passé à lire une page.

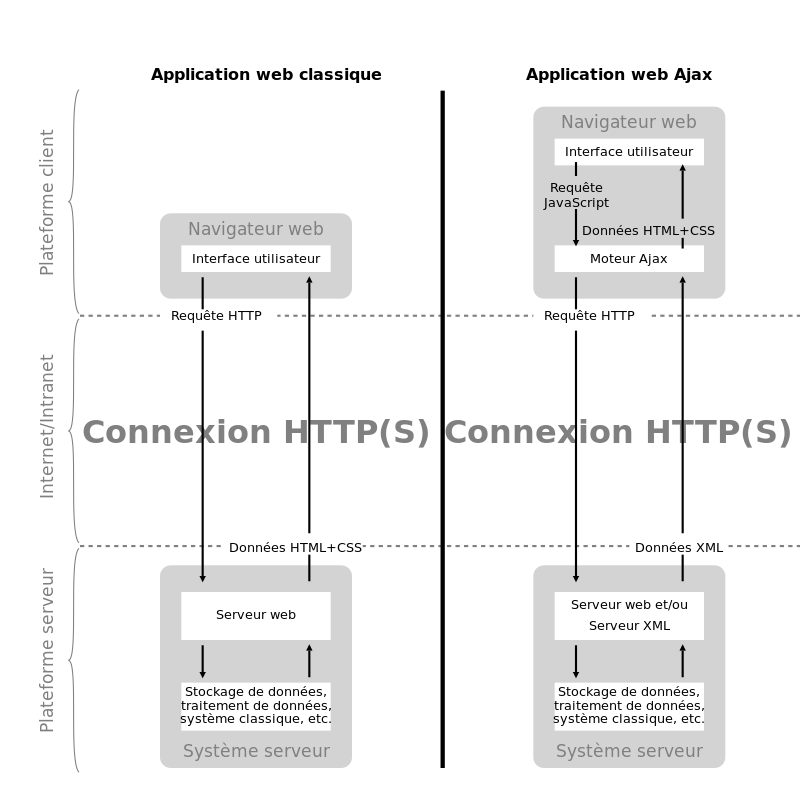
Du code **JavaScript** peut être intégré directement au sein des [pages web](https://fr.wikipedia.org/wiki/Page_web), pour y être exécuté sur le poste [client](https://fr.wikipedia.org/wiki/Client-serveur). C'est alors le [navigateur web](https://fr.wikipedia.org/wiki/Navigateur_web) qui prend en charge l'exécution de ces programmes appelés scripts.

Généralement, JavaScript sert à contrôler les données saisies dans des formulaires [HTML](https://fr.wikipedia.org/wiki/Hypertext_Markup_Language), ou à interagir avec le document HTML via l'interface [Document Object Model](https://fr.wikipedia.org/wiki/Document_Object_Model), fournie par le navigateur (on parle alors parfois de [HTML dynamique](https://fr.wikipedia.org/wiki/DHTML) ou DHTML). Il est aussi utilisé pour réaliser des applications dynamiques, des transitions, des animations ou manipuler des données réactives, à des fins ergonomiques ou cosmétiques.

JavaScript n'est pas limité à la manipulation de documents HTML et peut aussi servir à manipuler des documents [SVG](https://fr.wikipedia.org/wiki/SVG), [XUL](https://fr.wikipedia.org/wiki/XUL) ou même du XML.

**2. Utilisation d’Ajax**

[**Ajax**](https://fr.wikipedia.org/wiki/Ajax_(informatique)) (de l'anglais Asynchronous JavaScript And XML) est un ensemble de techniques découplant l'échange de données entre le navigateur et le serveur web de l'affichage d'une page web, ce qui permet de modifier le contenu des pages web sans les recharger. Grâce à l'objet JavaScript **[XMLHTTPRequest](https://fr.wikipedia.org/wiki/XMLHTTPRequest)**, cette méthode permet d'effectuer des requêtes **HTTP** sur le serveur web depuis le navigateur web, et permet également de traiter les réponses [**HTTP**](https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol) du serveur web pour modifier le contenu de la page web. La réponse était en général au format **XML** qui tend aujourd'hui à être remplacé par le format [**JSON**](https://fr.wikipedia.org/wiki/JavaScript_Object_Notation) qui a l'avantage d'être natif en JavaScript. Le script manipule l'ensemble d'objets [**DOM**](https://fr.wikipedia.org/wiki/Document_Object_Model) qui représente le contenu de la page web. Les technologies **[XMLHTTPRequest](https://fr.wikipedia.org/wiki/XMLHTTPRequest" \o "XMLHTTPRequest)**, [**XML**](https://fr.wikipedia.org/wiki/XML) et [**DOM**](https://fr.wikipedia.org/wiki/Document_Object_Model) ont été ajoutées aux navigateurs web entre 1995 et 2005. La méthode **Ajax** permet de réaliser des [applications Internet riches](https://fr.wikipedia.org/wiki/Web_2.0#Application_Internet_riche), offrant une maniabilité et un confort supérieur, c'est un des sujets phares du mouvement [**Web 2.0**](https://fr.wikipedia.org/wiki/Web_2.0).



Dans une application Web, la méthode classique de dialogue entre un [navigateur](https://fr.wikipedia.org/wiki/Navigateur_web) et un [serveur](https://fr.wikipedia.org/wiki/Serveur_web) est la suivante : lors de chaque manipulation faite par l'utilisateur, le navigateur envoie une [requête](https://fr.wiktionary.org/wiki/requ%C3%AAte) contenant une référence à une [page Web](https://fr.wikipedia.org/wiki/Page_Web), puis le serveur Web effectue des calculs, et envoie le résultat sous forme d'une page Web à destination du navigateur. Celui-ci affichera alors la page qu'il vient de recevoir. Chaque manipulation entraîne la transmission et l'affichage d'une nouvelle page. L'utilisateur doit attendre l'arrivée de la réponse pour effectuer d'autres manipulations.

En utilisant **Ajax**, le dialogue entre le navigateur et le serveur se déroule la plupart du temps de la manière suivante : un programme écrit en langage de programmation [**JavaScript**](https://fr.wikipedia.org/wiki/JavaScript), incorporé dans une page web, est exécuté par le navigateur. Celui-ci envoie en arrière-plan des demandes au serveur Web, puis modifie le contenu de la page actuellement affichée par le navigateur Web en fonction du résultat reçu du serveur, évitant ainsi la transmission et l'affichage d'une nouvelle page complète.

La méthode classique de dialogue utilise des mécanismes propres au [World Wide Web](https://fr.wikipedia.org/wiki/World_Wide_Web), qui sont incorporés dans tous les navigateurs, et ne nécessite pas de programmation. Au contraire, le fonctionnement d'**Ajax** nécessite de programmer en **JavaScript** les échanges entre le navigateur et le serveur Web. Il nécessite également de programmer les modifications à effectuer dans la page Web à la réception des réponses, sans quoi les dialogues se font à l'insu de l'utilisateur.

En **Ajax**, comme le nom l'indique, les demandes sont effectuées de manière [**asynchrone**](https://fr.wikipedia.org/wiki/Asynchronisme) : le navigateur Web continue d'exécuter le programme **JavaScript** alors que la demande est partie, il n'attend pas la réponse envoyée par le serveur Web et l'utilisateur peut continuer à effectuer des manipulations pendant ce temps.

**2. Utilisation de JSON**

[**JSON**](https://fr.wikipedia.org/wiki/JavaScript_Object_Notation)(*JavaScript Object Notation*) est un format utilisant la notation des objets JavaScript pour transmettre de l'information structurée, d'une façon plus compacte et plus proche des [langages de programmation](https://fr.wikipedia.org/wiki/Langages_de_programmation), que **XML**.

Malgré l'existence du DOM et l'introduction récente de E4X dans la spécification du langage **JavaScript**, **JSON** reste le moyen le plus simple d'accéder à des données, puisque chaque flux **JSON** n'est rien d'autre qu'un objet **JavaScript** sérialisé. De plus, **JSON** reste un format de données structurées, et peut être utilisé facilement par tous les langages de programmation.

Depuis 2009, les navigateurs commencent à intégrer un support natif du format **JSON**, ce qui facilite sa manipulation, la sécurité (contre l'évaluation de scripts malveillants inclus dans une chaine **JSON**), et la rapidité de traitement.

Exemple de **JSON** :

{

**"clef1"**: "valeur",

**"clef2"**: 12345,

**"clef3"**: **true**,

**"clef4”:** {**"clef5”:** "valeur"}

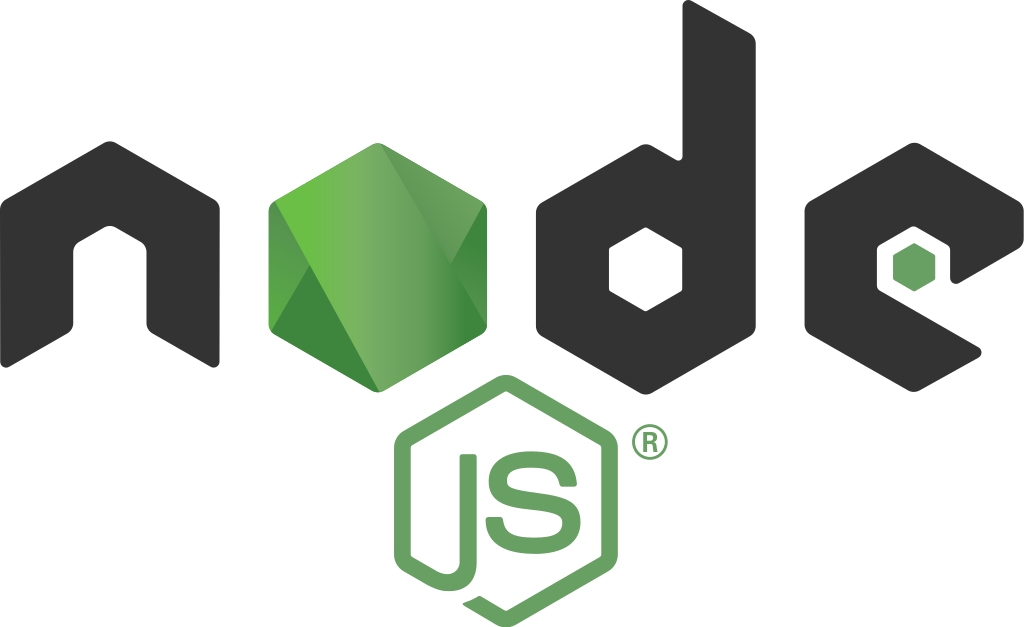
}

**3. Utilisation sur un serveur**

**JavaScript** peut également être utilisé comme langage de programmation sur un [**serveur HTTP**](https://fr.wikipedia.org/wiki/Serveur_HTTP) à l'image des langages comme [**PHP**](https://fr.wikipedia.org/wiki/PHP)**,**[**ASP**](https://fr.wikipedia.org/wiki/Active_Server_Pages)**, etc**. D'ailleurs le projet [**CommonJS**](https://fr.wikipedia.org/wiki/CommonJS) travaille dans le but de spécifier un écosystème pour **JavaScript** en dehors du navigateur (par exemple sur le serveur ou pour les applications de bureau natives). Le projet a été lancé par **Kevin Dangoor** en janvier 2009. Le projet **CommonJS** n'est pas affilié avec le groupe de l'Ecma International TC39 travaillant sur ECMAScript, mais certains membres du TC39 participent au projet.

Il existe par ailleurs des projets indépendants et Open Source d'implémentation de serveurs en **JavaScript**. Parmi eux, on pourra distinguer[**Node.js**](https://fr.wikipedia.org/wiki/Node.js) que l’on a utilisé dans ce projet, une plateforme polyvalente de développement d'applications réseau se basant sur le moteur **JavaScript** V8 de Google Chrome et les spécifications [**CommonJS**](https://fr.wikipedia.org/wiki/CommonJS).

**II. Node.JS**



**Node.js** est une [plateforme logicielle](https://fr.wikipedia.org/wiki/Plate-forme_(informatique)) [libre](https://fr.wikipedia.org/wiki/Logiciel_libre) en [**JavaScript**](https://fr.wikipedia.org/wiki/JavaScript), orientée **back end** c’est-à-dire côté serveur vers les applications [réseau](https://fr.wikipedia.org/wiki/R%C3%A9seau_informatique) [évènementielles](https://fr.wikipedia.org/wiki/Programmation_%C3%A9v%C3%A9nementielle)  qui doivent pouvoir [monter en charge](https://fr.wikipedia.org/wiki/Scalability).

Elle utilise la [machine virtuelle](https://fr.wikipedia.org/wiki/Machine_virtuelle) [**V8**](https://fr.wikipedia.org/wiki/V8_(moteur_JavaScript)) de **Google Chrome**, la librairie [**libuv**](https://en.wikipedia.org/wiki/libuv) pour sa [boucle d'évènements](https://en.wikipedia.org/wiki/Event_loop), et implémente sous [**licence MIT**](https://fr.wikipedia.org/wiki/Licence_MIT) les spécifications [**CommonJS**](https://fr.wikipedia.org/wiki/CommonJS).

Parmi les modules natifs de **Node.js**, on retrouve [**http**](https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol) qui permet le développement de [**serveur** **HTTP**](https://fr.wikipedia.org/wiki/Serveur_HTTP). Ce qui autorise, lors du déploiement de sites internet et d'applications web développés avec **Node.js**, de ne pas installer et utiliser des [serveurs web](https://fr.wikipedia.org/wiki/Serveur_informatique) tels que **[Nginx](https://fr.wikipedia.org/wiki/Nginx" \o "Nginx)** ou [**Apache**](https://fr.wikipedia.org/wiki/Apache_HTTP_Server). Concrètement, **Node.js** est un environnement bas niveau permettant l’exécution de [**JavaScript**](https://fr.wikipedia.org/wiki/JavaScript) côté serveur.

**Node.js** est utilisé notamment comme plateforme de serveur Web, elle est utilisée par GoDaddy, IBM, Netflix, Amazon Services, Vivaldi, SAP, LinkedIn, Microsoft, Yahoo, Walmart, Rakuten, Sage et même PayPal.

En tant qu'environnement d'exécution **JavaScript** asynchrone et orienté événement, **Node.js** est conçu pour générer des applications extensibles. Dans cet exemple ("hello world"), plusieurs connexions peuvent être gérées de manière concurrente. À chaque connexion, la fonction de rappel (callback function) est déclenchée, mais s’il n'y a rien à faire, Node.js restera inactif.

const http = require('http');

const hostname = '127.0.0.1';

const port = 3000;

const server = http.createServer((req, res) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Hello World');

});

server.listen(port, hostname, () => {

console.log(`Server running at http://${hostname}:${port}/`);

});

Ceci contraste avec le modèle de concurrence plus commun dans lequel les processus système sont utilisés. La gestion réseau basée sur les processus est relativement inefficace et difficile à utiliser. De plus, les utilisateurs de **Node.js** n'ont pas à se soucier des problèmes d'inter blocage des processus puisqu'il n'y a pas de verrouillage. Aucune fonction de **Node.js** ou presque n'effectue d'entrée/sortie, donc le processus ne se bloque pas. Et comme rien n'est bloquant, développer un système extensible est relativement aisé avec **Node.js**.

Lorsque **Node.js** démarre, il initialise la boucle d'événements, traite le script d'entrée fourni qui peut effectuer des appels d'API asynchrones, programmer des minuteries ou appeler **process.nextTick()**, commence alors le traitement de la boucle d'événements.

┌───────────────────────────┐

┌─>│ timers │

│ └─────────────┬─────────────┘

│ ┌─────────────┴─────────────┐

│ │ pending callbacks │

│ └─────────────┬─────────────┘

│ ┌─────────────┴─────────────┐

│ │ idle, prepare │

│ └─────────────┬─────────────┘ ┌───────────────┐

│ ┌─────────────┴─────────────┐ │ incoming : │

│ │ poll │<─────┤ connections, │

│ └─────────────┬─────────────┘ │ data, etc. │

│ ┌─────────────┴─────────────┐ └───────────────┘

│ │ check │

│ └─────────────┬─────────────┘

│ ┌─────────────┴─────────────┐

└──┤ close callbacks │

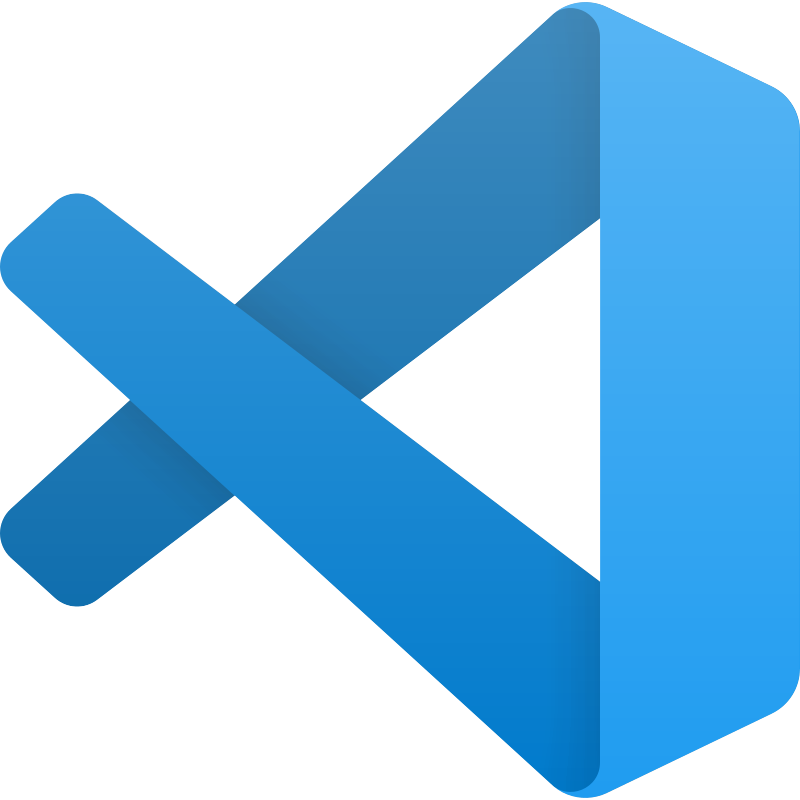
└───────────────────────────┘

Chaque case sera appelée une "phase" de la boucle d'événements.

Chaque phase a une file d'attente FIFO de rappels à exécuter. Bien que chaque phase soit spéciale à sa manière, généralement, lorsque la boucle d'événements entre dans une phase donnée, elle effectuera toutes les opérations spécifiques à cette phase, puis exécutera des rappels dans la file d'attente de cette phase jusqu'à ce que la file d'attente soit épuisée ou que le nombre maximal de rappels a exécuté. Lorsque la file d'attente est épuisée ou que la limite de rappel est atteinte, la boucle d'événements passe à la phase suivante, et ainsi de suite.

Étant donné que chacune de ces opérations peut planifier davantage d'opérations et que les nouveaux événements traités dans la phase d'interrogation sont mis en file d'attente par le noyau, les événements d'interrogation peuvent être mis en file d'attente pendant le traitement des événements d'interrogation. Par conséquent, les rappels de longue durée peuvent permettre à la phase d'interrogation de s'exécuter beaucoup plus longtemps que le seuil d'un temporisateur.

**III. Visual Studio Code**



**Visual Studio Code** est un [éditeur de code](https://fr.wikipedia.org/wiki/%C3%89diteur_de_texte) extensible développé par Microsoft pour Windows, Linux et MacOS.

Les fonctionnalités incluent la prise en charge du [débogage](https://fr.wikipedia.org/wiki/D%C3%A9bogage), la [mise en évidence de la syntaxe](https://fr.wikipedia.org/wiki/Coloration_syntaxique), la complétion intelligente du code, les [snippets](https://fr.wikipedia.org/wiki/Snippet), la [refactorisation](https://fr.wikipedia.org/wiki/Refactorisation) du code et [Git](https://fr.wikipedia.org/wiki/Git) intégré. Les utilisateurs peuvent modifier le [thème](https://en.wikipedia.org/wiki/Theme_(computing)), les [raccourcis clavier](https://fr.wikipedia.org/wiki/Raccourci_clavier), les préférences et installer des [extensions](https://en.wikipedia.org/wiki/Plug-in_(computing)) qui ajoutent des fonctionnalités supplémentaires.

Le [code source](https://fr.wikipedia.org/wiki/Code_source) de Visual Studio Code provient du projet logiciel [libre](https://fr.wikipedia.org/wiki/Logiciel_libre) et [open source](https://fr.wikipedia.org/wiki/Open_source) VSCode de Microsoft publié sous la [licence MIT](https://fr.wikipedia.org/wiki/Licence_MIT) permissive, mais les binaires compilés constituent un [freeware](https://fr.wikipedia.org/wiki/Freeware), c'est-à-dire un logiciel gratuit pour toute utilisation mais [privateur](https://fr.wikipedia.org/wiki/Logiciel_privateur" \o "Logiciel privateur).

Dans le **[Stack Overflow](https://fr.wikipedia.org/wiki/Stack_Overflow" \o "Stack Overflow) 2019 Developer Survey**, Visual Studio Code a été classé comme l'[outil d'environnement de développement](https://fr.wikipedia.org/wiki/Environnement_de_d%C3%A9veloppement) le plus populaire, avec 50,7 % des 87 317 répondants déclarant l'utiliser.

**IV. MongoDB**



**MongoDB** (de l'anglais *[humongous](https://fr.wiktionary.org/wiki/humongous" \o "wikt:humongous)* qui peut être traduit par « énorme ») est un [système de gestion de base de données](https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_gestion_de_base_de_donn%C3%A9es) [orienté documents](https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es_orient%C3%A9e_documents), [répartissable sur un nombre quelconque d'ordinateurs](https://fr.wikipedia.org/wiki/Scalability) et ne nécessitant pas de schéma prédéfini des données. Il est écrit en [**C++**](https://fr.wikipedia.org/wiki/C%2B%2B)**.** Le serveur et les outils sont distribués sous [licence **SSPL**](https://fr.wikipedia.org/wiki/Server_Side_Public_License), les pilotes sous [licence Apache](https://fr.wikipedia.org/wiki/Licence_Apache) et la documentation sous [licence **Creative Commons**](https://fr.wikipedia.org/wiki/Licence_Creative_Commons). Il fait partie de la mouvance [**NoSQL**](https://fr.wikipedia.org/wiki/NoSQL).

**1. Données manipulées**

**MongoDB** permet de manipuler des objets structurés au format [**BSON**](https://fr.wikipedia.org/wiki/BSON) ([JSON](https://fr.wikipedia.org/wiki/JSON) [binaire](https://fr.wikipedia.org/wiki/Fichier_binaire)), sans schéma prédéterminé. En d'autres termes, des clés peuvent être ajoutées à tout moment « à la volée », sans reconfiguration de la base.

Les données prennent la forme de documents enregistrés eux-mêmes dans des collections, une collection contenant un nombre quelconque de documents. Les collections sont comparables aux [tables](https://fr.wikipedia.org/wiki/Table_(base_de_donn%C3%A9es)), et les documents aux [enregistrements](https://fr.wikipedia.org/wiki/Enregistrement_(base_de_donn%C3%A9es)) des bases de données relationnelles. Contrairement aux bases de données relationnelles, les [champs](https://fr.wikipedia.org/wiki/Champ_(base_de_donn%C3%A9es)) d'un enregistrement sont libres et peuvent être différents d'un enregistrement à un autre au sein d'une même collection. Le seul champ commun et obligatoire est le champ de clé principale ("id"). Par ailleurs, **MongoDB** ne permet ni les requêtes très complexes standardisées, ni les JOIN, mais permet de programmer des requêtes spécifiques en [**JavaScript**](https://fr.wikipedia.org/wiki/JavaScript).

Pour des données mises en page de la manière suivante :

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Nom** | **Prénom** | **Age** |
| 7 | RAKOTONIMANANA | Andritiana | 20 |
| 8 | RAKOTOARIVELO | Miandry | *22* |
| 9 | ANDRIAMASIMANANA | Ny Hasina | NULL |

On remarque qu'une valeur peut être manquante (*NULL*), mais que chaque ligne comporte le même nombre de champs.

Différemment, pour lamême mise en page une collection **NoSQL** peut présenter des champs différents à chaque ligne (et même imbriquer plusieurs champs dans une ligne) :

{

"\_id": ObjectId("4efa8d2b7d284dad101e4bc7"),

"Nom": "RAKOTONIMANANA",

"Prénom": "Andritiana",

"Age": 20

},

{

"\_id": ObjectId("4efa8d2b7d284dad101e4bc8"),

"Nom" : "RAKOTOARIVELO",

"Prénom" : "Miandry",

"Adresse" : "Ivandry",

"Ville" : "Antananarivo"

},

{

"\_id": ObjectId("4efa8d2b7d284dad101e4bc9"),

"Nom": "ANDRIAMASIMANANA",

"Âge" : "NULL",

"Adresse" : "Ambohimangakely",

"Ville" : "Antananarivo"

}

On remarque tout de suite que des champs nouveaux ont été ajoutés dans les enregistrements les plus récents, sans impact sur l'enregistrement le plus ancien. C'est ce qui fait la spécificité des **NoSQL**. On remarque également que ce stockage sous forme « attribut="valeur" » est très proche du [**XML**](https://fr.wikipedia.org/wiki/Extensible_Markup_Language). Par ailleurs, une sauvegarde de base de données **(dump)** en langage de requête **SQL** peut également prendre cette forme « attribut="valeur" », mais elle n'est exploitable que recréée sous forme de tables (dans un [**SGBD**](https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_gestion_de_base_de_donn%C3%A9es)).

Les documents d'une collection **MongoDB** peuvent comporter des [champs](https://fr.wikipedia.org/wiki/Champ_(base_de_donn%C3%A9es)) différents (le champ "\_id" est un champ obligatoire, généré et ajouté par **MongoDB**, c'est un index unique qui permet d'identifier le document).

Dans un document, des [champs](https://fr.wikipedia.org/wiki/Champ_(base_de_donn%C3%A9es)) peuvent être ajoutés, supprimés, modifiés et renommés à tout moment. Contrairement aux [bases de données relationnelles](https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es_relationnelle), il n'y a pas de schéma prédéfini. La structure d'un document est très simple et se compose de paires clef/valeur à la manière des [tableaux associatifs](https://fr.wikipedia.org/wiki/Tableau_associatif), la clef est le nom du [champ](https://fr.wikipedia.org/wiki/Champ_(base_de_donn%C3%A9es)), la valeur son contenu. Les deux sont séparés d'un signe deux-points " : " comme le montre l'exemple ci-dessus. Une "valeur" peut être un nombre ou du texte, mais aussi une donnée binaire (comme une image) ou une collection d'autres paires clefs/valeurs comme le montre l'exemple ci-dessous :

{

"\_id": ObjectId("4efa8d2b7d284dad101e4bc7"),

"Nom" : "RAKOTO",

"Prénom" : "Lita",

"Âge" : 29,

"Adresse" :

{

"Rue" : "Mahabibo",

"Ville" : "Mahajanga"

}

}

Ici sont imbriqués des documents : le [champ](https://fr.wikipedia.org/wiki/Champ_(base_de_donn%C3%A9es)) "Adresse" contient un document de deux [champs](https://fr.wikipedia.org/wiki/Champ_(base_de_donn%C3%A9es)) : "Rue" et "Ville".

Certaines opérations sur les [champs](https://fr.wikipedia.org/wiki/Champ_(base_de_donn%C3%A9es)) telles que l'incrémentation peuvent être effectuées de façon [atomique](https://fr.wikipedia.org/wiki/Atomicit%C3%A9_(informatique)), c'est-à-dire sans déplacement ni copie du document.

**2. Manipulation des données**

**MongoDB** est livré avec des [liaisons](https://fr.wikipedia.org/wiki/Binding) pour les principaux [langages de programmation](https://fr.wikipedia.org/wiki/Langages_de_programmation) notamment le **JavaScript.** Ces pilotes permettent de manipuler la [base de données](https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es) et ses données directement depuis ces langages. Cependant **MongoDB** possède également un outil qui peut être utilisé en [ligne de commande](https://fr.wikipedia.org/wiki/Ligne_de_commande) et qui donne accès au langage natif de la [base de données](https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es) : le [**JavaScript**](https://fr.wikipedia.org/wiki/JavaScript), par l'intermédiaire duquel on peut également manipuler la base.

En tapant:

./mongo

dans la [ligne de commande](https://fr.wikipedia.org/wiki/Ligne_de_commande) du [système d'exploitation](https://fr.wikipedia.org/wiki/Syst%C3%A8me_d%27exploitation), on lance [l'interpréteur de commandes](https://fr.wikipedia.org/wiki/Interpr%C3%A9teur_de_commandes) interactif de **MongoDB**.

Dans une [base de données](https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es) nommée faciall qui contient plusieurs collections, voici comment on afficherait tous les documents de la collection nommée personnes:

> use faciall *// Sélectionne la base de données "faciall"*

> db.personnes.find(); *// Cherche et affiche tous les documents de la collection "personnes".*

Le résultat s'imprime à l'écran :

{"\_id”: ObjectId("4efa8d2b7d284dad101e4bc7"),"Prénom": "Andritiana"}

{"\_id": ObjectId ("4efa8d2b7d284dad101e4bc8”, Prénom": "Miandry"}

La documentation officielle de **MongoDB** décrit en détail les mécanismes de manipulation de données par l'intermédiaire de l'outil **mongo**. Pour manipuler les bases **MongoDB** depuis un [langage de programmation](https://fr.wikipedia.org/wiki/Langage_de_programmation), il convient en revanche de se reporter à la documentation correspondant au pilote (*driver*) du langage en question.

**3. Utilisation comme un système de fichier**

**MongoDB** peut être utilisé comme [système de fichiers](https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_fichiers), cette fonction assez inhabituelle pour une base de données permet pourtant de profiter de toutes les caractéristiques décrites dans la section [**Déploiement**](https://fr.wikipedia.org/wiki/MongoDB#D%C3%A9ploiement), à savoir la réplication et la répartition de données sur **un ensemble de**[**serveurs**](https://fr.wikipedia.org/wiki/Serveur_(informatique)), et de les mettre à profit pour gérer des fichiers.

Cette fonction, nommée **[GridFS](https://fr.wikipedia.org/w/index.php?title=GridFS&action=edit&redlink=1" \o "GridFS (page inexistante))**, est incluse dans les [pilotes](https://fr.wikipedia.org/wiki/Pilote_informatique) fournis avec **MongoDB** et utilisable sans difficulté particulière dans les langages de programmation. Le programmeur dispose de fonctions pour manipuler les fichiers et leur contenu, **MongoDB** se chargeant de leur gestion.

Dans un système fonctionnant sur plusieurs [serveurs **MongoDB**](https://fr.wikipedia.org/wiki/Serveur_(informatique)), les fichiers peuvent ainsi être automatiquement répartis et dupliqués entre les ordinateurs de façon transparente, offrant au système la tolérance aux pannes et la répartition de la charge que **MongoDB** apporte aux données.

Les [bases de données hiérarchiques](https://fr.wikipedia.org/wiki/Bases_de_donn%C3%A9es_hi%C3%A9rarchiques) ont été développées pour les matériels peu puissants des années 1970. Ceux des années 1990, plus puissants, ont pu utiliser le [modèle relationnel](https://fr.wikipedia.org/wiki/Mod%C3%A8le_relationnel) (bien plus consommateur, mais beaucoup plus souple). Les années 2010 permettent l'introduction des bases **not-only-**[**SQL**](https://fr.wikipedia.org/wiki/SQL) ("[**NoSQL**](https://fr.wikipedia.org/wiki/Nosql)" : pas-seulement-[SQL](https://fr.wikipedia.org/wiki/SQL)) comme **MongoDB** associant aux données des attributs (ou [champs](https://fr.wikipedia.org/wiki/Champ_(base_de_donn%C3%A9es))) placés à la demande en temps réel, et exploitables ensuite par le biais de [requêtes](https://fr.wikipedia.org/wiki/Recherche_d%27information). Chacune de ces deux approches tire profit de la baisse des coûts et de l'augmentation de puissance du matériel pour permettre des développements plus simples et plus rapides.

**V. Git**



**Git** est un [logiciel de gestion de versions](https://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_versions) [décentralisé](https://fr.wikipedia.org/wiki/Gestion_de_versions#Gestion_de_versions_d.C3.A9centralis.C3.A9e). C'est un [logiciel libre](https://fr.wikipedia.org/wiki/Logiciel_libre) créé par [**Linus Torvalds**](https://fr.wikipedia.org/wiki/Linus_Torvalds), auteur du [noyau **Linux**](https://fr.wikipedia.org/wiki/Noyau_Linux), et distribué selon les termes de la [licence publique générale **GNU**](https://fr.wikipedia.org/wiki/Licence_publique_g%C3%A9n%C3%A9rale_GNU) **version 2**. Le principal contributeur actuel de **git** et depuis plus de 16 ans est **[Junio C Hamano](https://fr.wikipedia.org/w/index.php?title=Junio_C_Hamano&action=edit&redlink=1" \o "Junio C Hamano (page inexistante))**. En 2016, il s’agit du [logiciel de gestion de versions](https://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_versions) le plus populaire qui est utilisé par plus de douze millions de personnes.

**Git** ne repose pas sur un [serveur](https://fr.wikipedia.org/wiki/Serveur_informatique) centralisé, mais il utilise un système de connexion [pair à pair](https://fr.wikipedia.org/wiki/Pair_%C3%A0_pair). Le code informatique développé est stocké non seulement sur l’ordinateur de chaque contributeur du projet, mais il peut également l'être sur un serveur dédié. C'est un outil de [bas niveau](https://fr.wikipedia.org/wiki/Langage_de_programmation_de_bas_niveau), qui se veut simple et performant, dont la principale tâche est de gérer l'évolution du contenu d'une [arborescence](https://fr.wikipedia.org/wiki/Arborescence).

**Git** indexe les fichiers d'après leur [somme de contrôle](https://fr.wikipedia.org/wiki/Somme_de_contr%C3%B4le) calculée avec la [fonction de hachage](https://fr.wikipedia.org/wiki/Fonction_de_hachage)[**SHA-1**](https://fr.wikipedia.org/wiki/SHA-1). Quand un [fichier](https://fr.wikipedia.org/wiki/Fichier_informatique) n'est pas modifié, la somme de contrôle ne change pas et le fichier n'est stocké qu'une seule fois. En revanche, si le fichier est modifié, les deux versions sont stockées sur le disque.

Contrastant avec les architectures de [logiciel de gestion de versions](https://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_versions) habituellement utilisées jusqu'alors, **Git** repose entièrement sur un petit nombre de structures de données élémentaires. [**Linus Torvalds**](https://fr.wikipedia.org/wiki/Linus_Torvalds)expliquaitainsi : « **Par bien des aspects, vous pouvez considérer Git comme un simple**[**système de fichiers**](https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_fichiers)**. Il est adressé par contenu, et possède la notion de versionnement, mais je l'ai vraiment conçu en prenant le point de vue d'un spécialiste des systèmes de fichiers (après tout, j'ai l'habitude de travailler sur des noyaux) et je n'avais absolument aucune envie de créer un système de gestion de version traditionnel.** » Les premières versions de **Git** offraient une interface rudimentaire pour manipuler ces objets internes avant que les fonctionnalités courantes de gestion de version ne soient ensuite progressivement ajoutées et raffinées.

**Git** est considéré comme performant, au point que certains autres logiciels de gestion de version (**[Darcs](https://fr.wikipedia.org/wiki/Darcs" \o "Darcs),**[**Arch**](https://fr.wikipedia.org/wiki/GNU_Arch)), qui n'utilisent pas de [base de données](https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es), se sont montrés intéressés par le système de stockage des fichiers de **Git** pour leur propre fonctionnement. Ils continuent toutefois à proposer des fonctionnalités plus évoluées.

Dès le début, **Git** a été pensé dans le but de fonctionner de façon décentralisée, c'est d'ailleurs l'une des clefs de son succès. La décentralisation de **Git** a aussi beaucoup apporté au développement des logiciels libres, puisque le besoin de demander un compte sur un dépôt [**SVN**](https://fr.wikipedia.org/wiki/Apache_Subversion) ou [**CVS**](https://fr.wikipedia.org/wiki/Concurrent_versions_system) centralisé devient obsolète. Il suffit de **forker** un projet ou de le **cloner** pour commencer à travailler dessus (avec tout l'historique du projet en local) et ensuite de proposer sa contribution (**pull request**) au dépôt principal (mainteneur principal du projet).

Les serveurs **Git** utilisent par défaut le [port](https://fr.wikipedia.org/wiki/Port_(logiciel)) 9418 pour le [protocole](https://fr.wikipedia.org/wiki/Protocole_de_communication) spécifique à Git. Les protocoles HTTP, [HTTPS](https://fr.wikipedia.org/wiki/HyperText_Transfer_Protocol_Secure) et [SSH](https://fr.wikipedia.org/wiki/Secure_Shell) peuvent aussi être utilisés.

**Fonctionnement**

Git possède deux [structures de données](https://fr.wikipedia.org/wiki/Structure_de_donn%C3%A9es) : une base d'objets et un cache de [répertoires](https://fr.wikipedia.org/wiki/R%C3%A9pertoire_(informatique)). Il existe quatre types d'objets:

* L'objet **blob** (pour **[binary large object](https://fr.wikipedia.org/wiki/Binary_large_object" \o "Binary large object)** désignant un ensemble de données brutes), qui représente le contenu d'un [fichier](https://fr.wikipedia.org/wiki/Fichier_informatique)
* L’objet **tree** (mot anglais signifiant arbre), qui décrit une arborescence de fichiers. Il est constitué d'une liste d'objets de type **blobs** et des informations qui leur sont associées, tel que le nom du fichier et les [permissions](https://fr.wikipedia.org/wiki/Droits_sur_un_fichier). Il peut contenir [récursivement](https://fr.wikipedia.org/wiki/Type_r%C3%A9cursif) d'autres **trees** pour représenter les sous-répertoires
* L'objet **commit** (résultat de l'opération du même nom signifiant « valider une **transaction** »), qui correspond à une arborescence de fichiers (**tree**) enrichie de métadonnées comme un message de description, le nom de l'auteur, etc. Il pointe également vers un ou plusieurs objets **commit** parents pour former un graphe d'historiques ;
* L'objet **tag** (étiquette) qui est une manière de nommer arbitrairement un **commit** spécifique pour l'identifier plus facilement. Il est en général utilisé pour marquer certains **commits**, par exemple par un numéro ou un nom de version.

La base des objets peut contenir n'importe quel type d'objets. Une couche intermédiaire, utilisant des **index** (les sommes de contrôle), établit un lien entre les objets de la base et l'arborescence des fichiers.

Chaque objet est identifié par une somme de contrôle [**SHA-1**](https://fr.wikipedia.org/wiki/SHA-1) de son contenu. **Git** calcule la somme de contrôle et utilise cette valeur pour déterminer le nom de fichier de l'objet. L'objet est placé dans un répertoire dont le nom correspond aux deux premiers caractères de la somme de contrôle. Le reste de la somme de contrôle constitue alors le nom du fichier pour cet objet.

Git enregistre chaque révision dans un fichier en tant qu'**objet blob uniqu**e. Les relations entre les objets blobs sont déterminées en examinant les objets **commit**. En général, les objets **blobs** sont stockés dans leur intégralité en utilisant la compression de la **[zlib](https://fr.wikipedia.org/wiki/Zlib" \o "Zlib)**. Ce principe peut rapidement consommer une grande quantité de place disque ; de ce fait, les objets peuvent être combinés dans des archives, qui utilisent la compression différentielle (c'est-à-dire que les blobs sont enregistrés sous la forme de différences par rapport aux autres blobs).

**Quelques commandes**

Git dispose notamment des commandes suivantes :

* Git init​ crée un nouveau dépôt
* Git clone​ clone un dépôt distant
* Git add​ ajoute de nouveaux objets blobs dans la base des objets pour chaque fichier modifié depuis le dernier *commit*. Les objets précédents restent inchangés
* Git commit​ intègre la somme de contrôle [SHA-1](https://fr.wikipedia.org/wiki/SHA-1) d'un objet **tree** et les sommes de contrôle des objets **commits** parents pour créer un nouvel objet **commit**
* Git branch​ liste les branches
* Git merge​ fusionne une branche dans une autre
* Git rebase​ déplace les commits de la branche courante devant les nouveaux commits d’une autre branche
* Git log​ affiche la liste des commits effectués sur une branche
* Git push​ publie les nouvelles révisions sur le **remote**. (La commande prend différents paramètres)
* Git pull​ récupère les dernières modifications distantes du projet (depuis le **Remote**) et les fusionne dans la branche courante
* Git stash​ stocke de côté un état non commité afin d’effectuer d’autres tâches
* Git checkout​ annule les modifications effectuées, déplacement sur une référence (branche, hash)
* Git switch​ changement de branche
* Git remote​ gestion des remotes.

**VI. Github**



GitHub est un service **web d'**[**hébergement**](https://fr.wikipedia.org/wiki/H%C3%A9bergeur_web)**et de gestion de développement de logiciels, utilisant le**[**logiciel de gestion de versions**](https://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_versions)[**Git**](https://fr.wikipedia.org/wiki/Git). Ce site est développé en [**Ruby on Rails**](https://fr.wikipedia.org/wiki/Ruby_on_Rails)**et**[**Erlang**](https://fr.wikipedia.org/wiki/Erlang_(langage))**par Chris Wanstrath, PJ Hyett et Tom Preston-Werner.** **GitHub** propose des comptes professionnels payants, ainsi que des comptes gratuits pour les projets de [logiciels libres](https://fr.wikipedia.org/wiki/Logiciels_libres). Le site assure également un contrôle d'accès et des fonctionnalités destinées à la collaboration comme le suivi des bugs, les demandes de fonctionnalités, la gestion de tâches et un wiki pour chaque projet.

**GitHub** est centré vers l'aspect social du développement. En plus d'offrir l'hébergement de projets avec [**Git**](https://fr.wikipedia.org/wiki/Git), le site offre de nombreuses fonctionnalités habituellement retrouvées sur les [réseaux sociaux](https://fr.wikipedia.org/wiki/R%C3%A9seau_social) comme les flux, la possibilité de suivre des personnes ou des projets ainsi que des graphes de réseaux pour les [dépôts](https://fr.wikipedia.org/wiki/D%C3%A9p%C3%B4t_(informatique)) (repository). **GitHub** offre aussi la possibilité de créer un [wiki](https://fr.wikipedia.org/wiki/Wiki) et une page web pour chaque dépôt. Le site offre aussi un [logiciel de suivi de problèmes](https://fr.wikipedia.org/wiki/Logiciel_de_suivi_de_probl%C3%A8mes) (issue *tracking system*). **GitHub** propose aussi l'intégration d'un grand nombre de services externes, tels que l'intégration continue, la gestion de versions, badges, chat basés sur les projets, etc. Les documentations des projets sont écrites en langage **[Markdown](https://fr.wikipedia.org/wiki/Markdown" \o "Markdown)** (fichiers .md). L'option **[webhook](https://fr.wikipedia.org/wiki/Webhook" \o "Webhook)** permet de communiquer avec des outils d'[intégration continue](https://fr.wikipedia.org/wiki/Int%C3%A9gration_continue).

**VII. Express.js**



**Express.js** est un [**framework**](https://fr.wikipedia.org/wiki/Framework) pour construire des [applications web](https://fr.wikipedia.org/wiki/Applications_web) basées sur [**Node.js**](https://fr.wikipedia.org/wiki/Node.js). C'est de fait le **framework** standard pour le développement de serveur en [**Node.js**](https://fr.wikipedia.org/wiki/Node.js). L'auteur original, **TJ Holowaychuck**, le décrit comme un serveur inspiré de [**Sinatra**](https://fr.wikipedia.org/wiki/Sinatra_(logiciel)) dans le sens qu'il est relativement minimaliste tout en permettant d'étendre ses fonctionnalités via des **plugins**.

D'autres tâches de développement web ne sont pas directement prises en charge par **Node** de façon native. Si vous voulez ajouter différentes manipulations pour divers requêtes **HTTP (GET, POST, DELETE, etc.),** gérer différemment des requêtes vers plusieurs chemins **URL** ("routes"), servir des pages statiques ou utiliser des modèles pour créer dynamiquement la réponse, alors vous devrez écrire tout le code vous-même ou, pour éviter de réinventer la roue, vous servir des cadres applicatifs web (**frameworks**).

[**Express**](https://expressjs.com/) est le **framework** actuellement le plus populaire dans **Node** et est la bibliothèque sous-jacente pour un grand nombre d'autres [cadres applicatifs web pour **Node**](https://expressjs.com/fr/resources/frameworks.html). Il fournit des mécanismes pour:

* Écrire des fonctions de traitement pour différentes requêtes HTTP répondant à différentes URI (par le biais des routes).
* Intégrer avec les moteurs de rendu de « vues » dans le but de générer des réponses en insérant des données dans des modèles (« templates »). Configurer certains paramètres d'applications comme le port à utiliser à la connexion et l'emplacement des modèles nécessaires pour la mise en forme de la réponse.
* Ajouter des requêtes de traitement « **middleware** » (intergiciel) où vous le voulez dans le tunnel gestionnaire de la requête.

Bien qu'Express soit assez minimaliste, des **middlewares** (fonctions intermédiaires) compatibles ont été créés pour résoudre quasiment tous les problèmes de développement web. Il existe des bibliothèques pour se servir des cookies, gérer les sessions, la connexion de l'utilisateur, les paramètres de l'URL, les données POST, les entêtes de sécurité et d'autres encore.

**A qui ressemble du code Express**

Dans un site web utilisé pour traiter des données, une application web attend des requêtes **HTTP** du navigateur web (ou d'un autre client). Quand une requête est reçue, l'application cherche quelle action est requise en fonction du modèle de l'URL et des possibles informations associés contenues dans les données **POST** ou **GET**. Selon ce qui est requis, il pourra alors lire ou écrire des informations dans une base de données ou effectuer d'autre tâches pour satisfaire la requête. L'application va alors retourner une réponse au navigateur web, souvent une page HTML créée dynamiquement pour le navigateur, en intégrant les données récupérées dans un modèle HTML.

**Express** fournit des méthodes pour spécifier quelle fonction est appelée pour une méthode HTTP particulière (GET, POST, SET, etc.) et un modèle d'URL ("Route"), ainsi que des méthodes pour spécifier quel moteur de rendu de vues ("view") est utilisé, où sont les modèles de vues et quel modèle utiliser pour générer une réponse. Vous pouvez utiliser les fonctions intermédiaires d'**Express** pour prendre en charge les cookies, les sessions, les utilisateurs, obtenir les paramètres **POST/GET**, etc. Vous pouvez utiliser n'importe que système de base données supporté par **Node** (Express ne définit aucun comportement relatif aux bases de données).

Les sections suivantes expliquent quelques choses communes que vous verrez en travaillant avec du code Express et Node.

Tout d'abord intéressons-nous à l'exemple [**Hello World**](https://expressjs.com/fr/starter/hello-world.html) standard d'Express ;

const express = require('express');

const app = express();

const port = 3000;

app.get('/', (req, res) => {

res.send('Hello World!')

});

app.listen(port, () => {

console.log(`à l'écoute sur le port ${port}!`)

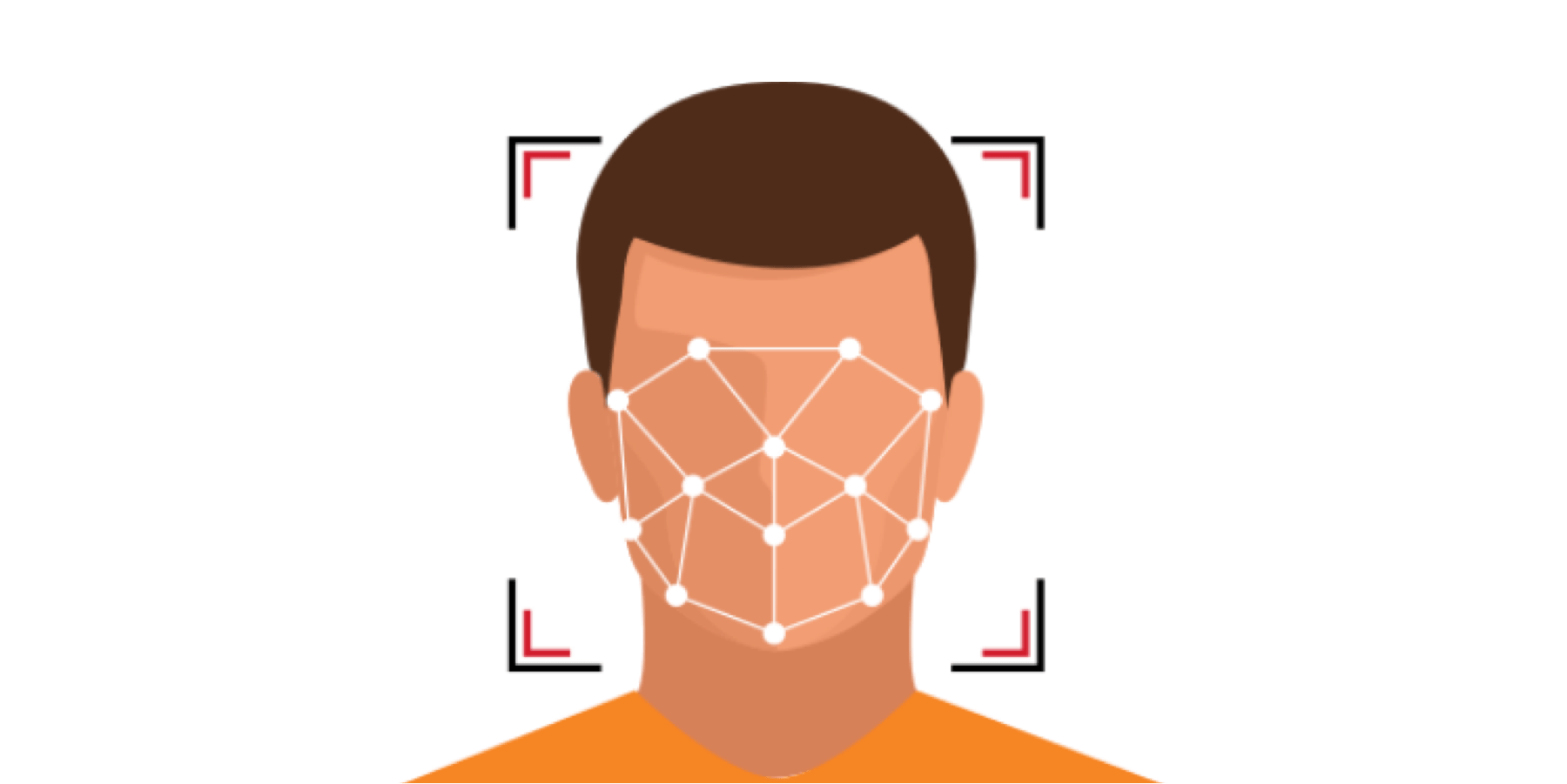
});

Les deux premières lignes importent (avec require() ) le module **express** et créent une [application **express**](https://expressjs.com/en/4x/api.html#app). Cet objet, qui est traditionnellement nommé **app**, possède des méthodes pour acheminer les requêtes HTTP, configurer les **intergiciels**, rendre les vues HTML, enregistrer un moteur de modèles et modifier les [paramètres de l'application](https://expressjs.com/en/4x/api.html#app.settings.table) qui contrôlent le comportement de l'application (par exemple, le mode d'environnement, si les définitions de route sont sensibles à la casse, etc.).

La partie centrale du code (les trois lignes commençant par **app.get**) montre une définition de route. La méthode **app.get**() spécifie une fonction de rappel qui sera invoquée chaque fois qu'il y aura une requête HTTP GET avec un chemin ('/') relatif à la racine du site. La fonction de rappel prend une requête et un objet réponse comme arguments, et appelle simplement **[send()](https://expressjs.com/en/4x/api.html" \l "res.send)** sur la réponse pour renvoyer la chaîne de caractères "Hello World!".

Le dernier bloc démarre le serveur sur le port 3000 et affiche un commentaire de journal dans la console. Avec le serveur en cours d'exécution, vous pouvez aller sur localhost:3000 dans votre navigateur pour voir l'exemple de réponse renvoyé.

**VIII. Face-api.js**



Le module **JavaScript** **face-api.js** implémente des réseaux de neurones convolutifs pour résoudre la détection et la reconnaissance des visages ainsi que la reconnaissance des repères faciaux. L'api **face-api.js** exploite [**TensorFlow.js**](https://www.infoq.com/news/2018/04/tensorflow-javascript-browser/) et est optimisé pour le bureau et le web mobile. Comme expliqué par le créateur de face-api.js, **Vincent Mühler**, la reconnaissance faciale avec le **deep learning** est un processus minutieux :

*« Pour rester simple, ce que nous voulons réellement réaliser, c'est identifier une personne donnée à partir d'une image de son visage, par exemple l'image d'entrée. La manière de procéder, est de fournir une (ou plusieurs) image(s) pour chaque personne que nous voulons reconnaître, avec leur nom, par exemple les données de références. Maintenant nous comparons l'image d'entrée aux données de références et trouvons l'image de référence la plus similaire. Si les deux images sont suffisamment similaires, nous donnons le nom de la personne, sinon nous affichons "inconnu". Ça parait être un bon plan ! Toutefois, deux problèmes persistent. Premièrement, que se passe-t-il si nous avons une image sur laquelle plusieurs personnes apparaissent, et que nous souhaitons toutes les reconnaître ? Deuxièmement, nous devons être en mesure d'obtenir une sorte de mesure de similarité pour deux images de visages afin de les comparer... Il existe plusieurs modèles disponibles avec face-api.js, incluant la détection de visage, la détection de repère de visage, la reconnaissance faciale, la reconnaissance d'expression faciale, l'estimation de l'âge et la reconnaissance du genre. »*

Les développeurs chargent les différentes instances de réseau neuronal via **faceapis.nets** et chargent de manière asynchrone un modèle, une **map** de tenseur nommée (**named tensor map**) ou des modèles non compressés avec un poids en **Float32Array**. Les versions récentes de **face-api.js** ont conservé leur cohérence avec les dernières versions de **TensorFlow.js**, en affinant et dépréciant certaines **APIs** à mesure que le projet se rapproche d'une version 1.0. La [documentation complète de l'API face-api.js](https://justadudewhohacks.github.io/face-api.js/docs/globals.html) est disponible, ainsi que [plusieurs exemples](https://github.com/justadudewhohacks/face-api.js). Le projet face-api.js est un logiciel open source disponible sous la licence **MIT**.

**Conception du projet**