I. ANALYSE DU PROJET :

Ce chapitre portera sur une spécification des besoins auxquels doit répondre le système, passant ensuite à l'analyse des différentes technologies de base adoptées pour réaliser ce projet, en terminant par les diagrammes de cas d'utilisation relatifs aux acteurs du système.

1. Analyse et spécifications des besoins :

L'analyse et la spécification des besoins représentent la première phase du cycle de développement d'un logiciel. Elle sert à identifier les acteurs réactifs du système et leur associer chacun l'ensemble d'actions avec lesquelles il intervient dans l'objectif de donner un résultat optimal et satisfaisant au client. Ainsi. Dans ce chapitre, nous commençons en premier lieu par une spécification des besoins auxquels doit répondre l'application, passant ensuite à l'analyse de ces besoins à travers l'introduction des acteurs, le schéma de la base de données et les diagrammes de cas d'utilisation relatifs à ces acteurs.

a. Cahier des charges :

À voir le fichier « ***Cahier-des-charges.pdf »***

b. Conception

Lors de cette section, nous allons identifier le diagramme des cas d'utilisation réalisé pour mettre en œuvre l'architecture de l'application. La motivation fondamentale de la modélisation est de fournir une démarche antérieure afin de réduire la complexité du système étudié lors de la conception et d'organiser la réalisation du projet en définissant les modules et les étapes de la réalisation. Plusieurs démarches de modélisation sont utilisées. Nous adoptons dans notre travail une approche objet basée sur un outil de modélisation **UML**.

En fait, **UML** est un standard ouvert contrôlé par un consortium d'entreprises qui a été fondé pour construire des standards qui facilitent l'interopérabilité et plus spécifiquement, l'interopérabilité des systèmes orientés objet. UML est issu de l'unification de nombreux langages de modélisation graphique orientée objet. Il unifie à la fois les notations et les concepts orientés objets.

- **Acteurs :**

Dans le cadre de mon application nous avons distingué trois types d'acteurs :

- **Le visiteur (**visitor**):**

  C'est un exploitant externe de l'application, ses propres autorisations/permissions son :

  - Consulter les utilisateurs existants dans l'application.

- Consulter les profil/portefeuille récapitulatif des utilisateurs existant dans la base de données.
- Consulter les profile/portefeuille complète(détaillé) des utilisateurs existant dans la base de données.
- S'inscrire (register) ; Se connecter (login).

- **Le développeur (**developer**):**

Après avoir **s'enregistrer/s'authentifier**, il sera considéré comme un utilisateur actif, il aura à sa disposition certaines fonctionnalité du système, à savoir :

- Consulter son Tableau de bord (Dashboard).

- Consulter son profil/portefeuille à savoir :
    - Consulter ses informations personnelles de base :
        - *Nom, prénom, photo, biographie…*
    - Consulter les informations sur ses expériences professionnelles.
    - Consulter les informations sur ses diplômes (*parcours académique*).
    - Consulter les dernier dépôt GitHub, à conditions s'il décide de mettre leur nom d'utilisateur GitHub.

- Manipuler son profil/portefeuille à savoir :
    - Mettre à jour ses propres informations personnelles de base :
        - *Nom, prénom, photo, biographie, compétence…*
    - Ajouter/supprimer une expérience (experience) .
    - Ajouter/supprimer un diplôme (education).

- Access à l'espace publications (posts) à savoir :
    - Consulter les publications existantes.
    - Manipuler ses propres publications :
        - Publier un nouveau post.
        - Modifier son propre post.
        - Supprimer son propre post.
    - Like/unlike un post.
    - Afficher le nombre de commentaires d'une publication.
    - Manipuler ses propres commentaires :
        - Laisser un nouveau commentaire.
        - Modifier son propre commentaire.
        - Supprimer son propre commentaire.

- Supprimer son compte.

- **Diagramme des cas d'utilisation :**

Le système assure pour les acteurs diverses fonctions mises en valeur à travers le diagramme de cas d'utilisation illustré par la figure 1

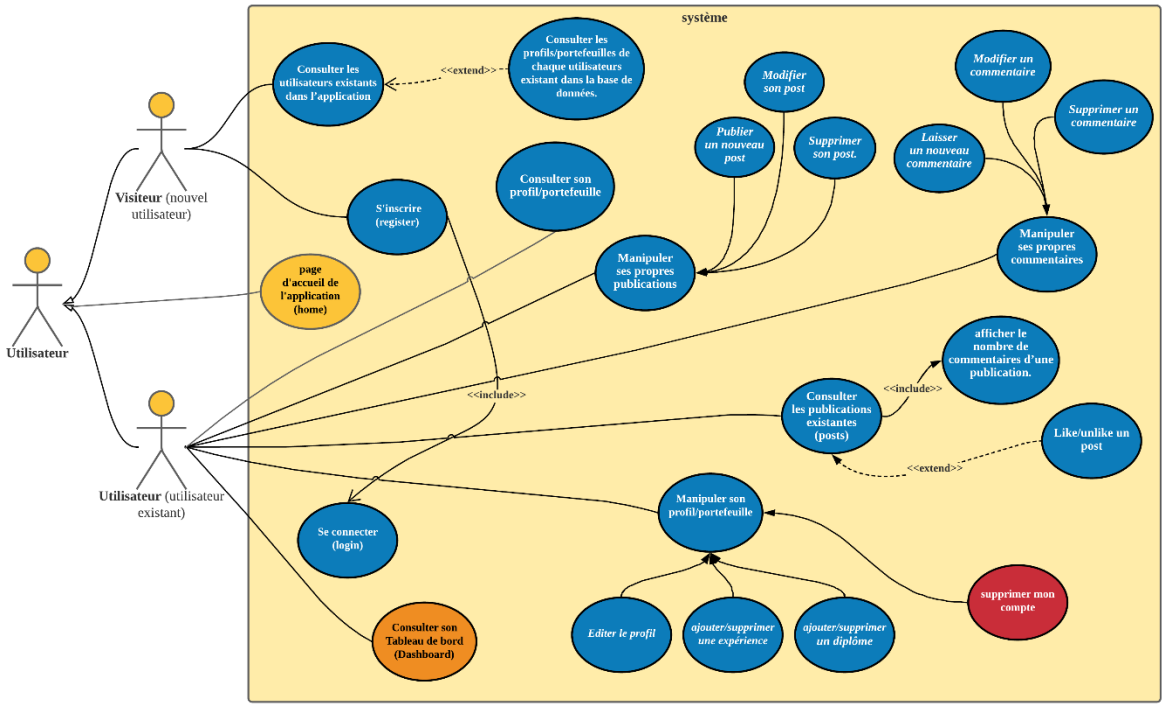


Figure 1: diagramme de cas d'utilisation (DCU)

c. Schéma non relationnel (*NoSQL database*) :

Les bases de données NoSQL sont de plus en plus utilisées mais savez-vous vraiment ce qu'elles renferment ? Prenons les exemples de Datastax (Cassandra) et MongoDB qui figurent parmi les solutions NoSQL les plus utilisées du marché.

A quoi sert une base NoSQL ?

Les bases relationnelles ont été « inventées » par Edgar F. Codd en 1970. A l'époque, ce fut une approche révolutionnaire (modèle relationnel des données, algèbre de relations, sélections, projections, jointures, etc…). Oracle fut la première société à commercialiser un produit conforme à ce concept, dès 1977, avec le succès que l'on connait. Ces dernières années, plusieurs avis nuancés ont été émis concernant le modèle relationnel :

- **Difficultés pour sauver et récupérer des grappes d'objets avec des programmes écrits en Java ou C#... (on parle de « mismatch d'impédance)**
- **Difficultés pour satisfaire les besoins des applications Web à grande échelle (nouveaux besoins métiers, nouvelles contraintes techniques)**

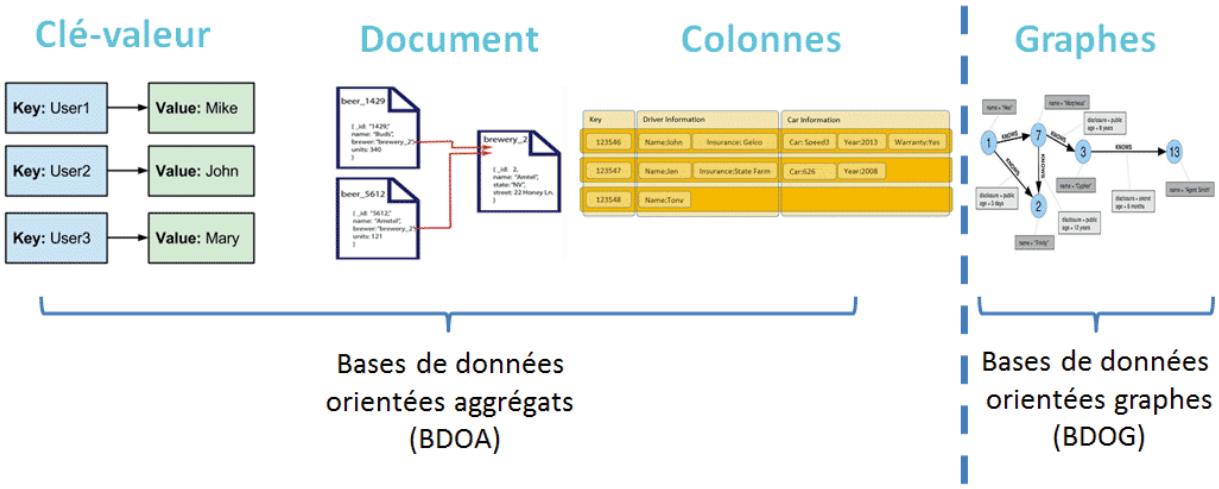
Voici les 3 aspects qui ont motivé la nouvelle vague de bases NoSQL (Not Only SQL..) :

- **Le VOLUME des données créées double tous les 2 ans. IDC estime qu'en 2020 le volume atteindra 44 Zettabytes (1 ZB = 1 milliards de terabytes)**
- **La VARIÉTÉ des types de données créées (Smartphones)**
- **La VÉLOCITÉ avec laquelle les données changent est également très importante (Internet of Things)**

En fait, le terme base NoSQL définit une nouvelle génération de produits qui ne suivent pas le modèle relationnel. Mais l'architecture de ces produits varie beaucoup entre eux.

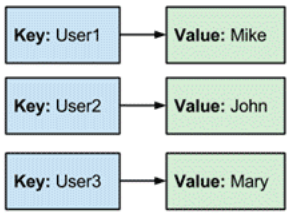
4 types de bases de données NoSQL

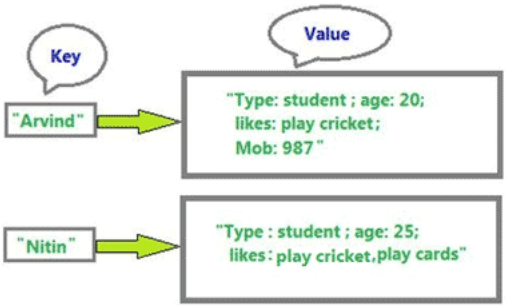
Clé- valeur, document, colonnes et **graphes** sont les 4 types de bases de données NoSQL. Etudions d'un peu plus près chacun de ces types.



Type 1 : Entrepôts clé-valeur (ECV)

Les données sont stockées en clé-valeur : une clé plus un BLOB (dans lequel on peut mettre : nombre, date, texte, XML, photo, vidéo, structure objet).





Facilement scalable

Temps de réponse en écriture / lecture très bas

Mises à jour compliquées

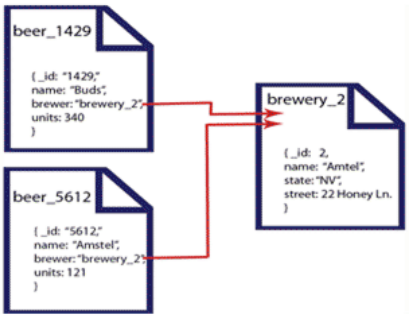
Requêtes rudimentaires

Exemples d'implémentation

Amazon DynamoDB (Beta) ORACLE BERKELEY DB 11g redis

Type 2 : Bases orientées documents

Ces bases de données stockent des données semi-structurées : le contenu est formaté JSON ou XML, mais la structure n'est pas contrainte.



Requétage plus complet

Flexibilité

Evolutif au cours du temps

Duplication des données

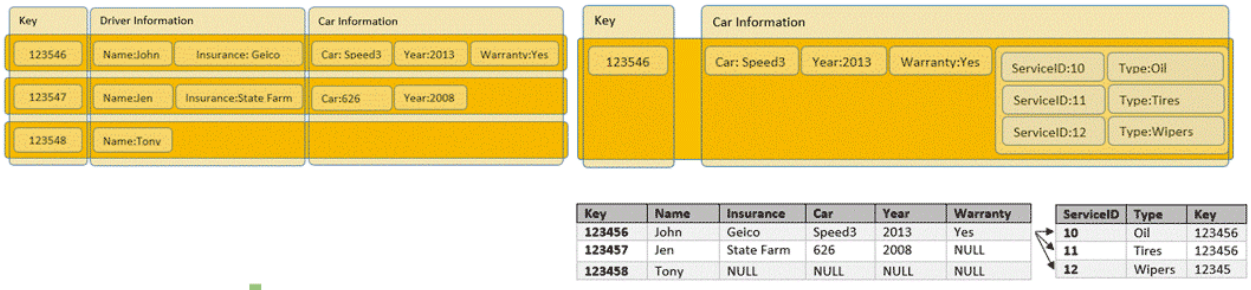
Cohérence pas forcément assurée

Exemples d'implémentation

CouchDB relax mongoDB terrastore

Type 3 : Bases orientées colonnes

Ces bases de données se rapprochent des bases de données relationnelles, à ceci près qu'elles permettent de remplir un nombre de colonnes variable.



Key	Name	Insurance	Car	Year	Warranty
123456	John	Geico	Speed3	2013	Yes
123457	Jen	State Farm	626	2008	NULL
123458	Tony	NULL	NULL	NULL	NULL

ServiceID	Type	Key
10	Oil	123456
11	Tires	123456
12	Wipers	12345

+ Capacité de stockage accrue

+ Accès rapide aux données

− Efficace surtout pour des données de même type et similaires

− Requêtage limité

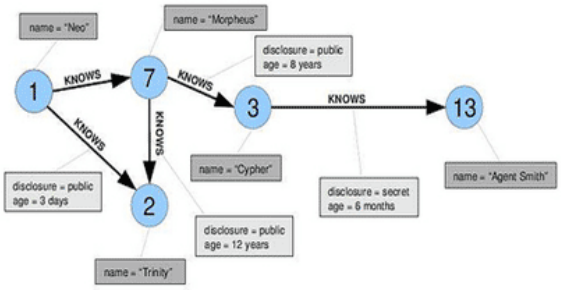
Exemples d'implémentation

APACHE HBASE

cassandra

Type 4 : Bases de données orientées graphes

Ces bases de données, basées sur la théorie des graphes, sont gérées par noeuds, relations et propriétés. Elles gèrent des données spatiales, sociales ou financières (dépôts/retraits).



+ Adapté à la gestion de données relationnelles

+ Architecture modelable

− Architecture limitée à certains cas

Exemple d'implémentation

TITAN

Neo4j

Nous notons que, parmi toutes les solutions NoSQL figurant sur ce Magic Quadrant (en plus des bases relationnelles), les produits ***DataStax*** (fondé sur Cassandra) et ***MongoDB*** figurent en très bonne place (carré « **LEADERS** »). Pour mémoire, ces deux produits appartiennent respectivement aux groupes 3 et 2 décrits plus-haut dans ce billet.

2. Développement et Technologies :

Nous allons maintenant aborder le contexte plus technique du stage avec une courte présentation des technologies utilises pour le développement de l'application. En commençant par l'architecture de l'application jusqu'au librairies d'interface utilisateur, ce qui nous a concerné le plus, c'est de travailler avec les dernières versions de ces technologies pour garantir l'efficacité du programme.

Le principe pour lequel on a choisi ces technologies c'est qu'il est difficile de développer un système logiciel qui respecte les exigences fonctionnels et non fonctionnels sans **utiliser l'expérience des autres**.

a. Plate-forme Node.js :

Node.js est une plate-forme JavaScript extrêmement puissante, basée sur le moteur JavaScript V8 de Google Chrome, utilisée pour développer des applications Web intensives d'E / S telles que des sites de streaming vidéo, des applications à une seule page, des applications de chat en ligne et d'autres applications Web.

Node.js est utilisé aussi bien par les grandes entreprises établies que par les startups nouvellement créées. Open source et totalement gratuite, la plateforme est utilisée par des milliers de développeurs à travers le monde. Il apporte de nombreux avantages à la table, ce qui en fait un meilleur choix que d'autres plates-formes côté serveur comme Java ou PHP dans de nombreux cas.

b. Applications Web (*Architecture trois-tiers*) :

Une application Web, comme vous le savez peut-être déjà, est un programme qui s'exécute sur un serveur et qui est rendu par un navigateur client, utilisant Internet pour accéder à toutes les ressources de cette application. Il peut généralement être facilement divisé en trois parties :

1. **Client**
2. **Serveur**
3. **Base de données**

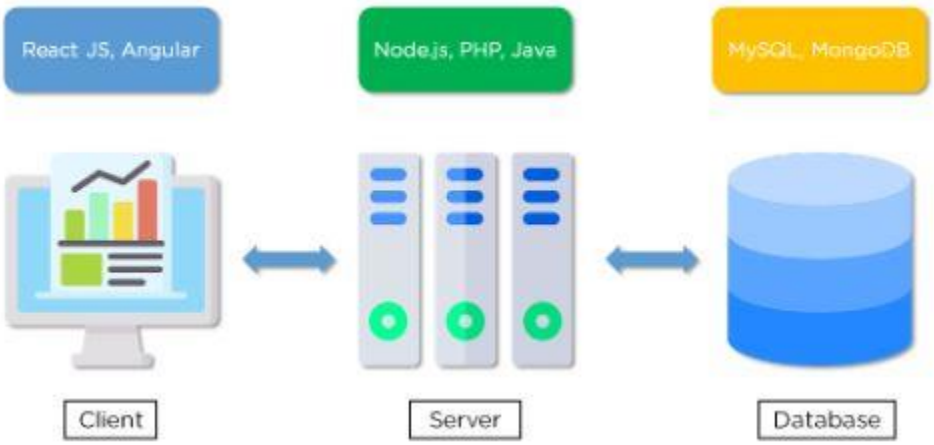


Figure 2: Architecture trois-tiers

- **Client :**

L'utilisateur interagit avec la partie frontale d'une application Web. Le front-end est généralement développé à l'aide de langages tels que les styles HTML et CSS, ainsi que d'une utilisation intensive de frameworks basés sur JavaScript tels que ReactJS et Angular, qui aident à la conception d'applications.

- **Serveur :**

Le serveur est chargé de prendre les demandes des clients, d'exécuter les tâches requises et de renvoyer les réponses aux clients. Il agit comme un middleware entre le front-end et les données stockées pour permettre des opérations sur les données par un client. Node.js, PHP et Java sont les technologies les plus utilisées pour développer et maintenir un serveur Web.

- **Base de données :**

La base de données stocke les données d'une application Web. Les données peuvent être créées, mises à jour et supprimées chaque fois que le client le demande. **MySQL** et **MongoDB** font partie des bases de données les plus utilisées pour stocker des données pour les applications Web.

c. Architecture du serveur Node.js :

Node.js utilise l'architecture «***Single Threaded Event Loop***» pour gérer plusieurs clients simultanés. Le modèle de traitement Node.js est basé sur le modèle événementiel JavaScript ainsi que sur le mécanisme de rappel JavaScript.

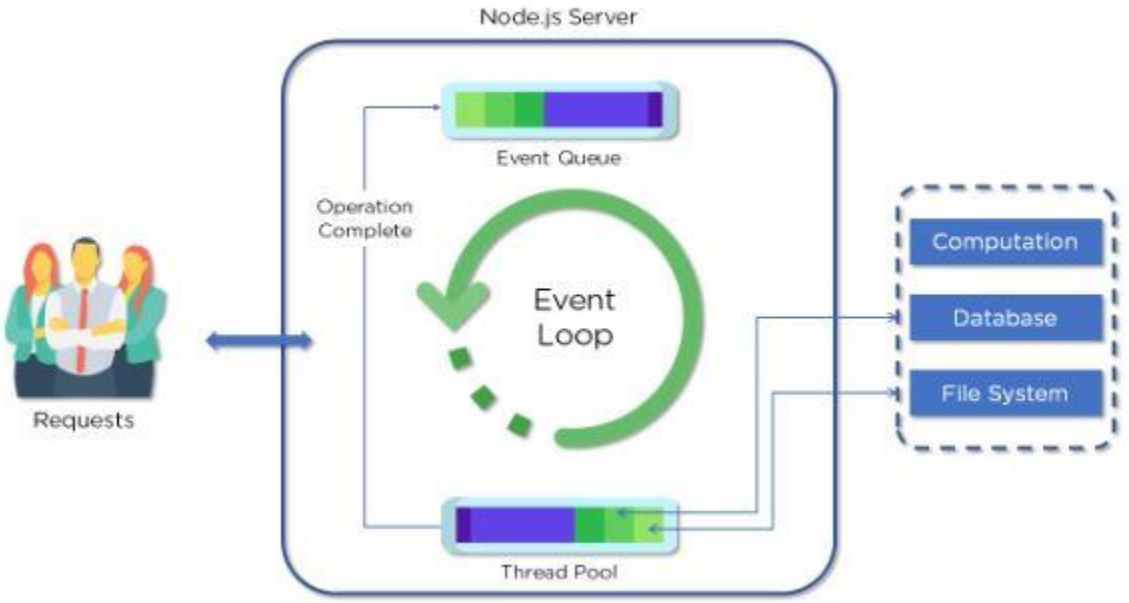


Figure 3: architecture Node.js

Voyons maintenant chaque partie de l'architecture Node.js et le flux de travail d'un serveur Web développé à l'aide de Node.js.

- **Parties de l'architecture Node.js :**

- **Demandes (Requests)**

Les demandes entrantes peuvent être bloquantes (complexes) ou non bloquantes (simples), selon les tâches qu'un utilisateur souhaite effectuer dans une application Web

- **Serveur Node.js (Node.js Server)**

Le serveur Node.js est une plate-forme côté serveur qui prend les demandes des utilisateurs, traite ces demandes et renvoie les réponses aux utilisateurs correspondants

- **File d'attente d'événements (Event Queue)**

La file d'attente d'événements dans un serveur Node.js stocke les demandes client entrantes et transmet ces demandes une par une dans la boucle d'événements

- **Pool de threads (Thread Pool)**

Le pool de threads comprend tous les threads disponibles pour effectuer certaines tâches qui pourraient être nécessaires pour répondre aux demandes des clients

- **Boucle d'événement (Event Loop)**

Event Loop reçoit indéfiniment les demandes et les traite, puis renvoie les réponses aux clients correspondants

- **Ressources externes (External Resources)**

Des ressources externes sont nécessaires pour traiter les demandes de blocage des clients. Ces ressources peuvent être destinées au calcul, au stockage de données, etc.

- **Le flux de travail de l'architecture Node.js (workflow) :**

Un serveur Web développé à l'aide de Node.js a généralement un flux de travail assez similaire au diagramme illustré ci-dessous. Explorons ce flux d'opérations en détail.

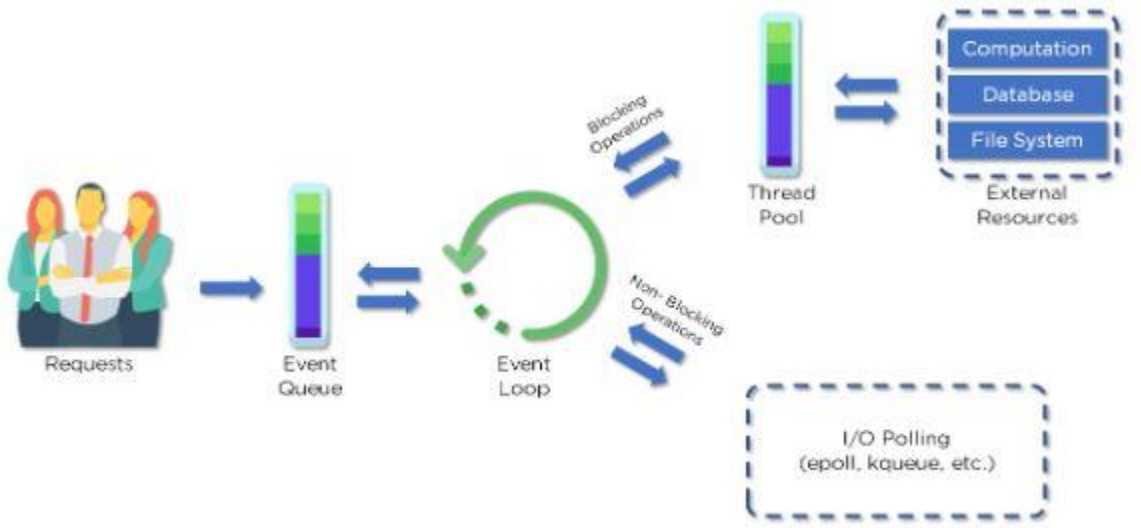


Figure 4: Flux de travail de l'architecture Node.js (workflow)

- Les clients envoient des demandes au serveur Web pour interagir avec l'application Web. Les demandes peuvent être non bloquantes ou bloquantes :
  - Rechercher des données
  - Suppression de données
  - Mise à jour des données
- Node.js récupère les demandes entrantes et ajoute ces demandes à la file d'attente d'événements
- Les demandes sont ensuite transmises une par une via la boucle d'événements. Il vérifie si les requêtes sont suffisamment simples pour ne nécessiter aucune ressource externe
- Event Loop traite les demandes simples (opérations non bloquantes), telles que l'interrogation d'E / S, et renvoie les réponses aux clients correspondants

Un seul thread du pool de threads est affecté à une seule requête complexe. Ce thread est responsable de l'exécution d'une demande de blocage particulière en accédant aux ressources externes, telles que le calcul, la base de données, le système de fichiers, etc.

Une fois que la tâche est complètement effectuée, la réponse est envoyée à la boucle d'événements qui à son tour renvoie cette réponse au client

d. Avantages de l'architecture Node.js

L'architecture Node.js présente plusieurs avantages qui donnent à la plate-forme côté serveur un avantage distinct par rapport aux autres langages côté serveur :

- **La gestion de plusieurs demandes client simultanées est rapide et facile**

  Avec l'utilisation de la file d'attente d'événements et du pool de threads, le serveur Node.js permet une gestion efficace d'un grand nombre de demandes entrantes.

- **Pas besoin de créer plusieurs threads**

  La boucle d'événements gère toutes les demandes une par une, il n'est donc pas nécessaire de créer plusieurs threads. Au lieu de cela, un seul thread suffit pour gérer une demande entrante bloquante.

- **Nécessite moins de ressources et de mémoire**

  Le serveur Node.js, la plupart du temps, nécessite moins de ressources et de mémoire en raison de la façon dont il gère les demandes entrantes. Étant donné que les demandes sont traitées une par une, le processus global devient moins pénible pour la mémoire.

Tous ces avantages contribuent à rendre les serveurs développés à l'aide de Node.js beaucoup plus rapides et réactifs par rapport à ceux développés à l'aide d'autres technologies de développement de serveurs.

e. M.E.R.N stack :

C'est quoi la MERN stack ?

MERN signifie "**mongodb**, **express**, **react**, **nodejs**", c'est une combinaison de 4 technologies vous permettant de créer un site web de A à Z, le frontend et le backend.

Le gros avantage de la MERN stack, c'est que l'on utilise uniquement le javascript, REACT côté client et NodeJS côté serveur.

Le point central de la MERN stack, c'est la capacité de développer un site web seul. Dis comme ça, on peut avoir l'impression que c'est évident, tous les développeurs web peuvent créer un site web de A à Z non ?

Et bien, absolument pas. La grande majorité des développeurs web ne savent créer qu'un côté du site web. Soit la partie base de données, algorithme et logique, c'est à dire le backend. Ou la partie design, expérience utilisateur et animations, c'est à dire le frontend.

Les développeurs full stack sont en mesure de gérer les deux côtés, c'est une compétence extrêmement recherchée. Les petits business n'ont pas les moyens de recruter deux développeurs, un full stack sera donc mieux payé et beaucoup plus sollicité.

La MERN stack, c'est la combinaison des technologies les plus puissantes et les plus versatiles sur le marché. On entend d'ailleurs aussi parler de MEAN stack. Dans ce cas, on remplace REACT par AngularJS, un autre Framework permettant de gérer le frontend.

Comment fonctionne la MERN stack ?

Il me semble important de comprendre comment fonctionne chaque technologie contenue dans la MERN stack. Dans cette partie, nous allons donc voir à quoi correspond exactement chaque lettre.

M pour MongoDB

Mongo DB est un système de base de données fonctionnant différemment du SQL. On n'utilise pas des tables mais des documents contenant des données sous format JSON.

Les documents MongoDB permettent de stocker beaucoup de données et d'établir des relations comme n'importe quel système SQL.

Dans la MEAN stack, on utilise MongoDB parce que les données sont gérées en JSON. Puisque ce langage provient initialement du Javascript, vous vous doutez qu'il est extrêmement simple de directement convertir des données de Javascript vers MongoDB et vice versa.

Pour se faire, il existe des librairies comme mongoose qui traduisent directement vos documents vers des objets et listes javascript directement utilisable dans vos programmes. La manipulation des données est donc extrêmement simple et efficace.

N pour Node JS

Alors, je sais que ne je vais pas dans l'ordre, mais il est important de prendre le N pour comprendre le E.

Node JS est un langage de programmation utilisant la syntaxe de Javascript. Vous comprenez donc que c'est un langage qui fonctionne en conjonction avec Javascript. Il est beaucoup plus simple de faire communiquer les deux parties.

De plus, si vous maîtrisez déjà javascript, apprendre Node JS sera beaucoup plus simple, et vice versa.

Enfin, Node JS est un langage de programmation extrêmement rapide. Bien plus que tous les autres langages de programmation web. Il utilise la technologie asynchrone pour gagner en puissance et en vitesse. C'est une technologie difficile à maîtriser mais extrêmement performante entre de bonnes mains.

Beaucoup de grandes entreprises utilisent Node JS. Si vous utilisez des systèmes de chat en ligne comme Facebook, vous pouvez être sûr que c'est Node JS derrière la machine.

- **E pour Express.js**

Vous savez ce qu'on utilise pour la base de données, vous savez quel langage on utilise pour le serveur, mais comment créer un serveur web ? Et comment manipuler la base de données ? Et bien c'est là qu'express entre en jeux.

Express c'est un Framework permettant de développer un site web facilement et efficacement. Il utilise le système de middleware permettant d'empiler des fonctions afin de créer un site web complexe.

Avec express, on ne va pas directement créer le site web sur lequel l'utilisateur va se connecter. On va créer la **REST API**
Une rest API, c'est un site web qui va permettre de récupérer des données et de les modifier via des requêtes HTTP. On exécutera donc ces requêtes sur la partie React afin de récupérer des données et de les enregistrer.

- **R pour React.js**

Parlons pour finir de React JS. C'est la technologie qui va nous permettre de créer la partie visible du site web.

React JS est un framework SPA. C'est une technologie qui permet de rendre tout le site web en Javascript. Cette méthode va rendre votre site beaucoup plus rapide puisqu'il n'aura jamais à télécharger de contenu statique.
Les animations, les transitions et les chargements sont bien plus rapides.

Sans trop rentrer dans les détails, React fonctionne avec un système de composants. On crée des bouts de code qui rendent de l'HTML et on peut les utiliser dans différentes parties de notre site web, ce qui permets de réduire énormément le temps de développement.

De grands sites web comme Netflix ou Facebook utilisent React. C'est le choix par excellence si vous voulez créer un site web extrêmement rapide.

Pour finir, sachez que React permets aussi de créer des applications mobiles et des logiciels. Il est donc possible de porter votre application sur plusieurs supports.

Comment est-ce que tout fonctionne ensemble ?

Maintenant que vous comprenez comment chaque partie fonctionne individuellement, je vais vous expliquer comment est-ce qu'une application MERN utilise toutes ces technologies en conjonction pour créer des applications puissantes et portatives.

On peut facilement découper la MERN stack en deux grands blocs. Comme un site web classique qui possède un backend et un frontend, la MERN stack possède aussi cette coupure.

MongoDB, Node JS et express permettent de développer la partie backend. Sur un site internet normal, cette partie va se charger de manipuler les bases de données, mais aussi de gérer les routes et le rendu des scripts / documents html.

Dans le cas de la MERN stack, notre but va être de créer des url, par exemple : site/utilisateurs/creer sur lesquels on va pouvoir créer, modifier et lire les données contenues dans MongoDB.

Ces url fonctionnent comme des fonctions. On peut voir les appels HTTP comme des déclencheurs. On donne des données dans la requête et le serveur s'occupe de modifier dans la base de données et de tout renvoyer sous format JSON (lisible directement par javascript)

Et c'est là qu'entre la seconde partie de la MERN stack, React. Cette technologie va exécuter ces requêtes HTTP de manière invisible à l'utilisateur grâce à une technologie appelée AJAX.

Ainsi, on peut avoir des téléchargements de données dynamiques sans avoir à recharger la page, ce qui rends le site internet beaucoup plus rapide que la moyenne.

AJAX cache toutes les requêtes, l'utilisateur aura l'impression que les données auront toujours étaient là alors qu'elles viennent d'être téléchargées. Pensez-y, lorsque vous mettez un j'aime ou écrivez un commentaire sur un site, la page ne se recharge pas ? Et bien c'est grâce à AJAX.

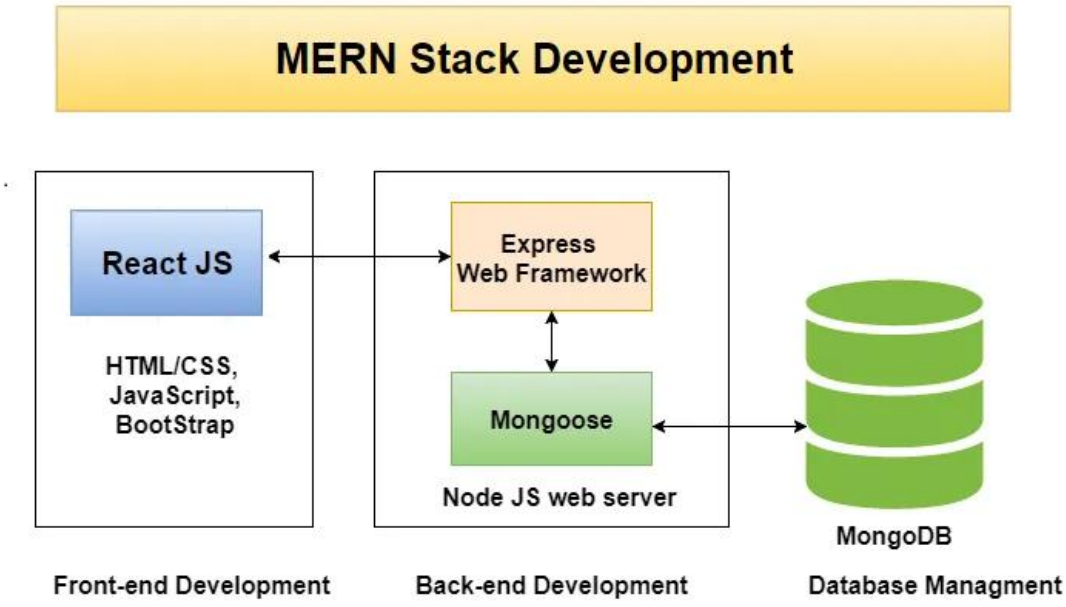


Figure 5: Schéma illustrant la MERN stack

Pourquoi utiliser la MERN stack ?

Après cette introduction un peu longue, je vais vous expliquer pourquoi je pense que la MERN stack est un excellent choix si vous voulez développer un site web de qualité. Après cette section, nous verrons les inconvénients, rien n'est jamais parfait.

Avantages sur le backend

Comme je l'ai dit, on peut dissocier la MERN stack en deux parties, le frontend et le backend. Toute la logique de votre base de données est totalement isolée de votre site web visible. Ce système implique une chose très importante, votre REST API est réutilisable pour une application mobile ou un logiciel très facilement.

Pensez-y, votre API ne change jamais, elle est la et ses url http ne dépendent pas de votre site visible. Il est donc possible de créer d'autres frontend, par exemple, il est possible de créer deux sites partageant le même système d'utilisateurs très facilement.

Node JS est extrêmement rapide. Le langage de programmation se base sur l'asynchronisme, il est donc très performant. Si votre backend est rapide, les utilisateurs recevront leurs données beaucoup plus rapidement, ce qui améliorera l'expérience utilisateur et réduira le coût de fonctionnement du serveur.

Avantages sur le frontend

React JS est aussi extrêmement impressionnant. C'est un système basé sur le SPA. C'est à dire qu'il n'y a qu'un seul document HTML et que les éléments se mettent à jour via Javascript, et ce sans aucun téléchargement.

Un site React JS est aussi performant que possible techniquement parlant. Les pages web utilisant ce système semblent être sans téléchargement si elles sont bien développées. Cela rend l'expérience utilisateur très appréciable, il peut naviguer sur le site sans temps d'attente.

Si vous voulez développer un site très performant facilitant la navigation des utilisateurs et donc la vente par exemple, React est un très bon choix.

Les désavantages de la MERN stack

Comme je l'ai dit, rien n'est parfait dans le monde de la programmation. Il y a des points qui font que la MERN stack peut être un mauvais choix. Je vais vous les présenter dans cette section afin de vous donner un avis complet sur la MERN stack.

- **Les désavantages du backend**

Comme je l'ai dit, Node JS utilise un système asynchrone. C'est une technologie permettant d'optimiser au maximum les performances d'un programme, en l'occurrence d'un site web.

Le problème, c'est que c'est une technologie assez complexe à maîtriser. Pour apprendre Node JS, il faut beaucoup travailler, bien plus que pour PHP par exemple.

Mongo DB possède aussi un inconvénient. C'est un système qui n'utilise pas le SQL, c'est donc un autre langage à apprendre. Beaucoup de développeurs maîtrisent SQL, ils y sont habitués et changer de système de base de données peut faire peur.

C'est néanmoins un système très performant si vous apprenez à le maîtriser, encore une fois c'est beaucoup de boulot.

Concrètement, il y a peu d'inconvenants technique à la MERN stack, c'est plutôt une question d'apprentissage et de maîtrise.

- **Les désavantages de React JS**

React JS possède quelques inconvenants technique. Plutôt des inconvenants d'adaptation. Contrairement aux sites internet normaux qui utilisent plusieurs documents HTML, ici on n'en a qu'un seul et on utilise un système de composants.

Il faut donc s'adapter, apprendre un nouveau moyen de développer sur le web. C'est un apprentissage long, vous allez totalement changer votre manière de réfléchir au développement web.

Enfin, React JS demande de manier les requêtes AJAX, c'est encore une fois un système complexe qui demande une réflexion importante.