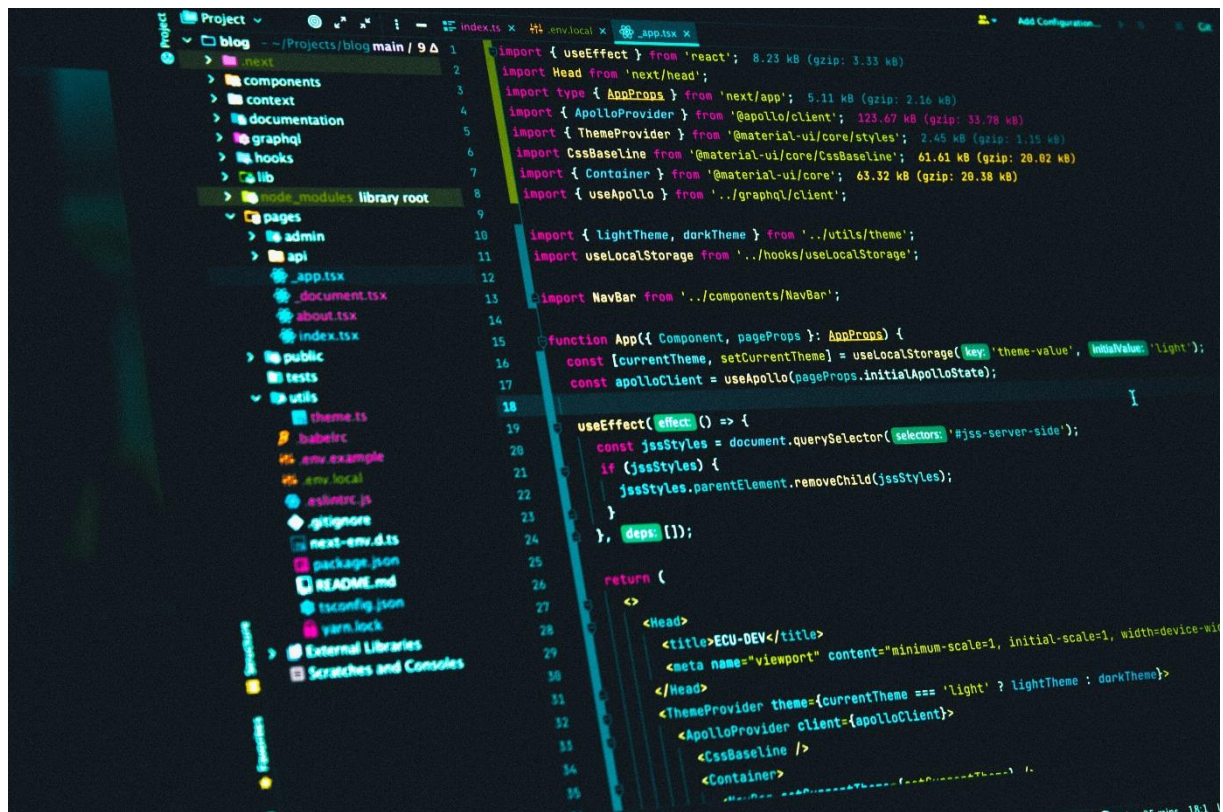


## Créer une API Rest

Projet - CDA



The image shows a code editor with a project structure on the left and the `_app.tsx` file open in the main editor. The project structure includes a `blog` directory with subdirectories like `components`, `context`, `documentation`, `graphql`, `hooks`, `lib`, `node_modules`, `pages`, `public`, `tests`, and `utils`. The `_app.tsx` file contains the following code:

```

1 import { useEffect } from 'react'; 8.23 kB (gzip: 3.33 kB)
2 import Head from 'next/head';
3 import type { AppProps } from 'next/app'; 5.11 kB (gzip: 2.16 kB)
4 import { ApolloProvider } from '@apollo/client'; 123.67 kB (gzip: 33.78 kB)
5 import { ThemeProvider } from '@material-ui/core/styles'; 2.45 kB (gzip: 1.15 kB)
6 import { CssBaseline } from '@material-ui/core/CssBaseline'; 61.61 kB (gzip: 20.02 kB)
7 import { Container } from '@material-ui/core'; 63.32 kB (gzip: 20.38 kB)
8 import { useApollo } from '../graphql/client';
9
10 import { LightTheme, darkTheme } from '../utils/theme';
11 import useLocalStorage from '../hooks/useLocalStorage';
12
13 import NavBar from '../components/NavBar';
14
15 function App({ Component, pageProps }: AppProps) {
16   const [currentTheme, setCurrentTheme] = useLocalStorage({ key: 'theme-value', initialValue: 'light' });
17   const apolloClient = useApollo(pageProps.initialApolloState);
18
19   useEffect(() => {
20     const jssStyles = document.querySelector(selectors: '#jss-server-side');
21     if (jssStyles) {
22       jssStyles.parentElement.removeChild(jssStyles);
23     }
24   }, []);
25
26   return (
27     <>
28     <Head>
29       <title>ECU-DEV</title>
30       <meta name="viewport" content="minimum-scale=1, initial-scale=1, width=device-width" />
31     </Head>
32     <ThemeProvider theme={currentTheme === 'light' ? LightTheme : darkTheme}>
33       <ApolloProvider client={apolloClient}>
34         <CssBaseline />
35         <Container>

```

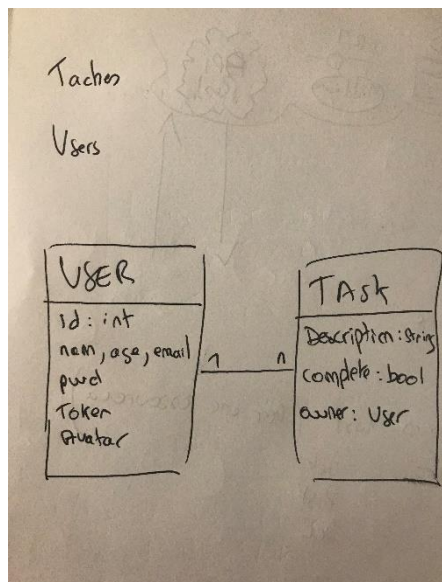
## Table des matières

Créer une API Rest.....	1
Consigne .....	3
Ressources .....	5
Corrigés.....	5
Exemples de tutos tuto :.....	6
Outils .....	6
Rappels théoriques.....	7
API REST.....	7
MVC .....	15
SGBD .....	16
SQL vs noSQL .....	16
NoSQL .....	18
MangoDB.....	19
MCD .....	19
ORM.....	20
Mongoose.....	20
Difference between MongoDB and Mongoose.....	21
NoSQL .....	21
Try-Catch.....	22
CallBack .....	22
Gitlab .....	23
Virtualisation .....	24
Sécurité.....	26
Tests.....	28
Divers.....	31

**Durée**  
20h

## Consigne

Individuellement ou en pair, à l'aide de vos connaissances et des éléments de synthèse. Développer une API Rest sur la thématique de votre choix, par défaut une Todo List.



## Objectif

L'objectif est de produire avec les bonnes pratiques de développement une API Rest.

Votre projet devra contenir à minima les éléments suivants :

- L'usage d'un ORM (ex : mongoose),
- La mise en place d'authentification (ex : JWT),
- Une page Gitlab documenté,
- Une page de documentation de l'API auto-généré (ex : apidoc),
- Un jeu de test unitaire (ex : jest, mocha, chai)

*Une veille et le partage d'information est fortement recommandé.*

Les CORS Policy doivent être géré. La prévention aux éléments de sécurité également (XSS/CSRF). Le chiffrement du mot de passe.

## Pour aller plus loin

- Une gestion des caches,
- multiples filtres sur l'API (pagination, sort)
- Import de fichiers (ex : Multer, filepond),
- Une mise en production (ex : render, heroku)
- Mise en place du CI/CD

**Recommandations techniques**

- Git,
- npm,
- Node,
- Express,
- MongoDB,
- Mongoose,

```
"dependencies": {  
  "apicache": "^1.6.3",  
  "bcrypt": "^5.0.1",  
  "body-parser": "^1.19.1",  
  "chalk": "^5.0.0",  
  "cookie-parser": "^1.4.6",  
  "cors": "^2.8.5",  
  "dotenv": "^16.0.1",  
  "express": "^4.17.2",  
  "jsonwebtoken": "^8.5.1",  
  "mongoose": "^6.1.6",  
  "multer": "^1.4.5-lts.1",  
  "nodemon": "^2.0.18",  
  "sharp": "^0.31.3",  
  "validator": "^13.7.0"  
},  
"devDependencies": {  
  "chai": "^4.3.6",  
  "chai-http": "^4.3.0",  
  "mocha": "^10.0.0"  
}  
}
```

Figure 1- Exemple de package npm

## Ressources

## Corrigés

**Code source de base d'un API Rest sans authentication :**

<https://mega.nz/file/I9wVkdRK#XaoAhpHccUK0jzjYAXHAmLub4E-oD4ZPLLXz1h4WsQ>

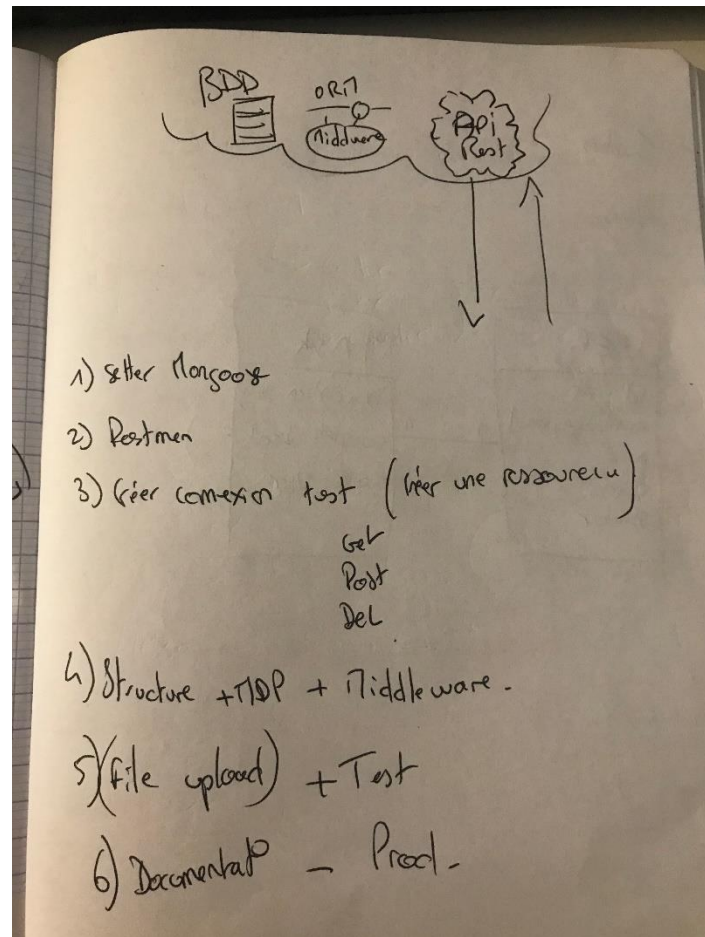


Figure 2- Marche à suivre

Exemples de tutos tuto :

**10. MongoDB and Promises (Task App)**

<https://mega.nz/folder/soliwAKQ#fveG2IDytsNJW5hUjmyoiQ>

**11. REST APIs and Mongoose (Task App)**

<https://mega.nz/folder/RtRm0LhC#-HBTQbj071L13sjF4KUgjQ>

**12. API Authentication and Security (Task App)**

[https://mega.nz/folder/904GGI5C#RP0PBxBdqBOMm\\_K87j-F5A](https://mega.nz/folder/904GGI5C#RP0PBxBdqBOMm_K87j-F5A)

**13. Sorting, Pagination, and Filtering (Task App)**

[https://mega.nz/folder/MoZgFAwS#6fgR0\\_Mg0awpQ56k-jtQfA](https://mega.nz/folder/MoZgFAwS#6fgR0_Mg0awpQ56k-jtQfA)

**14. File Uploads (Task App)**

<https://mega.nz/folder/YxhkEabK#nINfAp-5lx0xAxBWNRIJRw>

**16. Testing Node.js (Task App)**

<https://mega.nz/folder/lgg0RQSC#hsfXakuTzvK6C93DH8BJ9g>

Outils

**MongoDBCompass** : app desktop pour gérer ses bases de données

<https://www.mongodb.com/products/compass>

**MongoDBAtlas** : Pour créer sa bdd dans le cloud

<https://www.mongodb.com/fr-fr/atlas/database>

--

**Gitlab** : Documentation / CI - CD

<https://gitlab.com/>

**Render** : Pour déployer

<https://render.com/>

--

**Nodemon** : pour le live reload avec nodejs

<https://www.npmjs.com/package/nodemon>

**Postman** : Postman est un logiciel qui va vous permettre d'appeler / tester une API.

<https://www.eewee.fr/postman-cest-quoi/>

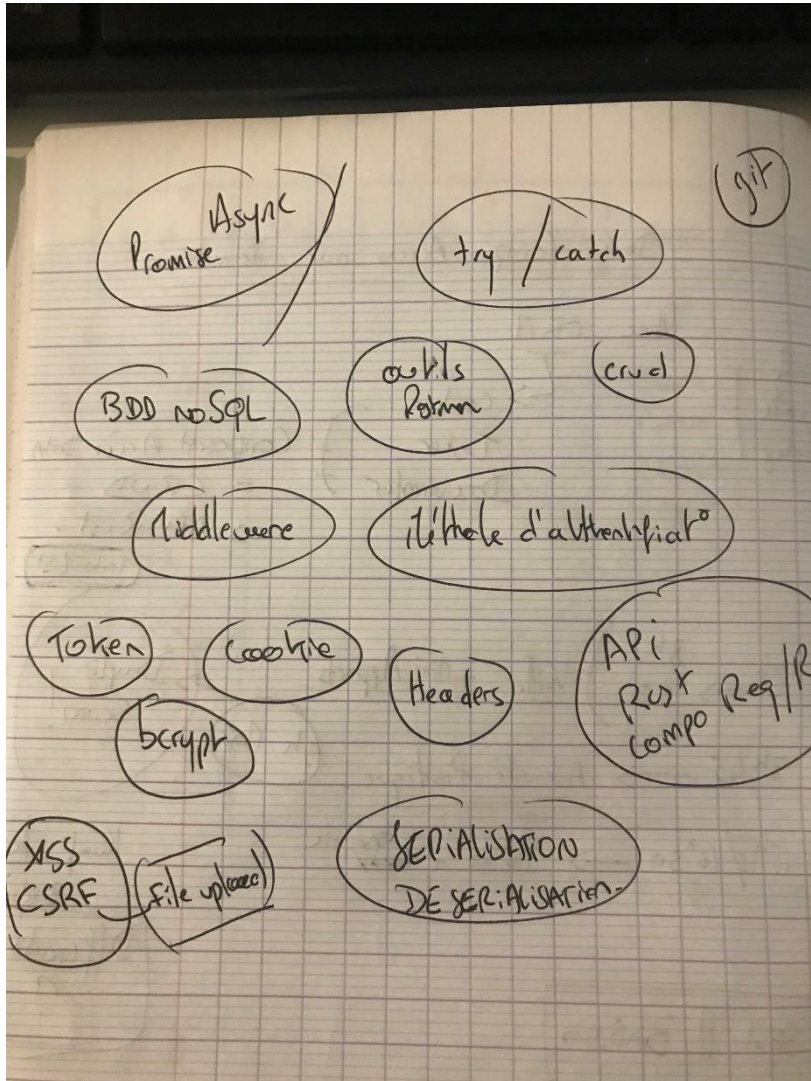
<https://youtu.be/pUbrKIdUhjo>

<https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6818136-definisiez-des-requetes-et-reponses-typiques>

Alternative : *Insomnia*



## Rappels théoriques



## API REST

REST (Representational state transfer)

REST est un style d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer des services web. Les services web conformes au style d'architecture REST, aussi appelés services web RESTful, établissent une interopérabilité entre les ordinateurs sur Internet

Les données REST peuvent être en langage JSON ou XML, mais le JSON est le plus courant.

<https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web?archived-source=3449001>

<https://www.illustradata.com/api-rest/>

<https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6818136-definissez-des-requetes-et-reponses-typiques>

## Vocabulaire

### Ressources

Les données REST sont représentées dans ce qu'on appelle des ressources. Une ressource peut être tout type d'objet nominal (on lui attribue un nom)  
voyez les ressources comme des boîtes dans lesquelles vous rangerez des objets par catégorie et sur lesquelles vous collez une étiquette pour savoir quoi mettre dedans.

### Collection

Les ressources sont regroupées dans un groupe que l'on appelle une collection. On s'y réfère avec la forme au **pluriel** du nom de la ressource. Par exemple une ressource superhero donnerait superheroes

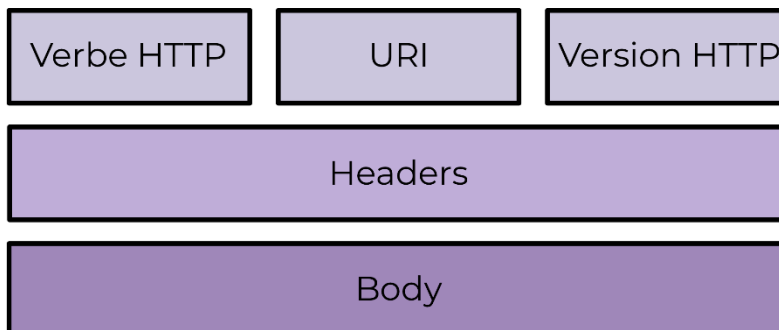
### URL / URI / EndPoint

On confond souvent les deux. On va tout simplifier avec un exemple ! Reprenons notre site de Game of Thrones. Si le personnage de Jon Snow a pour ID 890, alors l'URI serait /characters/890. L'URL serait <https://gameofthrones-informations.com/characters/890>

L'URL de la requête est l'endpoint complet que vous utilisez pour votre requête. Il associe le nom de domaine + le path de votre ressource. À présent, vous savez comment accéder aux données que vous souhaitez !

## Structure d'une requête

Verbe HTTP + URI + Version HTTP + Headers + Body (facultatif)

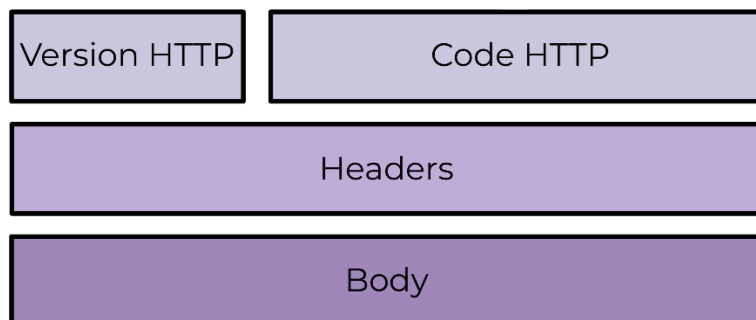


Verbes HTTP	
• GET	↳ Obtenir
• PUT	↳ Mettre
• POST	↳ Publier
• DELETE	↳ Supprimer



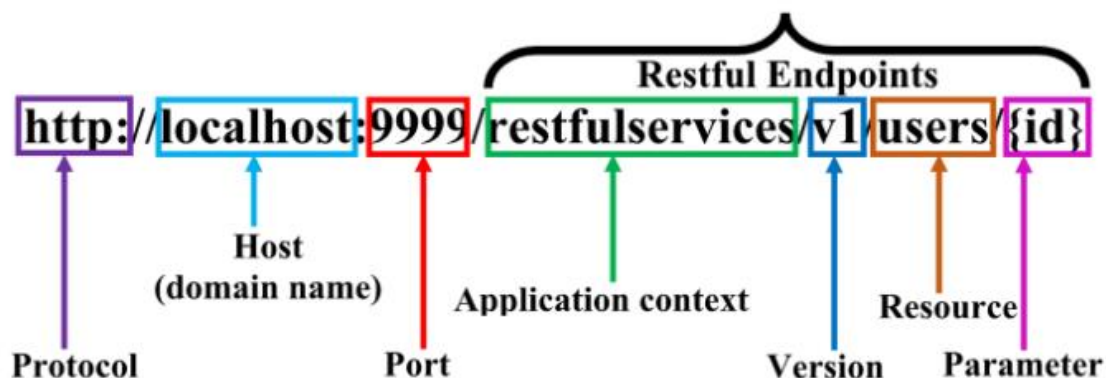
### Structure d'une réponse

Version HTTP + Code de réponse HTTP + Headers + Body



### Code réponse :

Les codes de réponse HTTP sont des sortes de feux de signalisation 🚦 avec des codes spécifiques, pour informer les clients si la requête est un succès ou un échec



### Code status

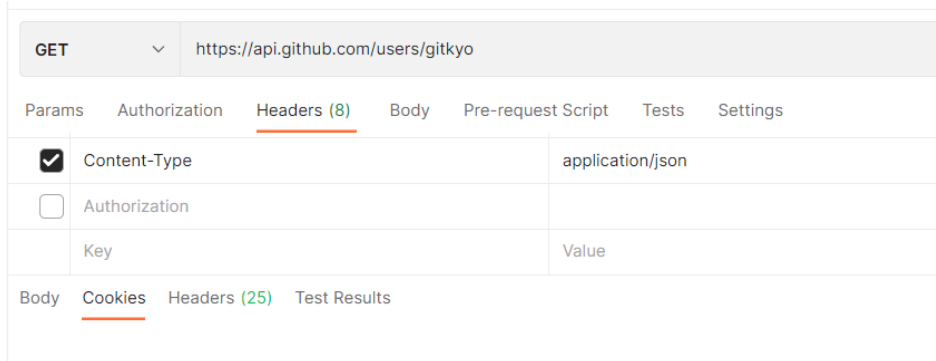
Code	Verbes	Type	Description
200	GET, POST, PUT, DELETE	Ressource	Succès
201	POST, PUT	Ressource	Création avec succès
202	DELETE	-	La requête est acceptée
204	POST, PUT, DELETE	-	Succès sans réponse
400	GET, POST, PUT, DELETE	Erreur	Mauvaise requête
401	GET, POST, PUT, DELETE	Erreur	Mauvaise authentification
403	GET, POST, PUT, DELETE	Accès interdit	Accès interdit
404	GET, POST, PUT, DELETE	Erreur	Ressource introuvable
500	GET, POST, PUT, DELETE	Erreur	Erreur de serveur interne
503	GET, POST, PUT, DELETE	Erreur	Service indisponible

<https://httpstatusdogs.com/>

### En-tête (Headers)

Les en-têtes HTTP permettent au client et au serveur de transmettre des informations supplémentaires avec la requête ou la réponse. Un en-tête de requête est constitué de son nom (insensible à la casse) suivi d'un deux-points :, puis de sa valeur (sans saut de ligne). L'espace blanc avant la valeur est ignoré.

ex d'usage sur Postman :



<https://developer.mozilla.org/fr/docs/Web/HTTP/Headers>

Les en-têtes peuvent être groupés selon leur contexte :

- [En-tête général](#) : en-têtes s'appliquant à la fois aux requêtes et aux réponses mais sans rapport avec les données éventuellement transmises dans le corps de la requête ou de la réponse.
- [En-tête de requête](#) : en-têtes contenant plus d'informations au sujet de la ressource à aller chercher ou à propos du client lui-même.
- [En-tête de réponse](#) : en-têtes contenant des informations additionnelles au sujet de la réponse comme son emplacement, ou au sujet du serveur lui-même (nom et version, etc.)
- [En-tête d'entité](#) : en-têtes contenant plus d'informations au sujet du corps de l'entité comme la longueur de son contenu ou son [type MIME](#).

### CORS

CORS (Partage de ressource cross-origin) est un mécanisme qui consiste à transmettre des entêtes HTTP qui déterminent s'il faut ou non bloquer les requêtes à des ressources restreintes sur une page web qui se trouve sur un domaine externe au domaine dont la ressource est originaire.

<https://developer.mozilla.org/fr/docs/Glossary/CORS>

Outil pour les en tête nodejs : <https://helmetjs.github.io/>

Ou <https://www.npmjs.com/package/cors>

```

48
49 // Add headers before the routes are defined
50 import cors from 'cors'
51 app.use(cors())
52

```

*Verbes http*

Action CRUD	Verbe HTTP associé
Create (Créer)	POST (Publier)
Read (Lire)	GET (Obtenir)
Update (Mettre à jour)	PUT (Mettre)
Delete (Supprimer)	DELETE (Supprimer)

*Protocole http*

Histoire : [https://developer.mozilla.org/fr/docs/Web/HTTP/Basics\\_of\\_HTTP/Evolution\\_of\\_HTTP](https://developer.mozilla.org/fr/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP)

**La v2 :**

Le protocole HTTP/2 diffère de HTTP/1.1 sur plusieurs aspects:

- Il est encodé en binaire plutôt qu'en texte. Il ne peut donc plus être lu ou écrit à la main. Malgré cette difficulté, il est désormais possible d'implémenter des techniques d'optimisation avancée.
- C'est un protocole multiplexé. Plusieurs requêtes en parallèle peuvent être gérées au sein de la même connexion, supprimant ainsi la limitation séquentielle de HTTP/1.x.
- HTTP/2 compresse les en-têtes, étant donné que des en-têtes similaires sont échangés lors d'une suite de requêtes, on supprime ainsi la duplication et l'échange inutiles des données similaires.

Il permet au serveur de remplir le cache du client avant qu'il ne soit demandé par ce dernier, on parle alors d'événements générés par le serveur.

Devenu un standard officiel en mai 2015, HTTP/2 a rencontré un large succès

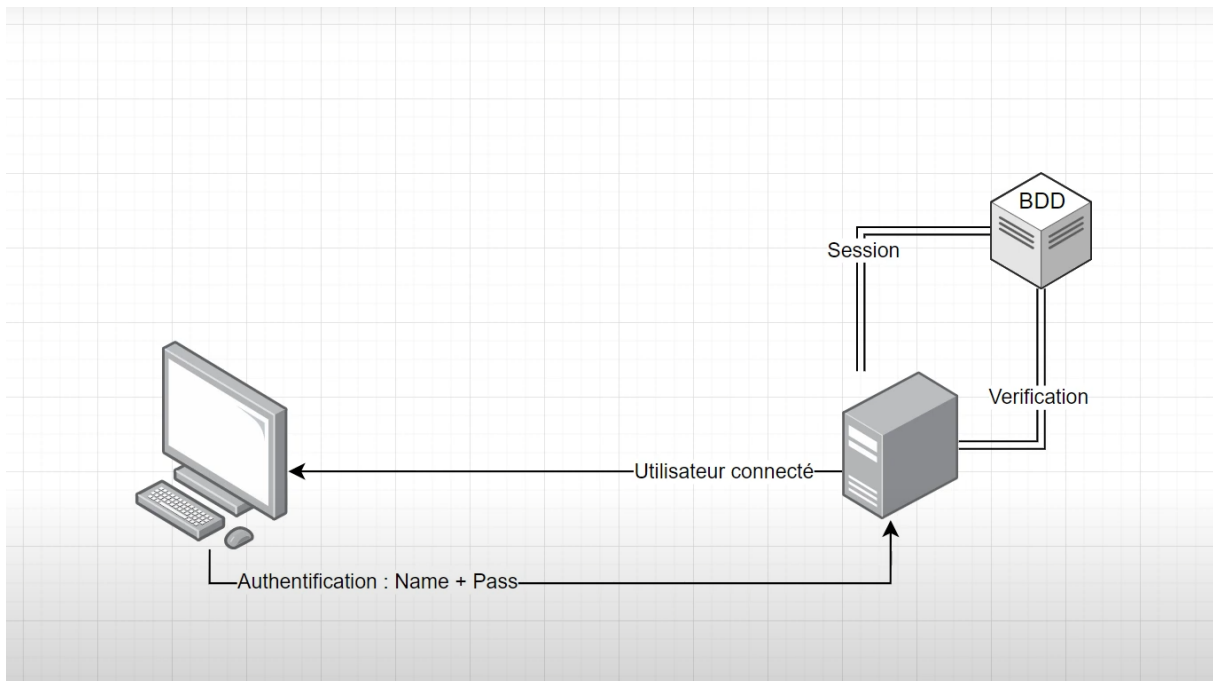
**V3 :**

HTTP/3 est la troisième version du protocole HTTP (Hypertext Transfer Protocol), anciennement HTTP over-QUIC. QUIC (Quick UDP Internet Connections) a été initialement développé par Google et est le successeur de HTTP/2. Des entreprises comme Google et Facebook utilisent déjà QUIC pour accélérer le Web.

<https://kinsta.com/fr/blog/http3/>

## Authentification

En pratique, trois techniques sont principalement utilisées pour sécuriser une API:



HTTP Basic Authentication

Oauth

OpenID

Une des méthodes d'authentification les plus utilisées consiste à exiger qu'un développeur s'inscrive par le site web de l'API pour obtenir un token ou clé (Le token est un peu comme un numéro de passeport : il est unique et permet de vous identifier).

### - Comment ajouter le token à la requête ?

On l'envoie soit dans les paramètres du header, soit dans l'endpoint lui-même. Ajoutez-le dans l'onglet Headers sous forme de clé-valeur où la clé sera Authorization et la valeur sera Token, votre token. Si mon token est abcde, alors la valeur sera : token abcde.

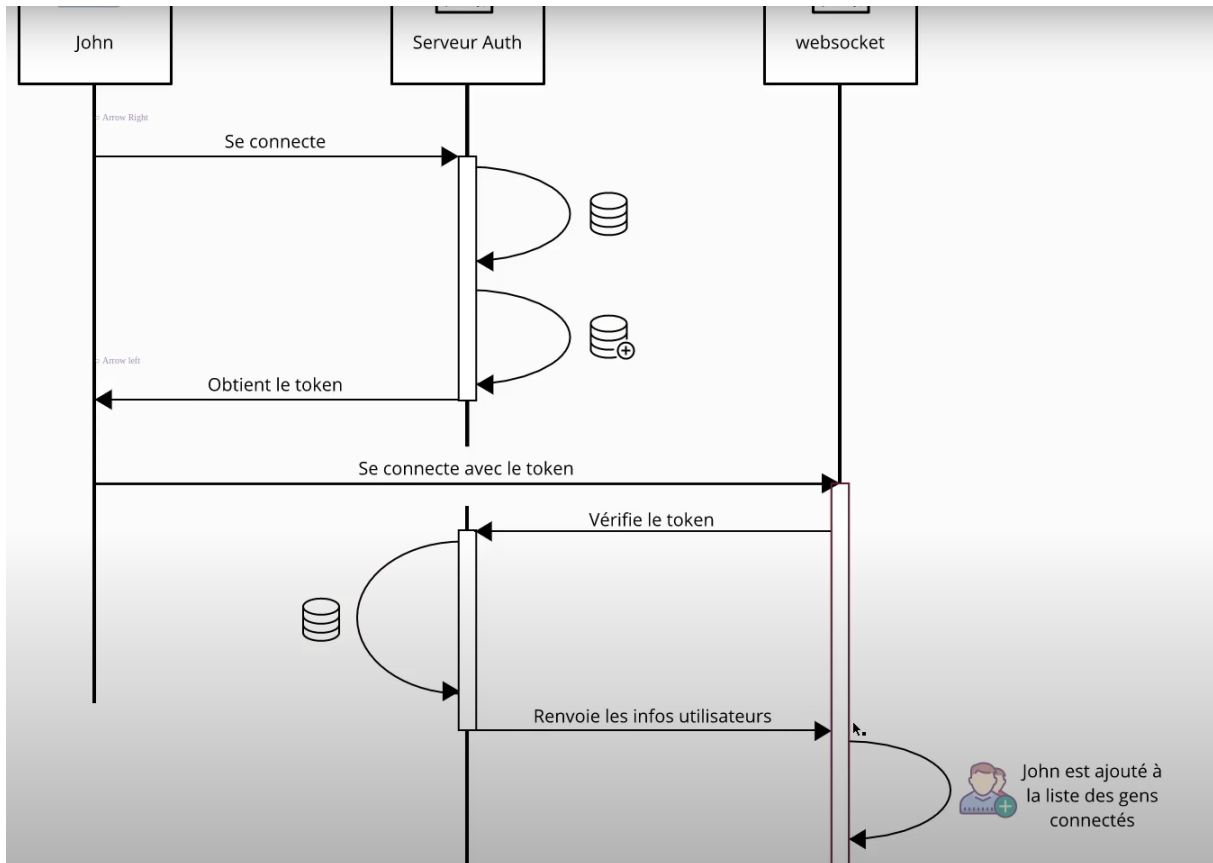
### HTTP Basic Authentication

Il s'agit de la solution la plus simple. Consiste à envoyer le login et le mot de passe dans l'entête de chaque requête.

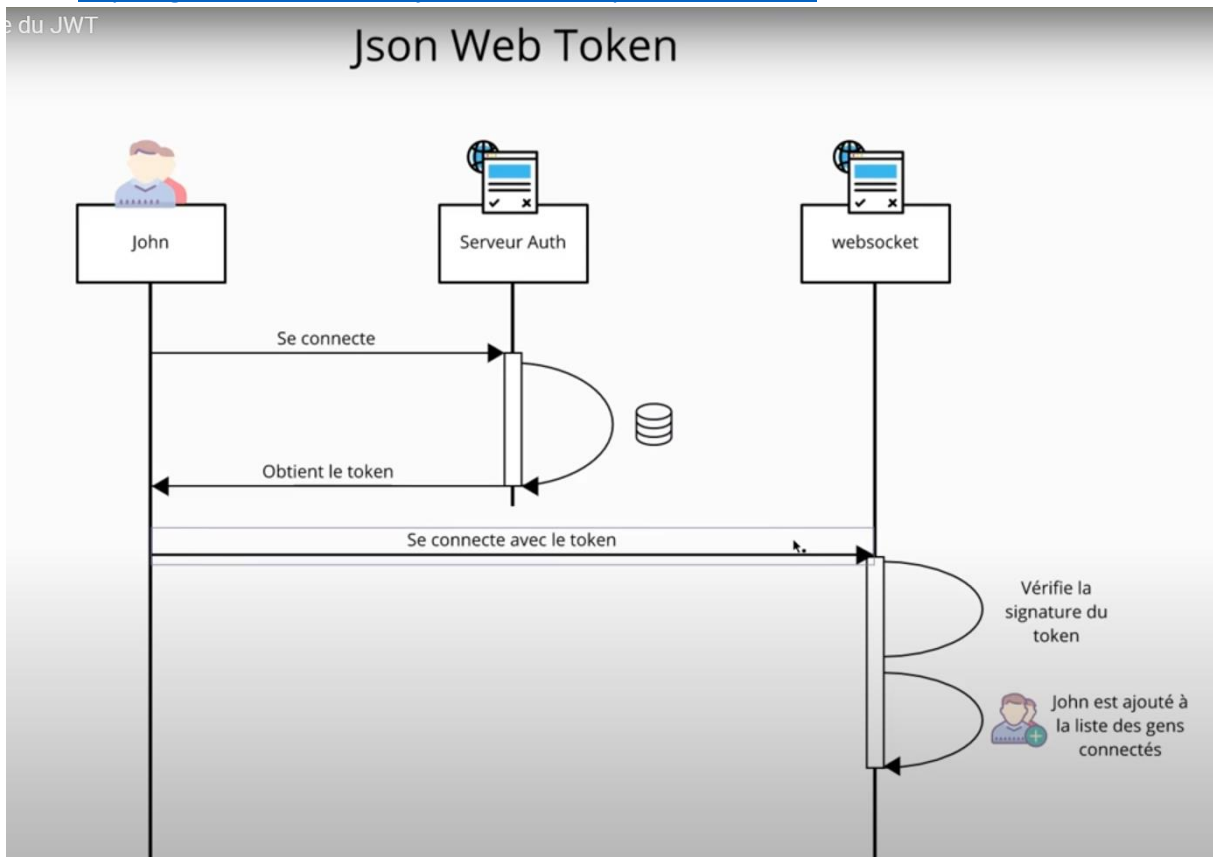
Bien évidemment, il est fortement conseillé d'utiliser le protocole HTTP en mode chiffré (HTTPS) pour ne pas circuler le mot de passe en clair dans les trames réseaux (faille aux attaques de type MITM).

Pour plus de sécurité, surtout si le client de l'API est à l'extérieur de votre réseau, il est préconisé d'utiliser un jeton de sécurité JWT (Json Web Token). Le JWT devrait avoir une durée de vie (TTL) limitée.

Token Traditionnel :



JWT : <https://grafikart.fr/tutoriels/json-web-token-presentation-958>



## Oauth

Oauth est un protocole de délégation d'autorisation nécessitant un serveur tiers comme fournisseur d'accès.

Malgré la complexité de sa mise en place, ce protocole est très apprécié pour sa sécurité.

Il existe deux versions:

**Oauth 1.0** – Il s'agit de la version la plus dure à mettre en place car nécessite un partage de clé entre client et serveur pour le calcul d'une signature (même principe que SSL). Cette version est recommandée pour les APIs sans HTTPS.

**Oauth 2.0** – Cette version est plus aisée à mettre en place car ne dispose pas de calcul de signature. Par contre, en l'occurrence le HTTPS est exigé. Néanmoins, cette version est moins sécurisée que Oauth 1.0.

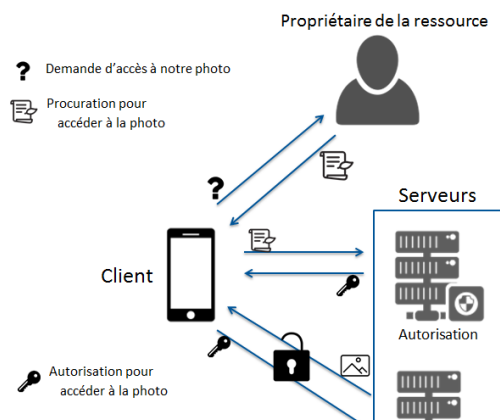
### Découvrez OAuth 2.0



Ce protocole, plébiscité par les géants du Web **comme Google, Facebook** et Microsoft, décrit comment un client peut **obtenir un accès à une ressource détenue par un tiers**.

Le protocole définit 4 rôles :

- Le **détenteur** des données (vous) ;
- Le **serveur de ressources** qui héberge les accès dont l'accès est protégé (les informations de votre profil Facebook par exemple) ;
- Le **client** qui souhaite accéder au serveur de ressources ;
- Le **serveur d'autorisation** qui délivre des jetons au client.



## OpenID

De base, OpenID est un système d'identification en mode SSO. Il permet à un client de se connecter auprès de plusieurs sites sans devoir créer un compte à chaque fois.

Le mode de fonctionnement ressemble à celui de Oauth. Il faut un fournisseur d'identité (OpenID providers) pour établir un lien de confiance entre le client et le serveur. Souvent, le fournisseur d'identité est un grand nom du Web (facebook, twitter, google, ...).

D'autre part, cette solution ne permet pas de gérer l'authentification. De ce fait, il existe une deuxième version (OpenID Connect) qui intègre une sous couche Oauth.

<https://developer.mozilla.org/fr/docs/Web/HTTP/Authentication>

[http://www.standard-du-web.com/http\\_authentication.php](http://www.standard-du-web.com/http_authentication.php)

<https://www.pierre-giraud.com/http-reseau-securite-cours/authentification/>

<https://openclassrooms.com/fr/courses/1761931-securisez-vos-applications/5702551-securisez-vos-communications-avec-tls>

<https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et->

[mysql/913196-implementez-un-systeme-de-connexion](https://blog.ippon.fr/2017/10/12/preuve-dauthentification-avec-jwt/)  
<https://blog.ippon.fr/2017/10/12/preuve-dauthentification-avec-jwt/>  
<https://blog.logrocket.com/implement-oauth-2-0-node-js/>  
<https://www.youtube.com/watch?v=I5tFIK5PPjc>

*Sécuriser une API*

**Veille cf plus bas**

<https://www.pingidentity.com/fr/resources/blog/post/complete-guide-to-api-security.html#Quest-ce-que-la-scurit-des-API->

*Optimiser*

**Mettre en cache une API**

<https://medium.com/@muhammadtaifkhan/cache-your-express-nodejs-api-1e9f80b87c88>

**Exemple d'API Rest php**

<https://github.com/mevdschee/php-crud-api>

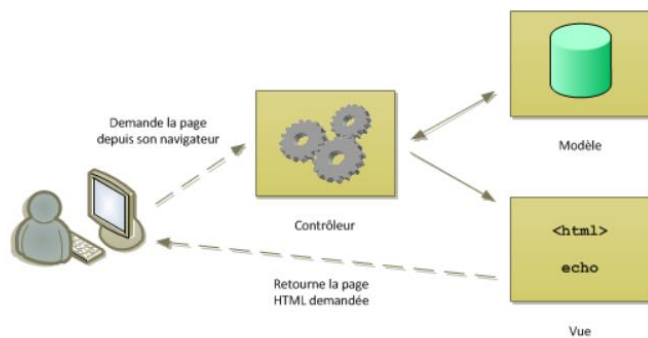
MVC

Le pattern MVC est une façon d'organiser un code source autour de trois piliers :

- **Le Model** qui représente la structure des données. Leur définition ainsi que les fonctions qui leur sont propres et qu'elles peuvent avoir. Ce module est complètement décollé du code métier ou de l'affichage de telle sorte à ce que la modification de la logique ou de l'interface n'affecte pas la structure des données.  
**Modèle** : cette partie gère les données de votre site. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc entre autres les requêtes SQL.
- **La View** (ou les vues) représente l'interface graphique à livrer au client qui en fait la requête. Avoir le code lié à l'interface isolé de la logique métier ou des données permet de faire des modifications à l'interface graphique sans avoir à se soucier de casser du code métier ou la structure des données.  
**La vue**, cette partie se concentre sur l'affichage. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples, pour afficher par exemple une liste de messages.
- **Le(s) Controller(s)** sont au cœur de la logique métier de votre application. Ils se situent entre les vues et le model. Les requêtes qu'un client va faire depuis l'interface graphique, la view, va être dirigée vers un controller qui sera en charge de manipuler les données dont il a besoin avec la brique Model, la traiter suivant le besoin métier, puis ordonner à la view de répondre au client avec les bons éléments.  
**Controller**, cette partie gère la logique du code qui prend des décisions. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. Le



contrôleur contient exclusivement du PHP. C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d'accès).



La requête du client arrive au contrôleur et celui-ci lui retourne la vue

<https://youtu.be/DUg2SWWK18I>

<https://practicalprogramming.fr/mvc>

<https://openclassrooms.com/fr/courses/4670706-adoptez-une-architecture-mvc-en-php/4678736-comment-fonctionne-une-architecture-mvc>

## SGBD

Un système de gestion de base de données est un logiciel système servant à stocker, à manipuler ou gérer, et à partager des données dans une base de données

[https://fr.wikipedia.org/wiki/Syst%C3%A8me\\_de\\_gestion\\_de\\_base\\_de\\_donn%C3%A9es](https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_gestion_de_base_de_donn%C3%A9es)

### Types de SGBD

<https://web.maths.unsw.edu.au/~lafaye/CCM/bdd/bddtypes.htm>

<https://www.base-de-donnees.com/comprendre-bases-de-donnees/les-4-types-de-bases-de-donnees/>

### Les caractéristiques d'un SGBD

<https://web.maths.unsw.edu.au/~lafaye/CCM/bdd/bddansi.htm>

## SQL vs noSQL



Les bases de données SQL représentent des données sous la forme de tables composées de n nombre de lignes de données, tandis que les bases de données NoSQL sont la collection de paires clé-valeur, de documents, de bases de données graphiques, etc. qui ne possèdent pas de définitions de schéma standard.

**Exemples de bases de données SQL:** MySQL, Oracle, Sqlite, Postgres et MS-SQL.

**Exemples de bases de données NoSQL:** MongoDB, BigTable, Redis, RavenDb, Cassandra, Hbase, Neo4j et CouchDb

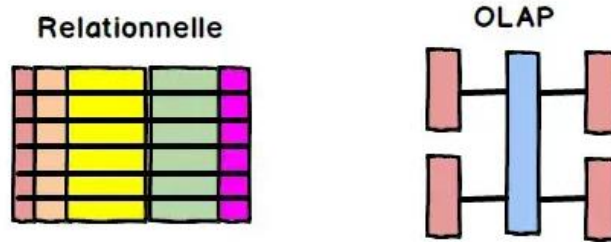
<https://waytolearnx.com/2019/03/difference-entre-sql-et-nosql.html>

Key Areas	MySQL	MongoDB
<b>Query Language</b>	Uses Structured Query Language(SQL)	Uses MongoDB Query Language
<b>Flexibility of Schema</b>	Pre-defined schema design	No restrictions on schema design
<b>Relationships</b>	Supports JOIN statements	Does not support JOIN statements
<b>Security</b>	Uses privilege-security based model	Uses role-based access control
<b>Performance</b>	Slower than MongoDB	Faster than MySQL
<b>Support</b>	Provides excellent support 24*7	Provides excellent support 24*7
<b>Key Features</b>	<ul style="list-style-type: none"> <li>•Triggers &amp; SSL Support</li> <li>•Provides text searching and indexing</li> <li>•Query caching</li> <li>•Integrated replication support</li> <li>•Different storage engines with various</li> </ul>	<ul style="list-style-type: none"> <li>•Auto-sharding</li> <li>•Comprehensive secondary indexes</li> <li>•In-memory speed</li> <li>•Native replication</li> <li>•Embedded data models support</li> </ul>
<b>Replication</b>	Supports Master-Slave Replication	Supports built-in replication, sharding, and auto-elections.
<b>Usage</b>	<ul style="list-style-type: none"> <li>•Best fit for data with tables and rows</li> <li>•Works better for small datasets</li> <li>•Frequent updates</li> <li>•Strong dependency on multi-row transactions</li> <li>•Modify large volume of records</li> </ul>	<ul style="list-style-type: none"> <li>•Best fit for unstructured data</li> <li>•Works better for large datasets</li> <li>•High write loads</li> <li>•High availability in an unstable environment</li> <li>•Data is location-based</li> </ul>
<b>Active Community</b>	Has a good active community.	The community of MySQL is much better than that of MongoDB.

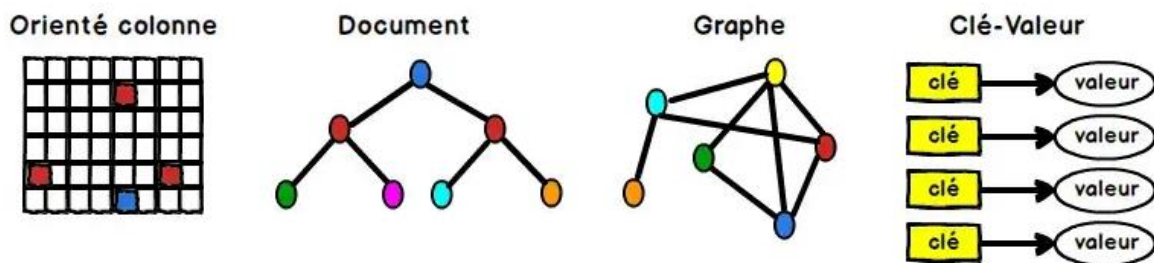
<https://medium.com/edureka/sql-vs-nosql-db-5d9b69ace6ac#:~:text=The%20SQL%20databases%20have%20a,varies%20from%20database%20to%20database.>

## Base de données SQL

WayToLearnX



## Base de données NoSQL



### Types de bases de données NoSQL

Il existe principalement quatre catégories de bases de données NoSQL. Chacune de ces catégories a ses attributs et ses limites. Aucune base de données spécifique n'est préférable pour résoudre tous les problèmes. Vous devez sélectionner une base de données selon les besoins de votre projet. Voici les quatre types de bases de données NoSQL :

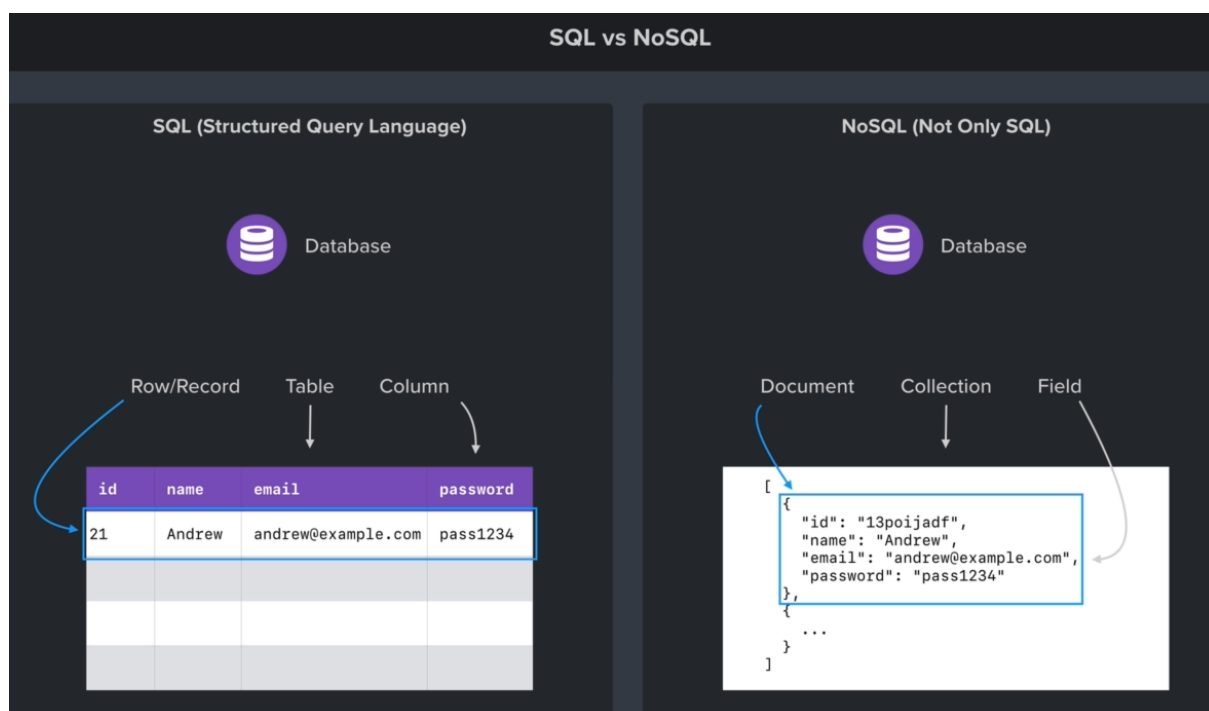
- Orienté clé-valeur
- Orienté colonne
- Orienté graphes
- Orienté document

<https://waytolearnx.com/2019/10/quest-ce-quune-base-nosql.html>

### NoSQL

Le NoSQL est un type de bases de données, dont la spécificité est d'être non relationnelles. Ces systèmes permettent le stockage et l'analyse du Big Data. ... Ainsi, NoSQL est utilisé pour le Big Data et les applications web en temps réel..

<https://datascientest.com/nosql>



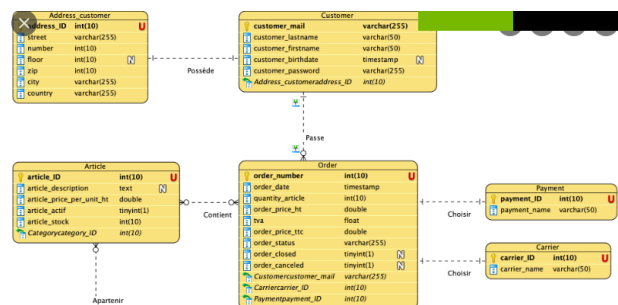
## MongoDB

Mongo est la base de données NoSQL.

MongoDB est un système de gestion de base de données orienté documents, répartitionnable sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données. Il est écrit en C++

<https://www.mongodb.com/fr-fr/what-is-mongodb>

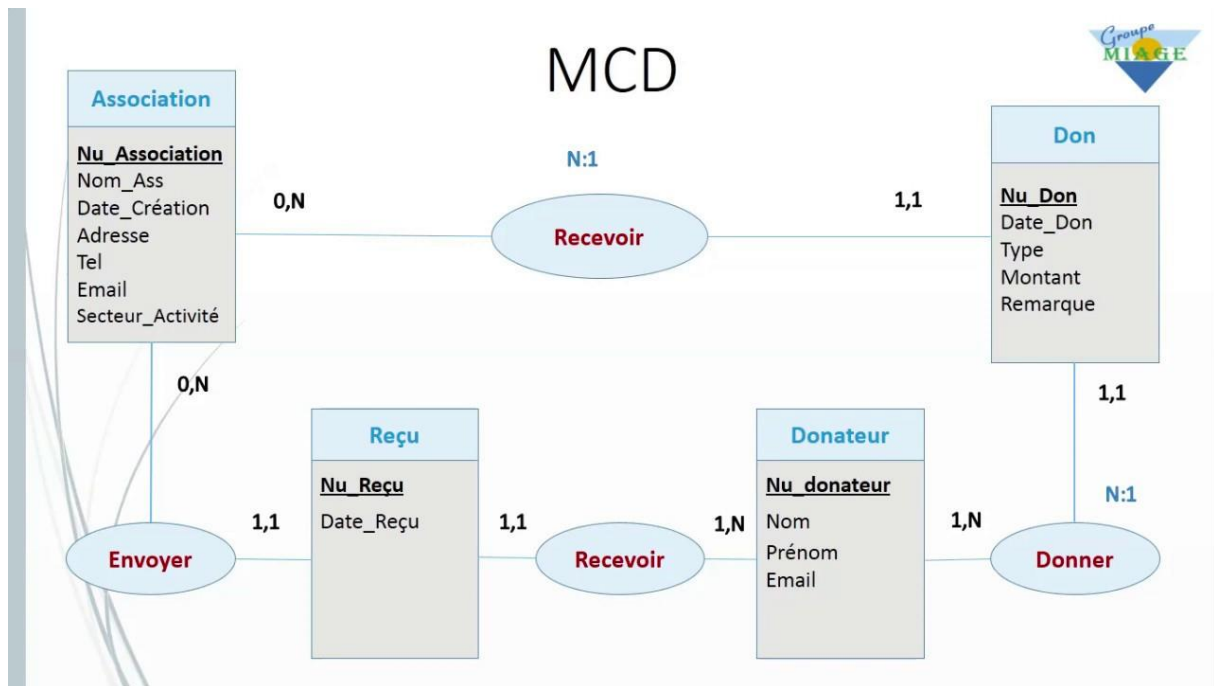
## MCD



Le MCD est une représentation graphique de haut niveau qui permet facilement et simplement de comprendre comment les différents éléments sont liés entre eux.

<https://www.base-de-donnees.com/mcd/>

<http://tony3d3.free.fr/files/Les-Cardinalites.pdf>



le logiciel génère automatiquement l'UML et le MLD à partir du MCD à partir du moment où les associations entre entités sont bien définies

<https://www.looping-mcd.fr/#>

## ORM

Il s'agit d'une technique de programmation informatique qui permet de simplifier l'accès à une base de données en proposant à l'informaticien des « objets » plutôt que d'accéder directement à des données relationnelles.

<https://www.base-de-donnees.com/orm/>

ORM PHP :

- Doctrine (symfony)
- Eloquent (laravel)

ORM nodejs :

<https://www.prisma.io/dataguide/database-tools/top-nodejs-orms-query-builders-and-database-libraries#mongoose>

## Mongoose

Mongoose est l'un des orms qui nous offre la possibilité d'accéder aux données Mongo avec des requêtes faciles à comprendre.

Mongoose joue un rôle d'abstraction sur votre modèle de base de données

<https://mongoosejs.com/>

Alternative pour typescript :

<https://www.prisma.io/>

## Difference between MongoDB and Mongoose

▲ I assume you already know that MongoDB is a NoSQL database system which stores data in the form of BSON documents. Your question, however is about the packages for Node.js.

333

▼ In terms of Node.js, [mongodb](#) is the **native driver** for interacting with a mongodb instance and [mongoose](#) is an **Object modeling tool** for MongoDB.

✓ `mongoose` is built on top of the `mongodb` driver to provide programmers with a way to model their data.



**EDIT:** I do not want to comment on which is better, as this would make this answer opinionated. However I will list some advantages and disadvantages of using both approaches.

Using `mongoose`, a user can define the schema for the documents in a particular collection. It provides a lot of convenience in the creation and management of data in MongoDB. On the downside, learning mongoose can take some time, and has some limitations in handling schemas that are quite complex.

However, if your collection schema is unpredictable, or you want a Mongo-shell like experience inside Node.js, then go ahead and use the `mongodb` driver. It is the simplest to pick up. The downside here is that you will have to write larger amounts of code for validating the data, and the risk of errors is higher.

<https://www.it-swarm-fr.com/fr/node.js/difference-entre-mongodb-et-mongoose/1051315508/>

## NoSQL

Le NoSQL est un type de bases de données, dont la spécificité est d'être non relationnelles. Ces systèmes permettent le stockage et l'analyse du Big Data. ... Ainsi, NoSQL est utilisé pour le Big Data et les applications web en temps réel..

<https://datascientest.com/nosql>

## Try-Catch

Un programme peut être confronté à une condition exceptionnelle (ou exception) durant son exécution.

Une exception est une situation qui empêche l'exécution normale du programme (elle ne doit pas être considérée comme un bug). Quelques exemples de situations exceptionnelles :

- un fichier nécessaire à l'exécution du programme n'existe pas ;
- division par zéro ;
- débordement dans un tableau ;
- etc.

Ps : il est possible d'étendre la class exception pour personnaliser ses messages d'erreurs.

on peut également catcher les Exception en fonctions de leurs types

En Java, une exception est concrétisée par une instance d'une classe qui étend la classe Exception.

Pour lever (déclencher) une exception, on utilise le mot-clé throw :

```
if (problem) throw new MyException("error");
```

Pour capturer une exception, on utilise la syntaxe try/catch :

```
try { /* Problème possible */ }
catch (MyException e) { /* traiter l'exception. */ }
```

Le bloc associé au mot-clé finally est toujours exécuté :

## CallBack

Une fonction de rappel (aussi appelée callback en anglais) est une fonction passée dans une autre fonction en tant qu'argument, qui est ensuite invoquée à l'intérieur de la fonction externe pour accomplir une sorte de routine ou d'action.

[https://developer.mozilla.org/fr/docs/Glossary/Callback\\_function](https://developer.mozilla.org/fr/docs/Glossary/Callback_function)

```
function salutation(name) {
    alert('Bonjour ' + name);
}

function processUserInput(callback) {
    var name = prompt('Entrez votre nom. ');
    callback(name);
}

processUserInput(salutation);
```