Introduction à Node.js

Said Gounane
2019/2020

Sommaire

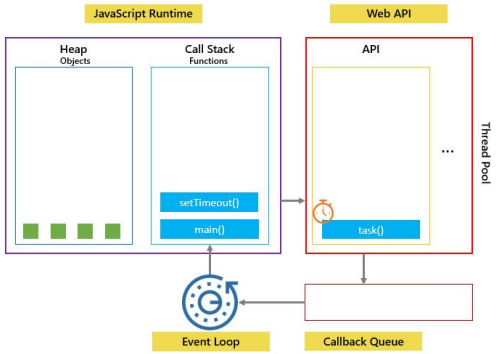
C’est Quoi Node.js

1. Node.je est un environnement côté serveur, open source
2. Node.js est multiplatform
3. Node.js utilise du javascript côté serveur

Pourquoi Node.js

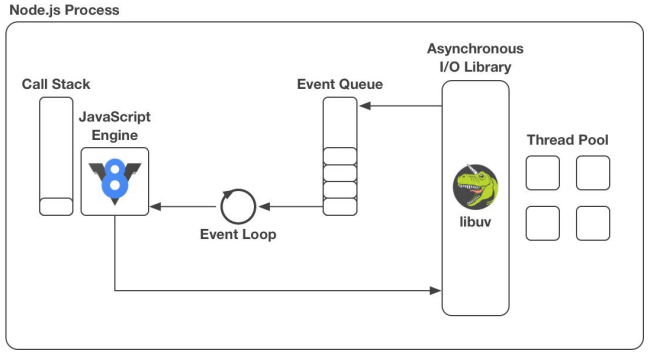
- Node.js utilise une programmation ***Asynchrone***
- Node.js est mono-Thread non-blocking.
- Un programme Node.js exécute les instructions sans arrêt et garantit un fonctionnement non bloquant.
- Node.js utilise le moteur d'exécution ultrarapide V8 de Google Chrome.
- Ce moteur est caractérisé par la compilation JIT (Just In Time). Il transforme le code JavaScript très rapidement en code machine.

Principe (js dans le navigateur)



https://www.javascripttutorial.net/javascript-event-loop/

Principe (node.js coté serveur)



https://itnext.io/javascript-promises-with-node-js-e8ca827e0ea3

Pourquoi Node.js

Pour lire un fichier dans le serveur avec:

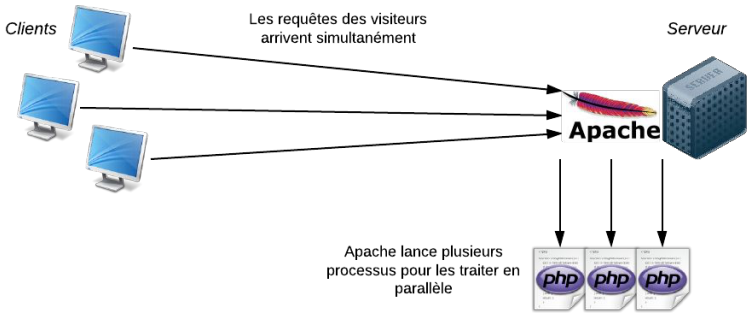
PHP

1. Envoyer la requête au serveur
2. Demander au système de fichier d'ouvrire le fichier
3. **Attendre l'ouverture du fichier**
4. Envoyer le contenue au client
5. Passer à la requête suivante

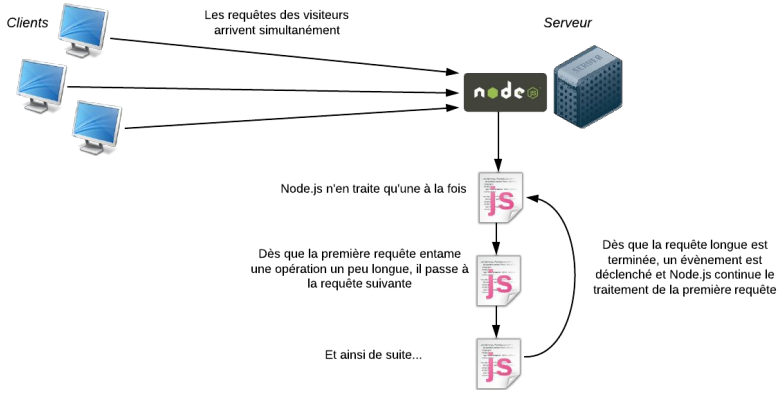
Node.js

1. Envoyer la requête au serveur
2. Demander au système de fichier d'ouvrire le fichier
3. Passer à la requête suivante
4. Quand le contenu du fichier est prêt Envoyer le contenue au client

Pourquoi Node.js



Pourquoi Node.js



Que peut faire Node.js

- Générer des page de contenue dynamique
- Créer, lire, modifier, supprimer des fichiers dans le serveur
- Créer, lire, modifier, supprimer des données dans une base de données
- Accéder au resource matériel du serveur (wifi, bluetooth, webcam ...)
- ...

Nodejs REPL (Read-Eval-Print-Loop)

- Télécharger et installer nodejs 16.18 (https://nodejs.org)
- Dans un terminal tapez ***node***

```
$ node
Welcome to Node.js v18.2.0.
Type ".help" for more information.
> console.log("Hello nodejs!!!")
Hello nodejs!!!
undefined
>
```


Premier script Node.js

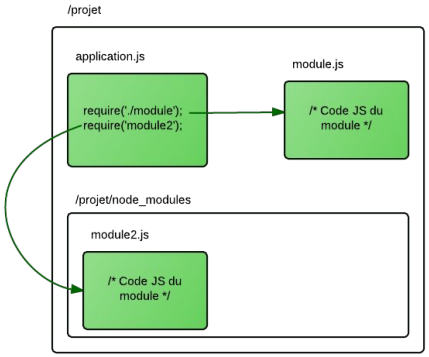
- Dans un fichier hello.js écrire
  - console.log("Hello Nodejs!!!")
- Dans un terminal tapez ***node hello.js***

```
$ node hello.js
Hello nodejs!!!
$
```

Les MODULES

Les modules

- Le noyau de Node.js est très petit.
- Ne peut pas faire grand chose
- Mais très riche on ***modules***



Les modules

```
1  var monmodule = require('./monmodule');
2
3  monmodule.direBonjour();
4  monmodule.direByeBye();
```

```
1  var direBonjour = function() {
2      console.log('Bonjour !');
3  }
4
5  var direByeBye = function() {
6      console.log('Bye bye !');
7  }
8
9  exports.direBonjour = direBonjour;
10 exports.direByeBye = direByeBye;
```

application.js

var monmodule = require('./monmodule');

monmodule.mafonction();

module.js

var mafonction = function() { ... };

exports.mafonction = mafonction;

npm

- Npm : Node Package Manager
- Avec npm on peut découvrir et installer des paquets (packages)
- Un paquet node.js est un répertoire contenant un ou plusieurs modules ou bibliothèques JavaScript.
- Pour initialiser un projet
  - ***$ npm init*** *// dans le dossier du projet*

Les modules

- Chercher un module
  - $***npm search nomModule***
- Installer un module
  - $***npm install nomModule*** //installation locale
  - $***npm install nomModule -g*** //installation globale
- Mettre à jour un module
  - $***npm update nomModule***
- Publier un module
  - $npm adduser //Creer un compte su npm
  - npm publish // publier le module

Les modules dans nodejs

Dans node.js on trouve trois types de modules:

- Module intégrés dans le corps du node.js
  - Inclusion directe
  - `const http = require('http');`
- Module tiers
  - Installation du module : $npm install express
  - Puis inclusion : `const express = require('express');`
- Module pérsonele
  - Création du module
  - Inclusion du module avec précision du chemin vers ce module
  - `const http = require('./monDossier/monModule');` //Omission de l'extension .js

Créer un module node.js

- Tout fichier node.js peut être considéré comme un module si on exporte ses fonctions et/ou ses données.
- Écrire des modules vous permettra de contribuer à la communauté Node.js.
- Tous les packages que vous utilisez sur npm ont été regroupés et partagés sous forme de modules.
- La création de modules est une compétence essentielle pour un Développeur Node.js

Créer un module node.js

- Pour créer un module, il faut:
  - $ *mkdir monModule* //Créer un dossier pour le module
  - $ *cd monModule*
  - $ *npm init -y* //initialiser package.json
  - $ *touch index.js* //créer le fichier d'entré du package
  - $ *nano index.js* //éditer le fichier et exporter des fonctions et/ou des données

package.json

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

Créer un module node.js

monModule/index.js

```
class RandomAge{
   constructor(minAge,maxAge){
       this.minAge=minAge;
       this.maxAge=maxAge
   }
   getRandomAge(){
       return Math.floor(Math.random()*(this.maxAge-this.minAge+1)+this.minAge)
   }
}
module.exports=RandomAge
//exports.RandomAge=RandomAge
```


Créer un module node.js

monModule/test.js

```
const RandomAge=require("./index")
// dans le cas de exports.RandomAge=RandomAge dans index.js
Const {RandomAge}=require("./index")
const ra=new RandomAge(20,60)
console.log(`MinAge: ${ra.minAge}`)
console.log(`MaxAge: ${ra.maxAge}`)
console.log(`Random Age: ${ra.getRandomAge()}`)
```


Installer le module

- Dans un nouveau projet monProjet:
  - $mkdir monProjet
  - $cd monProjet
  - $npm init -y
  - $npm install ../monModule
  - $touch index.js
- Importer le module ***monModule***

monProjet/index.js

```
const RandomAge=require("monModule")
const ra=new RandomAge(20,60)
console.log(`MinAge: ${ra.minAge}`)
console.log(`MaxAge: ${ra.maxAge}`)
console.log(`Random Age: ${ra.getRandomAge()}`)
```

package.json

```
{
 "name": "monprojet",
 "version": "1.0.0",
 "description": "",
 "main": "index.js",
 "scripts": {
   "test": "echo \"Error: no test
specified\" && exit 1"
 },
 "keywords": [],
 "author": "",
 "license": "ISC",
 "dependencies": {
   "monmodule": "file:../monModule"
 }
}
```


Node.js et le système de fichiers

Introduction

- Travailler avec des fichiers et des répertoires est l'un des besoins fondamentaux d'une application full-stack.
  - téléverser des images, des CV ou d'autres fichiers vers un serveur.
  - lire des fichiers de configuration,
  - Créer, lire, supprimer, déplacer des fichiers, ou même changer leurs permissions
- Le module de système de fichiers de Node.js fournit plusieurs API pour interagir avec les systèmes de fichiers de manière transparente.
- La plupart des API sont personnalisables avec des options et des drapeaux.
- On peut effectuer des opérations de fichiers synchrones et asynchrones.

Création et écriture

Pour écrire dans un fichier on peut utiliser les méthodes suivantes:

les versions ***Asynchrones***

- open("pathToFile","r/w/a",calleBack)
- append("pathToFile","encoding",data,calleBack)
- writeFile("pathToFile","encoding",data,calleBack)
- unlink("pathToFile",calleBack)
- rename("pathToFile","newName",calleBack)

Ou les versions ***synchrones***

- openSync("pathToFile","encoding","r/w/a")
- appendSync("pathToFile","encoding",data)
- writeFile("pathToFile","encoding",data)
- …

Premier script Node.js

fileReader.js

```
1  var fs = require('fs'),
2    path = './fichier.txt';
3
4  console.log('before');
5
6  fs.readFile(path, function(err, file) {
7    console.log('during');
8    console.log('' + file);
9  });
10
11 console.log('after');
```

- ***Resultat !!!***

Premier script Node.js

fileReader.js

```
1  var fs = require('fs'),
2    path = './fichier.txt';
3
4  console.log('before');
5
6  fs.readFile(path, function(err, file) {
7    console.log('during');
8    console.log('' + file);
9  });
10
11 console.log('after');
```

- ***Resultat !!!***

```
gnu@gnu-ThinkPad-T420s:~/EST/ISIL/WEB/Nodejs/01$ node fileReader.js
before
after
during
ceci est un texte de './fichier.txt'
```

Serveur web avec le module http du Node.js

C’est quoi HTTP

- **H**yper **T**ext **T**ransefer **P**rotocol
- Assure la Communication entre un client et un serveur
- Essentiellement sous forme de Requête/Réponse
- Chaque requête est indépendante des précédentes (Stateless)
- Charger des pages, envoyer des formulaires …

C’est quoi HTTPS

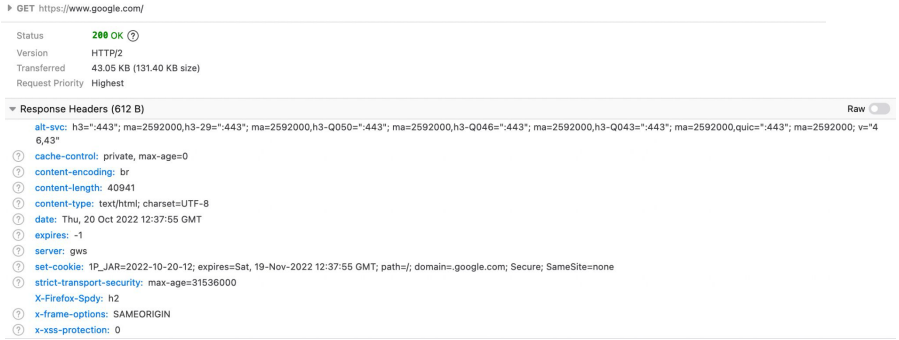
- **H**yper **T**ext **T**ransefer **P**rotocol **S**ecure
- Echange des données cryptées
- SSL (Secure Sockets Layer) /TLS (Transfer Layer Scurity)
- Il faut installer une certificat sur le serveur
- https://github.com/sgounane/upm

HTTP: Méthodes

- GET: récupérer des données du serveur
- POST: envoyer des données au serveur
- PUT: mettre à jour des données sur le serveur
- DELETE: supprimer des données du serveur

HTTP : Header

HTTP : Header



HTTP: Status Code

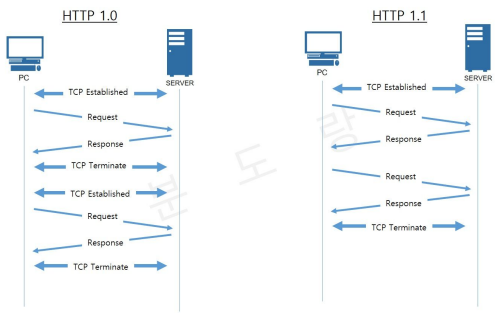
- 100 - 199 : Les réponses informatives,
- 200 - 299 : Les réponses de succès ,
- 300 - 399 : Les messages de redirection,
- 400 - 499 : Les erreurs du client,
- 500 - 599 : Les erreurs du serveur.

https://developer.mozilla.org/fr/docs/Web/HTTP/Status

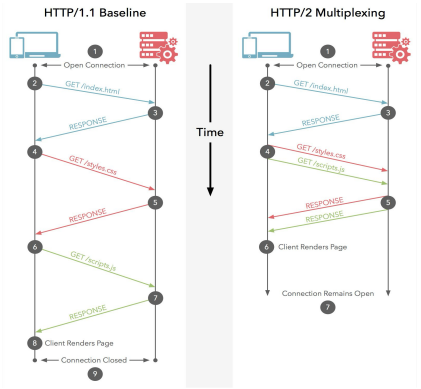
HTTP: Status Code

- 200 : OK
- 201 : OK Created
- 301 : Moved Permanently
- 304 : Not Modified
- 400 : Bad Request,
- 401 : Unauthorized
- 403 : Forbidden
- 404 : Not Found
- 405 : Method Not Allowed
- 500 : Internal Server Error
- 502 : Bad Gateway

HTTP 1.0 Vs HTTP 1.1



HTTP 1.1 Vs HTTP 2



Premier serveur web avec Node.js

app.js

```
const http=require("http");
const host="localhost";
const port=3000;
function requestHandler(req,res) {
   res.writeHead(200);
   res.write("<h1>HI!!</h1>");
   res.end();
   //res.end("<h1>Hl</h1>")
}
const server=http.createServer(requestHandler)
server.listen(port,host,()=>console.log(`Server is running on http:// ${host}:${port} `))
```

- Pour lancer l'application (le serveur HTTP)
  - $ ***node app.js***
- Pour visualiser le résultat dans le navigateur
  - ***http://localhost:3000/***

Envoyer différents types de contenu

- La réponse renvoyée d'un serveur Web peut prendre divers formats.
  - JSON; HTML ; XML ; CSV
  - PDF ; des fichiers compressés ; images ; audio ; vidéo.
- Pour envoyer un contenu dans node.js, il faut faire deux choses :
  - Définir l'en-tête ***Content-type*** dans les réponses HTTP avec le valeur appropriée.
  - S'assurer que ***res.send()*** obtient les données dans le bon format.

Envoyer différents types de contenu

- ***Content-type*** peut être:
  - Du texte brut : **text/plain**
  - Du HTML : **text/html**
  - Du CSS : **text/css**
  - Une image JPEG : **image/jpeg**
  - Une vidéo MPEG4 : **video/mp4**
  - Un fichier ZIP : **application/zip**
  - etc.

Envoyer du JSON

```
const http=require("http");
const host="localhost";
const port=3000;
function requestHandler(req,res) {
   res.setHeader("Content-type", "application/json")
   res.writeHead(200);
   res.end(JSON.stringify({nom:"gounane",age:22}))
}
const server=http.createServer(requestHandler)
server.listen(port,host,()=>console.log(`Server is running on http://${host}:${port} `))
```


Envoyer du HTML

```
const http=require("http");
const host="localhost";
const port=3000;
function requestHandler(req,res) {
   res.setHeader("Content-type", "text/html")
   res.writeHead(200);
   res.end("<h1> Bonjour Tous!!<h1>")
}
const server=http.createServer(requestHandler)
server.listen(port,host,()=>console.log(`Server is running on http://${host}:${port} `))
```

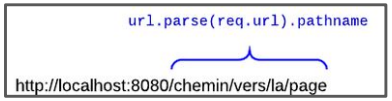

Envoyer une page HTML depuis un fichier

```
const http=require("http");
const fs=require("fs")
const host="localhost";
const port=3000;
function requestHandler(req,res) {
   fs.readFile(__dirname + "/index.html",(err,data)=>{
       res.setHeader("Content-type", "text/html")
       res.writeHead(200);
       console.log(data)
       res.end(data)
   })
}
const server=http.createServer(requestHandler)
server.listen(port,host,()=>console.log(`Server is running on http:// ${host}:${port} `)
```

Node.js: Gestion des requêtes

Pour récupérer la page demandée par le visiteur:

1. on utilise le module "***url***" pour décomposer le url de la requête
2. Extraire Le chemin de la ressource en question (***pathname***)
3. Envoyer la réponse adequate



Node.js: Gestion des requêtes

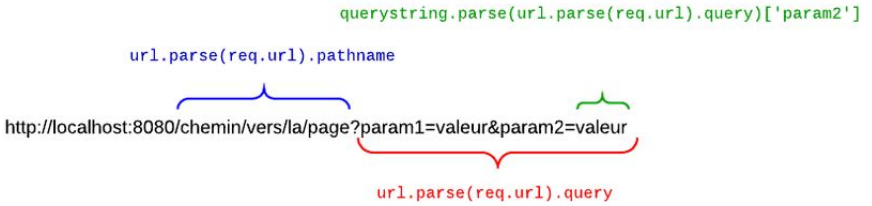
```
var http = require('http');
var url = require('url');

var server = http.createServer(function(req, res) {
    var page = url.parse(req.url).pathname;
    console.log(page);
    res.writeHead(200, {"Content-Type": "text/plain"});
    if (page == '/') {
        res.write('Accueil');
    }
    else if (page == '/contact') {
        res.write('Nous contscter');
    }
    else if (page == '/cours/web/nodejs') {
        res.write('Node.js is greate');
    }
    res.end();
});

server.listen(3000);
```


Node.js: Gestion des requête

Pour récupérer les paramètres de la requête:

1. Utiliser le module "***url***" pour décomposer le url de la requête
2. Extraire Les paramètres en question (***query***) => chaîne de caractères
3. Utiliser le module “***querystring***” pour décomposer cette chaîne en {key: value}
4. Envoyer la réponse adequate

Node.js: Gestion des requête



Node.js: Gestion des requête

```
var http = require('http');
var url = require('url');

var querystring = require('querystring');

var server = http.createServer(function(req, res) {
    var params = querystring.parse(url.parse(req.url).query);
    res.writeHead(200, {"Content-Type": "text/plain"});
    if ('prenom' in params && 'nom' in params) {
        res.write('Vous vous appelez ' + params['prenom'] + ' ' + params['nom']);
    }
    else {
        res.write('Il manque des parametres!!');
    }
    res.end();

});

server.listen(3000);
```

1. Lancer le serveur: ***$ node params.js***
2. Envoyer la requête : ***http://localhost:3000/?nom=yahyaoui&prenom=anas***
3. Puis la requête: ***http://localhost:3000***

Node.js: les evenements

- On Node.js, un grand nombre d'objets émettent des évènements.
- Ces objets héritent tous d'un objet EventEmitter fourni par Node.
- Par exemple L'objet **server** de **HTTP** possède les événements:
  - ***Request***
  - ***Connection***
  - ***Close***
  - ***Connect***
  - ***Upgrade***
  - ***clienError***

Node.js: les evenements

- Pour écouter à un événement sur un objet on appel la méthode on() de cet objet:

*obj.**on**('**evenement**', **callbackFunction**)*

Node.js: les evenements

- Pour écouter à un événement sur un objet on appel la méthode on() de cet objet:

*obj.**on**('**evenement**', **callbackFunction**)*

```
var server = http.createServer();
server.on('request', function(req, res) { });
```

```
var server = http.createServer(function(req, res) { });
```

Node.js: les evenements

```
1  var http = require('http');
2
3  var server = http.createServer(function(req, res) {
4    res.writeHead(200);
5    res.end('Salut ISIL !');
6  });
7
8  server.on('close', function() { // On écoute l'évènement close
9      console.log('Au revoir!!');
10 });
11
12 server.listen(3000); // Démarre le serveur
13 server.close(); // Arrête le serveur. Déclenche l'évènement close
14
```


Node.js: les evenements

- Pour créer un objet susceptible d'émettre un événement:
  - Inclure le module '***events***'
  - Créez un objet basé sur EventEmitter
  - Quelque part dans l'app, Pour écouter l'événement sur cet objet
    - *obj.**on**('**nomEvent**', **function(param1,param2, ...){ … }**)*
  - Pour déclencher l'événement on appele la méthode ***emit('nomEvent', param1, param2, ...)***

Node.js: les evenements

```
1  var EventEmitter = require('events').EventEmitter;
2
3  var monObjet = new EventEmitter();
4
5  monObj.on('monEvent', function(param){// on ecoute monEvent
6      console.log(param);
7
8  });
9
10 monObje.emit('monEvent', 'Un texte de monEvent');// Declancher monEvent
```


Express.js

Le framework Express.js

- Express.js est un micro-framework pour Node.js.
- Il met à disposition des outils de base pour aller plus vite dans la création d'applications Node.js.
- Pas assez complet comme les framework php (laravel, symphony ..) python(django).
- Pour installer express.js
  - ***npm install express***

Express.js: firstApp

```
var http = require('http');
var url = require('url');

var server = http.createServer(function(req, res) {
    var page = url.parse(req.url).pathname;
    console.log(page);
    res.writeHead(200, {"Content-Type": "text/plain"});
    if (page == '/') {
        res.write('Accueil');
    }
    res.end();
});

server.listen(3000);
```

```
var express = require('express');

var app = express();

app.get('/', function(req, res) {
    res.setHeader('Content-Type', 'text/plain');
    res.send('Accueil');
});

app.listen(3000);
```


Les routes

```
var express = require('express');
var app = express();

app.get('/', function(req, res) {
    res.setHeader('Content-Type', 'text/plain');
    res.write('Accueil');
});

app.get('/contact', function(req, res) {
    res.setHeader('Content-Type', 'text/plain');
    res.write('Nous contscter');
});

app.get('/cours/web/nodejs', function(req, res) {
    res.setHeader('Content-Type', 'text/plain');
    res.write('Node.js is greate');
});

app.use(function(req, res, next){
    res.setHeader('Content-Type', 'text/plain');
    res.status(404).send('Page introuvable !');
});

app.listen(3000);
```


Les routes dynamiques

Express permet de gérer des routes dont certaines portions peuvent varier en insérant une variable ***:mavariable*** dans l'URL de la route

Ce qui crée un paramètre accessible avec ***req.params.mavariable***.

```
app.get('/capteurs/:id/:temp', function(req, res) {
    res.setHeader('Content-Type', 'text/plain');
    res.end('La temperature mesurerpar le capteur n°' +
            req.params.id + ' est: ' + req.params.);
});
```

Routes Modulaires avec express.Router()

- Lorsque les applications Express.js deviennent complexes, organiser les routes de manière modulaire devient essentiel pour maintenir la clarté et la facilité de maintenance du code.
- ***express.Router()*** est une fonctionnalité puissante qui permet de créer des routes modulaires dans Express.js.
- Il permet de séparer les routes en fichiers distincts, offrant ainsi une meilleure organisation du code.

Routes Modulaires avec express.Router()

- Création d'un Module de Route

userRouter.js

```
const express = require('express');
const router = express.Router();
router.get('/', (req, res) => {
    res.send('Liste des utilisateurs');
});
router.get('/:id', (req, res) => {
    res.send(`Info de l'utilisateur ${req.params.id}`);
});
module.exports = router;
```


Routes Modulaires avec express.Router()

- Utilisation du Module dans l'Application Principale
  - Dans cet exemple, toutes les routes définies dans userRoutes.js seront préfixées par /users.
  - Par exemple, l'URL complète pour la liste des utilisateurs serait /users/

app.js

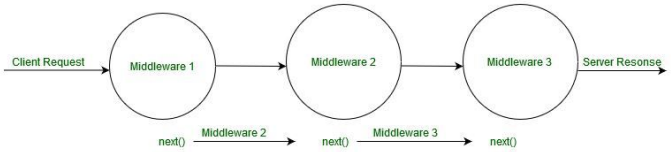
```
const express = require('express');
const userRouter = require('./userRouter'); // Chemin du module de route
const app = express();
app.get('/', (req, res) => {
   res.send('Home');
});
app.use('/users', userRouter);
app.listen(300,()=>console.log("listening on port 3000"))
```


Avantages des Routes Modulaires

- **Meilleure Organisation** : Les routes modulaires favorisent une meilleure organisation du code en regroupant des fonctionnalités liées dans des fichiers séparés.
- **Facilité de Maintenance** : Les modifications et les ajouts de fonctionnalités peuvent être effectués plus facilement car chaque module de route est autonome.
- **Réutilisabilité** : Vous pouvez réutiliser des modules de route dans différentes parties de votre application ou même dans d'autres projets.
- **Lisibilité** : La lisibilité du code est améliorée car chaque fichier se concentre sur une tâche ou une fonctionnalité spécifique.
- **Évolutivité** : Les applications complexes peuvent évoluer de manière plus saine en utilisant des modules de route, car ils facilitent l'ajout de nouvelles fonctionnalités sans affecter le reste de l'application.

les middlewares

- Ce sont des petits morceaux d'application qui rendent chacun un service spécifique.



les middlewares

- Les middlewares de base dans Express ne sont pas nombreux:
  - ***compression*** : permet la compression gzip de la page
  - ***cookie-parser*** : permet de manipuler les cookies
  - ***cookie-session*** : permet de gérer des informations de session (durant la visite d'un visiteur)
  - ***serve-static*** : permet de renvoyer des fichiers statiques contenus dans un dossier (images, fichiers à télécharger...)
  - ***serve-favicon*** : permet de renvoyer la favicon du site
  - etc.

les middlewares

- Ces middlewares sont interconnectés et peuvent communiquer entre eux en se renvoyant jusqu'à 4 paramètres
  - ***err***: les erreurs
  - ***req***: la requête du visiteur
  - ***res***: la réponse à renvoyer (la page HTML et les informations d'en-tête)
  - ***next***: un callback vers la prochaine fonction à appeler

les middlewares

```
1  var express = require('express');
2  var favicon = require('serve-favicon'); // Charge le middleware de favicon
3
4
5  var app = express();
6  // Indiquer que le dossier /public contient des fichiers
7  //statiques (middleware chargé de base)
8  app.use(express.static(__dirname + '/public'));
9
10 // Activer la favicon indiquée
11 app.use(favicon(__dirname + '/public/favicon.ico'));
12
13 // Répondre a la requete
14 app.use(function(req, res){
15     res.send('Hello');
16 });
17
18 app.listen(3000);
```

Créer un Middleware

- Dans Express.js, un middleware est simplement une fonction avec trois arguments : ***req*** (la requête), ***res*** (la réponse), et ***next*** (la fonction à appeler pour passer à la fonction middleware suivante).
- Un middleware peut effectuer des opérations sur la requête et la réponse, et peut appeler ***next()*** pour passer la main au middleware suivant.

```
const monMiddleware = (req, res, next) => {
   // Effectuez des opérations sur la requête ici
   console.log('Middleware fonctionne !');
   next(); // Appel à next pour passer au middleware suivant
};
```


Utiliser un Middleware

- **Application-Level Middleware** : Ces middlewares s'appliquent à chaque requête dans l'application

```
app.use(monMiddleware);
```


Utiliser un Middleware

- **Route-Level Middleware** : Ces middlewares s'appliquent uniquement aux routes spécifiques où ils sont définis.

```
app.get('/route', monMiddleware, (req, res) => {
   // Gestionnaire de route
});
```


Utiliser un Middleware

- **Error-Handling Middleware** : Ces middlewares sont utilisés pour gérer les erreurs.

```
app.use((err, req, res, next) => {
   // Gestion des erreurs
});
```

Utilisation Pratique des Middlewares

Authentification : Vérifiez l'authenticité des utilisateurs avant de permettre l'accès aux ressources protégées.

```
const vérifierAuthentification = (req, res, next) => {
   // Vérification de l'authentification
   if (utilisateurAuthentifié) {
       next();
   } else {
       res.status(401).send('Non autorisé');
   }
}
```


Utilisation Pratique des Middlewares

Journalisation : Enregistrez les détails des requêtes pour des raisons de suivi et de débogage.

```
const journalisation = (req, res, next) => {
   console.log(`Requête ${req.method} sur ${req.url}`);
   next();
};
```


Utilisation Pratique des Middlewares

- Body-Parser est un middleware pour Express.js qui:
  - analyse le corps des requêtes entrantes dans différents formats
  - expose ces données à travers l'objet req.body de l'application Express.
- Il peut gérer divers types de données, y compris:
  - les données JSON,
  - les données de formulaire,
  - les données de requête de l'URL (query parameters)
  - d'autres types de données.

Le middleware Body-parser

- Installer body-parser
  - $ npm i body-parser
- Importer et utiliser le module en tant que middleware.

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

// Utilisation de Body-Parser Middleware
app.use(bodyParser.json()); // Pour les données JSON
app.use(bodyParser.urlencoded({ extended: true })); // Pour les données de formulaire
```

Le middleware Body-parser

- Accéder aux données de la requête dans vos routes Express. Les données analysées sont disponibles dans req.body

```
app.post('/exemple', (req, res) => {
   const jsonData = req.body;
   // Faites quelque chose avec jsonData
   res.send('Données reçues : ' + JSON.stringify(jsonData));
 });
```


Avantages des Middlewares

- **Modularité** : Les middlewares favorisent la modularité en permettant de découper l'application en petites fonctions réutilisables.
- **Gestion des Requêtes** : Ils offrent un contrôle total sur les requêtes entrantes, permettant des validations, des transformations et des filtrages.
- **Réutilisabilité** : Les middlewares peuvent être réutilisés dans différentes parties de l'application, garantissant la cohérence dans le traitement des requêtes.

Les Sessions

- Les sessions sont un mécanisme essentiel dans le développement web permettant de **maintenir l'état de l'utilisateur entre les requêtes**.
- Express.js offre une gestion de session simple et efficace grâce à différents middlewares et à au module **express-session**.
- Pour installer ce module :
  - $ npm i express-session

Les Sessions: Configuration

- Dans l'application (par exemple, app.js), il faut configurer express-session en tant que middleware.
- Il faut également spécifier un secret pour signer les cookies de session, ce qui est essentiel pour la sécurité.

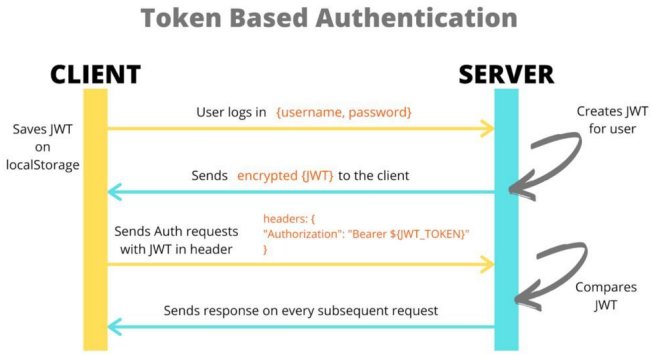
```
const express = require('express');
const session = require('express-session');
const app = express();
app.use(session({
   secret: 'votre_secret',
   resave: false,
   saveUninitialized: true,
   cookie: { maxAge: 60*60000 } // Durée de vie de la session en millisecondes (ici, 1
huere)
}));
```

Les Sessions : Utilisation

- Une fois configurées, les sessions sont accessibles via l'objet req.session. Vous pouvez y stocker des données liées à l'utilisateur.

```
app.get('/login', (req, res) => {
   // Vérifiez les informations d'authentification de l'utilisateur
   // Si les informations sont correctes, enregistrez l'utilisateur dans la session
   req.session.utilisateur = utilisateur; // utilisateur est l'objet de l'utilisateur
   res.send('Connecté avec succès');
});
app.get('/profil', (req, res) => {
   // Accédez aux données de l'utilisateur à partir de la session
   const utilisateur = req.session.utilisateur;
   if (utilisateur) {
       res.send('Bienvenue sur votre profil, ' + utilisateur.nom);
   } else {
       res.send('Non connecté');
   }
});
```


Json Web Token



https://www.freecodecamp.org/news/how-to-sign-and-validate-json-web-tokens/

JWT: signature

```
…..
const jwt=require("jsonwebtoken")
//const TOKEN_SECRET= require('crypto').randomBytes(64).toString('hex')
// console.log(TOKEN_SECRET)
const TOKEN_SECRET="un secret"
app.post('/api/login', (req, res) => {
  Const token=jwt.sign(
                              { username: req.body.username },
                              TOKEN_SECRET,
                              { expiresIn: '24h' }
                          );
  res.json(token);
}
```


JWT: verification

```
const jwt = require('jsonwebtoken');
function verifyToken(req, res, next) {
 const authHeader = req.headers['authorization'] //req.header("Authorization")
 const token = authHeader && authHeader.split(' ')[1]
 if (token == null) return res.sendStatus(401)
 jwt.verify(token,TOKEN_SECRET, (err, user) => {
   if (err) return res.sendStatus(403)
   req.user = user
   next()
 })
}
```

```
app.get("/api/products", verifyToken,(req,res)=>{
   res.status.send(req.use)
})
```

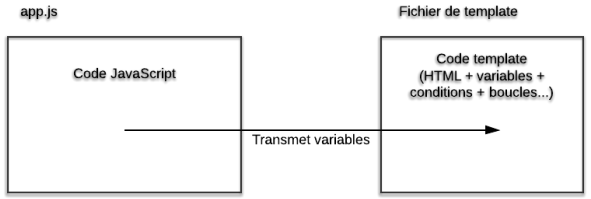
Les templates

- Envoyer un contenu Html en utilisant cette méthode n'est pas une tâche agréable
- Les templates permettent de produire du HTML et d'insérer au milieu du contenu, des variables (javascript).
- Il existe beaucoup de langages de templates: Mustache, Underscore, EJS, JSP, HandlebarsJS, pug (Jade)...

```
res.write('<!DOCTYPE html>'+
'<html>'+
'    <head>'+
'        <meta charset="utf-8" />'+
'        <title>Ma page Node.js !</title>'+
'    </head>'+
'    <body>'+
'       <p>Voici un paragraphe <strong>HTML</strong> !</p>'+
'    </body>'+
'</html>');
```


Les templates

- Principe:
  - depuis un fichier JavaScript, on appelle une template en lui transmettant les variables dont il a besoin pour construire la page.



PUG

1. Pug est un moteur de templates implémenté en JavaScript
2. Il permet de générer dynamiquement du HTML
3. Il est minimaliste et basé sur les indentations.
4. Pug était auparavant connu sous le nom de Jade

Pug: Exemple

- Installer ***pug*** dans le dossier de votre projet
  - $npm install pug
  - $npm install pug-cli -g
- Premier execution de pug
  - $pug index.pug -w -P
- Dans l'application express il faut définir le moteur de templates utilisé et le chemin d'accès aux templates
  - `app.set("view engine", "pug");`
  - `app.set("views", "path/to/views");`
- Finalement on appel la méthode *render(template,params)* de l'objet *response*

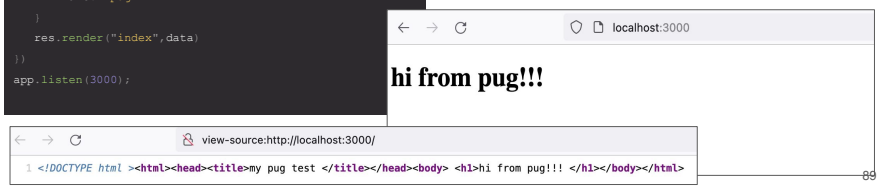
Pug: Exemple

app.js

```
const { application } = require("express")
const express=require("express")

const app=express()
app.set("view engine","pug")
app.set("views","./templates")
app.use((req,res)=>{
   let data={
       name: "pug"
   }
   res.render("index",data)
})
app.listen(3000);
```

templates/index.pug

```
doctype html
html
   head
       title my #{name} test
   body
       h1 hi from #{name}!!!
```



Classes et ID

- Pour attribuer une classe à un élément html, on utilise le “.”
  - *h1.titre … => <h1 class=”titre”> … </h1>*
- Pour attribuer un ID à un élément html, on utilise le “#”
  - *h1#titre … => <h1 id=”titre”> … </h1>*
- *Si on ne précise pas l’élément Html pug utilise la balise *
  - *.titre … =>   … *
  - *#titre … =>   … *
- *Pour ajouter un attribut à un élément on utilise les parenthèses “(attribut=valeur)”*
  - *img.garde(src=”logo.png”, alt=”mon logo”) => *

Texte

- On peut ajouter un texte à une template par plusieurs façons:

```
p Pug rocks!
P
     | You are logged in as
     | user@example.com
P.
     Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
     incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
     exercitation ullamco laboris nisi ut aliquip ex ea commodo consequa
```


Commentaires

- On peut ajouter des commentaires à une template par plusieurs façons:

```
// My wonderful navbar
nav#navbar-default
//- My wonderful navbar (silent comment)
nav#navbar-default
//
    My wonderful navbar
    It is just so, awesome!
nav#navbar-default
nav#navbar-default // My wonderful navbar
```

Javascript dans pug

- Une caractéristique importante de pug est la possibilité d'exécuter du javascript dans ses templates.
- On peut insérer des variables,des objets, des boucles, des conditions …ect
- Pour utiliser js dans pug il faut distinguer entre
  - **Buffered code** (le code tamponné) :
  - **Unbuffered code** (le code non tamponné)

unbuffered et buffered code

- unbuffered code (le code tamponné) :
  - commence par (-)
  - N'ajoute rien à la sortie, mais ses valeurs peuvent être utilisé dans le reste de la template
- buffered code (le code non tamponné)
  - Commence par (=)
  - Le javascript est évalué et rendu en sortie

```
//-unbuffered code
- const name = "said"
//- On peut maintenant utiliser name dans le reste de la template pug
//-buffered code
p= 'Hi my name is: ' + name
p= 'my age is: ' + 2*10 +'years old'
//-Pour des raison de securité le buffered code est échappé (escaped)
p= '<script>alert("Hi")</script>'
//- Donne en sortie:  <p>&lt;script&gt;alert(&quot;Hi&quot;)&lt;/script&gt;</p>p>
```


Interpolation

- L'interpolation est le processus de remplacer des gabarits (placeholders) par des valeurs des expressions js.
- L'interpolation se fait par plusieurs méthodes:
  - Le buffered code (déjà vu)
  - L'utilisation de #{expression js}

```
- const name = "said"
p Hi #{name}
p Hi #{name.toUpperCase()}
p Hi #{name.charAt(0).toUpperCase() + name.slice(1)}
//on peut omettre #{} pour affecter la valeur d'une var à un attribut d'un élément
img(src="portrait.png", alt=name)
```


Les boucles

- Pour parcourir les élément d'un tableau dans pug on utilise le mot clé ***each***

```
-const days=["lundi", "mardi", "mercredi", "jeudi", "vendredi",
"vendredi", "samedi", "dimanche"]
ol
    each day in days
        li= day
    else
        li le tableau est vide
```

Les boucles

- On peut aussi parcourir les éléments d’un objet

```
-
   const employee = {
       'name': 'said',
       'email': 'said@upm.ma',
       'age':21
   }
ul
each value, key in employee
   li= `${key}: ${value}`
each l’objet est {}
```


Conditionnels

- Les conditionnels offrent un moyen très pratique de rendre différents HTML en fonction du résultat d'une expression JavaScript :

```
-
   const student = {
       'name': 'said',
       'email': 'said@upm.ma',
       'age':12
   }
if student.age<11
   p kid
else if student.age<15
   p junior
else
   p senior
```

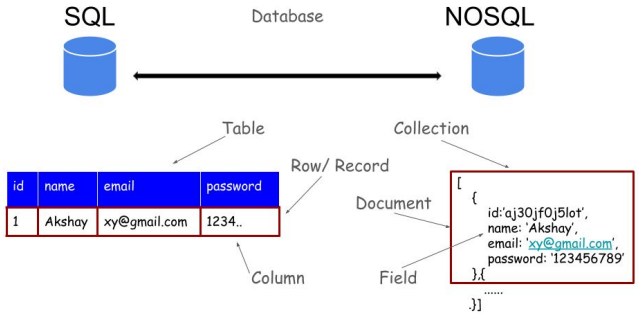

Nodejs

Databases: mongodb

Introduction

- MongoDB est une base de données de documents NoSQL (sans schéma).
- Cela signifie que
  - vous pouvez y stocker des documents JSON,
  - La **structure** de ces documents peut **varier** Ce qui rend les bases de données MongoDB très faciles à modifier et à mettre à jour à l'avenir
- C'est l'un des avantages de l'utilisation de NoSQL car il
  - accélère le développement d'applications
  - réduit la complexité des déploiements.

SQL vs NOSQL DB



Introduction

- Pour pouvoir expérimenter les exemples de code il faut:
  - Télécharger et installer MongoDB sur votre machine locale
  - Ou, utiliser un service cloud MongoDB à l' adresse https://www.mongodb.com/cloud/atlas .
  - Mongodb compass
- Pour utiliser mongodb avec nodejs il faut installer le pilote MongoDB
  - ***npm install mongodb***

Créer un base de donnée

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/upmdb";

MongoClient.connect(url, function(err, db) {
//si la base de donnée upmdb est introuvable elle sera
automatiquement créer
 if (err) throw err;
 console.log("Database created!");
 db.close();
});
console.log("End !!!")
```


Créer une collection

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
 if (err) throw err;
 var dbo = db.db("upmdb");
 dbo.createCollection("users", function(err, res) {
   if (err) throw err;
   console.log("Collection created!");
   db.close();
 });
});
```

Inserer un document

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
 if (err) throw err;
 let dbo = db.db("upmdb");
 let user = { name: "reda", password: "1234" };
 dbo.collection("users").insertOne(user, function(err, res) {
   if (err) throw err;
   console.log("1 document inserted");
   db.close();
 });
});
```


Insérer plusieurs documents

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
 if (err) throw err;
 let dbo = db.db("upmdb");
 let users=[{ name: "nadia", password: "1234" },{ name: "Khadija", password:
"1234" }];
 dbo.collection("users").insertMany(users, function(err, res) {
   if (err) throw err;
   console.log("2 documents inserted");
   db.close();
 });
});
```


Find

- ***db.collection(collection).findOne({}, (err,res)=>{})***: renvoi le premier documents dans la collection
- ***db.collection(collection).find({}).toArray((err,res)=>{})***: renvoi tous les documents dans la collection
- ***db.collection(collection).find({query}).toArray((err,res)=>{})***: renvoi tous les documents correspondants à “query” dans la collection
- ***db.collection(collection).find({query}, { projection: { _id: 0, name: 1, password: 1 } }).toArray((err,res)=>{})***: renvoi tous les documents correspondants à “query” dans la collection sans le champ ***id***

findOne

```
const MongoClient = require('mongodb').MongoClient;
const url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
 if (err) throw err;
 let dbo = db.db("upmdb");
 dbo.collection("users").findOne({}, function(err, result) {
   if (err) throw err;
   console.log(result);
   db.close();
 });
});
```

findAll

```
const MongoClient = require('mongodb').MongoClient;
const url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
 if (err) throw err;
 let dbo = db.db("upmdb");
 dbo.collection("users").find().toArray(function(err, result) {
   if (err) throw err;
   console.log(result);
   db.close();
 });
});
```


Find query

```
const MongoClient = require('mongodb').MongoClient;
const url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
 if (err) throw err;
 let dbo = db.db("upmdb");
 //let query={name:"said"};
 let query={$and:[{name:"said"},{password:"1234"}]};
 dbo.collection("users").find(query).toArray(function(err, result) {
   if (err) throw err;
   console.log(result);
   db.close();
 });
});
```


Update one/many

```
const MongoClient = require('mongodb').MongoClient;
const url = "mongodb://127.0.0.1:27017/";

MongoClient.connect(url, function(err, db) {
 if (err) throw err;
 const dbo = db.db("upmdb");
 let query = { name: "nadia" };
 let values = { $set: {password: "2345" } };
 //dbo.collection("users").updateMany(query, values, function(err, res) {
 dbo.collection("users").updateOne(query, values, function(err, res) {
   if (err) throw err;
   console.log(res);
   db.close();
 });
});
```

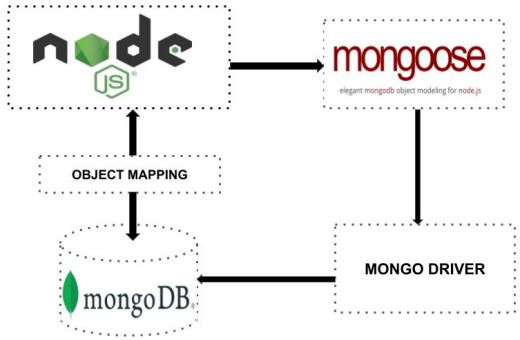

Delete one/many

```
let MongoClient = require('mongodb').MongoClient;
let url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db) {
 if (err) throw err;
 let dbo = db.db("upmdb");
 //let query = { name: 'said' };
 let query = { name: /^s/ }; //starts with "s"
 //dbo.collection("users").deleteMany(query, function(err, obj) {
 dbo.collection("users").deleteOne(query, function(err, obj) {
   if (err) throw err;
   console.log("1 document deleted");
   db.close();
 });
});
```

Mongodb async

```
const MongoClient = require('mongodb').MongoClient;
const url = 'mongodb://localhost:27017';
(async function() {
 let client;
 try {
   client = await MongoClient.connect(url);
   console.log("Connected correctly to server");
   const db = client.db("upmdb");
   let r = await db.collection('users').insertOne({name:"said"});
   console.log(r)
   r = await db.collection('users').insertMany([{name:"said"}, {name:"amale"}]);
   console.log(r)
   r = await db.collection('users').find({name:"said"}).limit(2).sort({password:1}).toArray();
   console.log(r)
 } catch (err) {
   console.log(err.stack);
 }
 client.close();
})();
```


Object Data Modeling : Mongoose



C’est quoi mongoose

- Mongoose est une bibliothèque ODM (Object Data Modeling) pour MongoDB et node.js.
- gère les relations entre les données,
- fournit une validation de schéma,
- utilisé pour traduire entre les objets dans le code et la représentation de ces objets dans MongoDB.

Pourquoi Mongoose

- Par défaut, MongoDB a un modèle de données flexible.
- Cela rend les bases de données MongoDB très faciles à modifier et à mettre à jour à l'avenir.
- Mais beaucoup de développeurs sont habitués à avoir des schémas rigides.
- Mongoose force un schéma semi-rigide dès le départ.
- Avec Mongoose, les développeurs ***doivent*** définir un ***schéma*** et un ***modèle***.

Terminologies

1. **Schéma**: Un « schéma » Mongoose est une structure de données de document (ou la forme du document) qui est appliquée via la couche d'application.
- **Modèles**: des constructeurs qui prennent un schéma et créent une instance d'un document

Schéma

- Un schéma définit la structure de vos documents de collection.
- Un schéma Mongoose correspond directement à une collection MongoDB.

```
const { Schema, Model } =
require("mongoose");
let user=new Schema({
   name: String,
   age:Number,
   email: {
       type:String,
       unique:true
   }
})
```


Modèle

- Les modèles prennent votre schéma et l'appliquent à chaque document de sa collection.
- Les modèles sont responsables de toutes les interactions de document telles que la création, la lecture, la mise à jour et la suppression (CRUD).

```
const { Schema, Model } =
require("mongoose");
let user=new Schema({
   name: String,
   age:Number,
   email: {
       type:String,
       unique:true
   }
})
let User=new Model("User",user)
```


Modèle

- Le premier argument (“***User”***) passé au modèle doit être la **forme singulière** du nom de votre collection.
- Mongoose le change automatiquement au pluriel , le transforme en minuscules (***users***) et l'utilise pour le nom de la collection de base de données.

```
const { Schema, Model } =
require("mongoose");
let user=new Schema({
   name: String,
   age:Number,
   email: {
       type:String,
       unique:true
   }
})
let User=new Model("User",user)
```

Connexion à la BD

```
const mongoose =require('mongoose')
const uri="mongodb+srv://<user>:<password>@<dbhost>/upmdb"
const dbConnect= async ()=>{
   try{
       const conn=await mongoose.connect(uri)
       console.log("connected to db ....")
   }catch(err){
       console.log(err)
   }
}
dbConnect()
```


Insérer un document

```
const User = require('./models/user')
const user=new User({
   name:"said",
   age:20,
   email:"said@upm.ma"
})
const addone=async ()=> {
   await user.save();
   console.log("user added to users collection..");
}
addone()
```


Insérer un document

```
const User = require('./models/user')
const addone=async ()=> {
    const user = await User.create({
       name:"said",
       age:20,
       email:"said@upm.ma"
    });
    console.log(user);
}
addone()
```


CRUD

```
const user = await User.findOne({});
//update user
user.name="Ahmed"
await user.save()
//find by id
const user = await User.findById("62472b6ce09e8b77266d6b1b").exec();
//projection
const user = await User.findById("62472b6ce09e8b77266d6b1b","name age").exec();
console.log(user);
//suppression
const user = await User.deleteOne({name:"said"}).exec(); //deleteMany()
console.log(user);
```

findOne

```
const User = require('./models/user')
const getOneUser=async ()=> {
   const user = await User.findOne({});
   console.log(user);
}
getOneUser()
```