**Node.js, Mangodb, Express.js**

1. Installation de Express.js

<https://expressjs.com/en/starter/installing.html>

Dans le terminal de vs code : npm install express

Puis dans le fichier mère.js appeler l'api :

const express = require("express");

const app = express();

Entrée du port d'écoute.

const PORT = 4000;

Appel de la fonction d'écoute

app.listen(PORT, () => console.log(`Example app(express) d'écoute sur l'adresse http://localhost:${PORT}`));

Installer nodemon pour éviter de relancer le serveur :

Faire dans le terminal de vs code 🡺 npm install nodemon

Dans le fichier package.json 🡺 "start": "nodemon serveur.js"

"serveur.js est le point d'entrée du programme."

1. Installation de Mangodb

<https://docs.mongodb.com/manual/>

Installer l'exécutable et compasse.

A la racine, de l'ordinateur, ( c:\) , il faut créé un dossier "data" puis a l'intérieur un autre dossier "db".

Copier l'adresse du dossier "bin " de Mongodb, puis aller dans variable d'environnement, cliquer deux fois sur "path", puis sur nouveau et coller l'adresse du dossier "bin", puis ok.

Ouvrer le terminal :

mongod et entrer. => fait démarrer le serveur, (ne pas fermer).

Dans une nouvelle fenêtre de terminal :

mongo et entrer. => accès à la base de données.

Utilisation de mongodb :

* use nomDeDossier 🡺 Création du dossier, et entrée dans le dossier.
* db.nomDeDossier.insertOne({objet}) 🡺 Insertions de valeur en forme d'objet dans la base de donnée.
* Db.nomDeDossier.find() 🡺 ouvre les objets en cours dans le dossier
* db.nomDeDossier.insertMany({plusieurs objets}) 🡺 insertion de plusieurs objets dans la dataBase.
* OPENCLASSROOMS :

# [Passez au Full stack avec Node.js, Express et MongoDB](https://openclassrooms.com/fr/courses/6390246-passez-au-full-stack-avec-node-js-express-et-mongodb)

ode.js, Express et MongoDB

Fichier app.js contient le rooter (API) :

// middleware => serie de fonction qui envoi et reçoi des requetes

// Chaque réponses ne peuvent être envoyées qu’une fois.

const express = require("express");

const app = express();

//middlewre simple avec fonction next() pour quelles s'enchaines.

app.use((req, res, next)=>{

    console.log("Requête bien reçu !");

    next(); // fonction qui permet de passé au middleware suivant.

});

app.use((req, res, next)=>{

    res.status(201); // modifie le code de la réponse

    next();

});

app.use((req, res, next)=>{

    res.json({message: "Votre requete à bien été reçu !"});// réponse envoyée

    next();

});

app.use((req, res, next)=>{

    console.log("La réponse à été envoyée avec succès")

});

module.exports = app;

MiddleWare =>

* Le premier enregistre « Requête reçue ! » dans la console et passe l'exécution ;
* Le deuxième ajoute un code d'état 201 à la réponse et passe l'exécution ;
* Le troisième envoie la réponse JSON et passe l'exécution ;
* Le dernier élément de middleware enregistre « Réponse envoyée avec succès ! » dans la console.

Fichier server.js contient le serveur.

const http = require("http");

const app = require("./app");

/\*

\*Reponse de base

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

\*    res.end("Voilà la réponse du super server !");

\*});

\*/

app.set("port", process.env.PORT || 3000)

(port défini, la variable d’environnement ou le port 3000)

const server = http.createServer(app);

server.listen(process.env.PORT || 3000);

Après quelques améliorations pour la stabilité sur fichier server.js :

const http = require("http");

const app = require("./app");

const normalisePort = (val) => {

  const port = parseInt(val, 10);

  if (isNaN(port)) {

    return val;

  }

  if (port >= 0) {

    return port;

  }

  return false;

};

const port = normalisePort(process.env.PORT || 3000);

app.set("port", port);

const errorHandler = error =>{

    if(error.syscall !== 'listen'){

        throw error;

    }

    const adress = server.address();

    const bind = typeof adress === "string" ? "pipe" + adress : "port" + port;

    switch (error.code){

        case "EACCES":

            console.error(bind + "Besoin de privileges élever.");

            process.exit(1);

            break;

        case "EADDRINUSE":

            console.error(bind + "est déjà utilisé");

            process.exit(1);

            break;

        default:

            throw error;

    }

};

const server = http.createServer(app);

server.on("error", errorHandler);

server.on("listening", ()=>{

    const adress = server.address();

    const bind = typeof adress === "string" ? "pipe" + adress : "port" + port;

    console.log("En écoute sur : "+bind);

})

server.listen(port);

* La fonction normalizePort renvoie un port valide, qu'il soit fourni sous la forme d'un numéro ou d'une chaîne ;
* La fonction errorHandler recherche les différentes erreurs et les gère de manière appropriée. Elle est ensuite enregistrée dans le serveur ;
* Un écouteur d'évènements est également enregistré, consignant le port ou le canal nommé sur lequel le serveur s'exécute dans la console.