TP3 : MongoDB et Node.js avancé

Objectifs
Accès à une base de données mongodb avec node.js
Connexion à un serveur HTTP avec node.js
Section
R5_10: Département Informatique
IUT GON Campus 3
Auteur
E.Porcq
Date
13/10/2025 durée 6h00

Dans le TD, les fichiers XXX sont fournis et les fichiers YYY sont à créer

1 Accès à une base mongodb par scripts

1.1 Préparation

- Lancer 2 consoles
- Sur la première lancer le démon mongod
- Sur la seconde, lancer mongosh
- Prévoir une troisième console (pour exécuter les scripts mongo et node)

1.2 Accès à la base en MongoScript (javascript)

https://www.mongodb.com/docs/mongodb-shell/write-scripts/
http://harry-wanki.developpez.com/tutoriels/mongodb/debuter-mongodb-introduction-base-donnees-nosql/
https://geekflare.com/fr/mongodb-queries-examples/

Lors du TP2, on a déjà expérimenté ce type de programmation, pratique mais basique.
Il est possible d'améliorer un peu ces programmes.
2 exemples ont été concaténés dans p12_1_etudiant.js.

- Créer une base maBDDTP3
- Créer une collection macollecs (sensible à la casse)
- Corriger ce programme pour qu'il utilise des fonctions. Tester ces fonctions.

1.3 Utilisation du module natif MongoDB pour node.js Version 3 ou plus de node.

https://www.npmjs.com/package/mongodb
debuter-avec-mongodb-pour-node-js.pdf
https://www.mongodb.com/docs/drivers/node/current/connect/mongoclient/
https://www.mongodb.com/docs/drivers/node/current/usage-examples/find/

- Il faut installer le module mongodb.-
- ~~Sur un poste de l'IUT~~
  - ~~net use x : \\ruche\porcqe /user:porcqe~~
  - ~~créer un dossier npm dans C:\Users\<votre session>\AppData\Roaming (obligatoire???)~~
  - ~~lancer les commandes :~~
    - ~~npm config set proxy http://10.103.0.8:3128~~
    - ~~npm config set https-proxy http://10.103.0.8:3128~~
- npm install mongodb --save

```
S:npm install mongodb --save
added 13 packages, and audited 14 packages in 21s
found 0 vulnerabilities
```

- nEn utilisant l'exemple fourni sur le site npmjs, donner le code permettant de se connecter à votre base p13_1.js.
  - Essayer aussi sans le connect()
- Réaliser un programme permettant d'insérer ces documents. P13_2.js
  { _id: 1, matiere: 'Anglais', importance: 1 },
  { _id: 2, matiere: 'EGO', importance: 2 },
  { _id: 3, matiere: 'PPP', importance: 2 },
  { _id: 4, matiere: 'Maths', importance: 2 },
  { _id: 5, matiere: 'BDD', importance: 4 },
  { _id: 6, matiere: 'UML', importance: 3 },
  { _id: 7, matiere: 'NoSQL', importance: 3 }
  - tester aussi avec une adresse IP fausse ou un port faux
- Compléter ce programme pour y afficher les documents d'importance supérieure à 2 triés par matière. P13_3.js

1.4 Utilisation du module Mongoose pour node.js

~~http://atinux.developpez.com/tutoriels/javascript/mongodb-nodejs-mongoose/~~
~~https://www.frugalprototype.com/api-mongodb-mongoose-node-js/~~
https://mongoosejs.com/docs/
https://mongoosejs.com/docs/models.html#compiling -- explication du pluriel

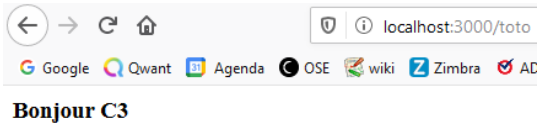
- Il faut (éventuellement) installer le module mongoose.
```
S:\prof>npm install mongoose --save
added 90 packages, and audited 91 packages in 22s
```
- Lire l'annexe 2 pour comprendre les différences entre ces 2 modules
- Tester les programmes p14_1.js et p14_2.js .
- Créer une nouvelle collection
- Donner un exemple de programmes (tester avec node, mongosh et load) permettant :
  - d'enregistrer un document (nom, prénom, adresse) p14_3.js . (tester avec une collection se terminant ou pas par un s et avec ou pas une majuscule)
  - de lire les documents p14_4.js
  - de modifier un document p14_5.js
  - de supprimer un document. p14_6.js

2 Application « client serveur »

2.1 Serveur d'écoute HTTP (ne pas refaire l'an prochain)

http://openclassrooms.com/courses/des-applications-ultra-rapides-avec-node-js/une-premiere-application-avec-node-js
http://nodejs.org/api/http.html

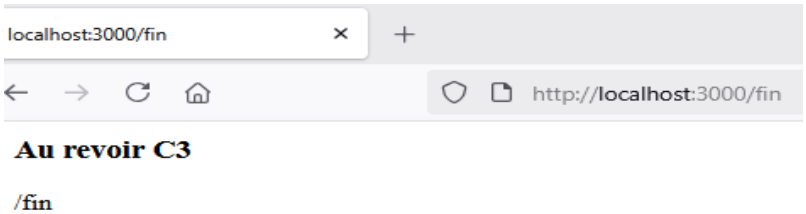
- Tester le premier exemple (p21_11.js) et chercher des informations sur la bibliothèque http (voir aussi annexe 1).
- Tester et commenter le code de l'application p21_12.js
  - avec http://localhost:3001/
  - avec http://localhost:3000/
  - avec http://localhost:3001/toto



- Ecrire une application (p21_13.js) qui lors d'un appel de la page fin :
  - afiche un message dans la console
  - afiche un message dans le navigateur
  - arrête le serveur.

```
S:\prof>node p21_13.js
Serveur en écoute sur le port 3000
param : /bonjour
-----------------------------------
param : /toto
-----------------------------------
param : /fin
-----------------------------------
A bientôt

S:\prof>
```



- Ecrire une application (p21_14.js) qui lors d'un appel à la page
  - quelconque, affiche la fonction et les paramètres

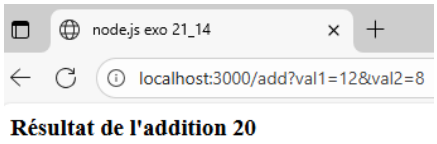
Exemple : http://localhost:3000/debut?x=5&y=6

```
S:\prof>node p21_14.js
Serveur en écoute sur le port 3000
-----------------------------------------
la page complète est : /debut?x=5&y=6
-----------------------------------------
la page est : /debut
-----------------------------------------
les paramètres sont  x=5&y=6
-----------------------------------------
[Object: null prototype] { x: '5', y: '6' }
-----------------------------------------
```

Utiliser le module url pour découper la page et les paramètres.
Utiliser le module querystring pour mettre les paramètres dans un tableau.

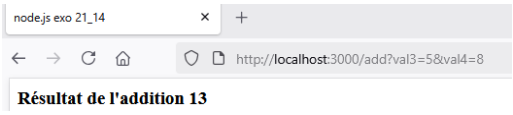
- add, appelle une fonction d'addition recevant les paramètres passés à l'URL. Le résultat sera affiché sur la page web et dans la console.
  - Version simple : on utilise le nom des paramètres en dur dans le code
  - Exemple : http://localhost:3000/add?val1=12&val2=8

```
S:\prof>node p21_14_V1.js
Serveur en écoute sur le port 3000
- 1 ----------------------------------------------------------
la page complète est : /add?val1=12&val2=8
- 2 ----------------------------------------------------------
la page est : /add
- 3 ----------------------------------------------------------
les paramètres sont  val1=12&val2=8
- 4 ----------------------------------------------------------
[Object: null prototype] { val1: '12', val2: '8' }
- 5 ----------------------------------------------------------
undefined
Résultat : 20
- 6 --------------------------
- 7 --------------------------
```



  - Version complète : on détecte le nom des paramètres dans le code
  - Exemple : http://localhost:3000/add?val3=5&val4=8

```
S:\prof>node p21_14_V2.js
Serveur en écoute sur le port 3000
- 1 ----------------------------------------------------------
la page complète est : /add?val3=5&val4=8
- 2 ----------------------------------------------------------
la page est : /add
- 3 ----------------------------------------------------------
les paramètres sont  val3=5&val4=8
- 4 ----------------------------------------------------------
[Object: null prototype] { val3: '5', val4: '8' }
- 5 ----------------------------------------------------------
undefined
Résultat : 13
- 6 --------------------------
- 7 --------------------------
```



- Compléter le programme précédent, le nommer p21_15.js Il doit afficher un message sur l'événement close.

```
S:\prof>node p21_15.js
Serveur en écoute sur le port 3000
la page complète est : /fin
------------------------------------------------------------
la page est : /fin
------------------------------------------------------------
les paramètres sont : null
------------------------------------------------------------
[Object: null prototype] {}
------------------------------------------------------------
Res : [Object: null prototype] {}
 Bye Bye
```

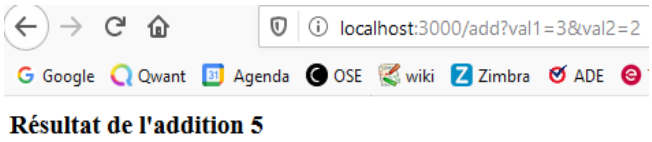
2.2 Le framework express.js

https://openclassrooms.com/courses/des-applications-ultra-rapides-avec-node-js/le-framework-express-js

Ce module permet d'accélérer le développement des applications WEB. A installer éventuellement.

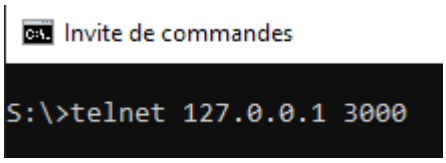
- Tester p22_11.js
- Compléter une application p22_12.js identique au p21_15.js mais en utilisant express.get().
  - Tester avec :
    - http://localhost:3000/
    - http://localhost:3000/add?val1=3&val2=2
    - http://localhost:3000/fin

```
S:\prof>node p22_12.js
Serveur en écoute sur le port 3000
3
2
5
```

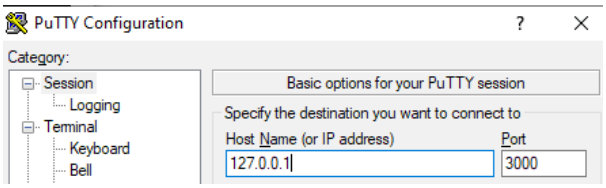


2.3 Serveur TCP

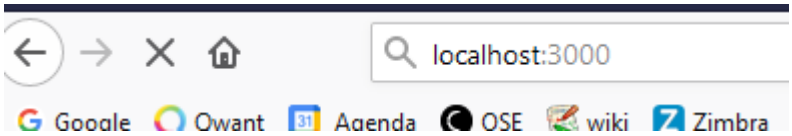
- Tester le premier exemple (p23_11.js) et chercher des informations sur le module net.
  - Il est possible d'utiliser, un navigateur, putty, telnet, SSH -p 3000 127.0.0.1



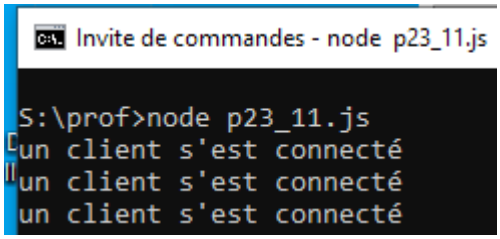
telnet



putty

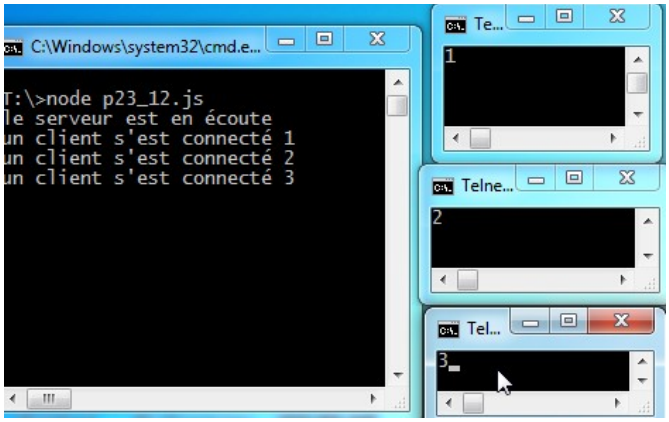


navigateur

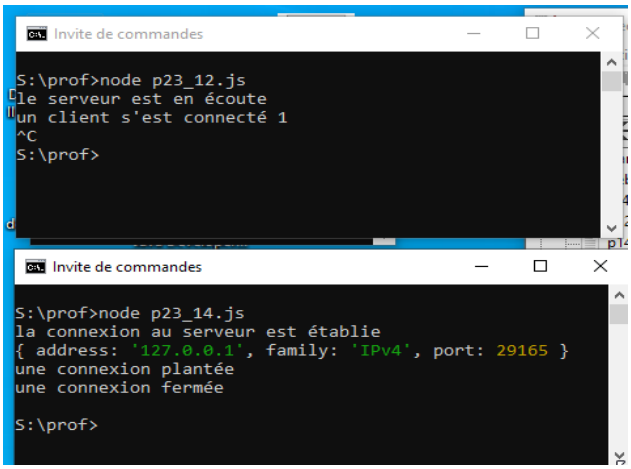


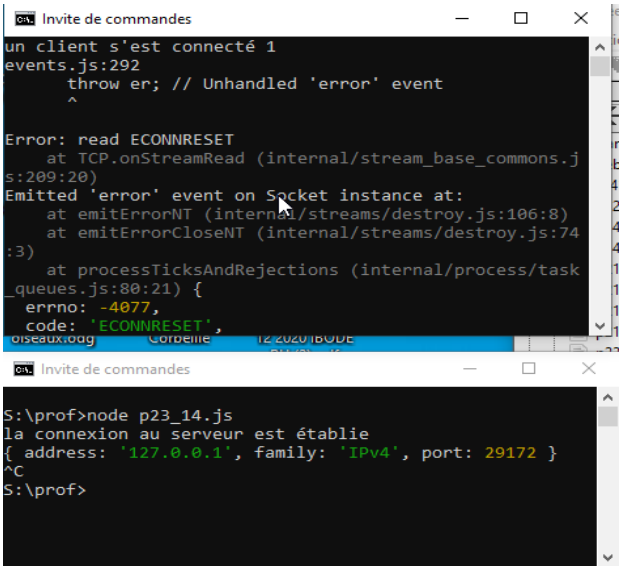
Serveur

- Le compléter pour qu'il envoie un message aux clients connectés et qu'il affiche les caractéristiques du serveur. (p23_12.js). Tester avec telnet



- Le compléter pour que le serveur s'arrête à la troisième connexion (p23_13.js)
- Tester un client se connectant au serveur en complétant le code . (p23_14.js).
  - Faites des essais en changeant le port de connexion





Annexe 1 : Node.js fonctions HTTP

server.listen(port,[callback])	méthode	Mettre le serveur en écoute sur le port indiqué. Il est possible d'associer une fonction qui sera appelée à la mise en écoute. L'événement listening est émis
Server.close([callback])	méthode	Fermer le serveur. Il est possible d'associer une fonction qui sera appelée lorsque la connexion sera fermée des deux côtés. L'événement close est émis.
Server.getConnections(callback)	méthode	la fonction est de la forme function(err,count) Count indique le nombre de clients connectés
Server.maxConnections	propriété	nombre max de clients autorisés à se connecter. La valeur par défaut est undefined (pas de limite)
{server|client}.address	méthode	retourne un objet contenant entre autres le port et l'adresse ip du serveur ou client
Client.connect(port,[host], [callback])	méthode	Permet de demander une connexion sur le port indiqué et éventuellement à l'adresse indiquée par host. Il est possible d'associer une fonction qui sera appelée, une fois la connexion établie. L'objet retourné est de type net.socket
client.end()	méthode	fermeture de la connexion par le client
Server on listening	événement	Le serveur est en attente de connexion
Server on connection	événement	La connexion au serveur est établie
{server|client} on close	événement	Le serveur et les clients ont fermé leur connexion
{server|client} error	événement	une erreur s'est produite
{server|client} data	événement	un paquet est arrivé

Annexe 2 : Mongoose et MongoDB native

http://voidcanvas.com/mongoose-vs-mongodb-native/

ORM or ODM – What is good for you?
ORM or Object Relational Mapping is based on the principal of having strict models or schemas.

Mongoose : Dans ORM, que mongoose utilise, vous devez définir la structure de votre schéma. Il doit y avoir un schéma fixe. ODM ou Object Document Mapping est le concept central de mongo lui-même; et il en va de même avec le pilote natif mongodb. Mongodb n'a besoin d'aucun schéma fixe. Vous pouvez insérer ou mettre à jour ce que vous voulez et comme vous le souhaitez. Le schéma permet de définir les types de variables et de structurer vos données (un peu comme si vous définissiez vos tables en SQL). Pour créer un Schema avec Mongoose, il suffit d'utiliser l'objet Schema. Les attributs peuvent être des schémas. *Les collections doivent se terminer par un s sinon, celui-ci est ajouté (à confirmer).*

Alors, est-ce un inconvénient d'avoir un schéma fixe? Non, certainement pas. Cela donne même une structure et une plus grande durabilité à votre code d'application. Cela n'entrave pas la fonctionnalité d'évolutivité de mongo; car si à l'avenir si votre application se développe et qu'il est nécessaire d'ajouter quelques champs supplémentaires, vous pouvez modifier le schéma et travailler en conséquence.

Mongodb native: Il y a des situations où un schéma fixe peut devenir une malédiction. Supposons que vous ayez une boutique de glossaire en ligne. Vous allez donc créer un schéma pour chaque type de produit; parce que tous ont des ensembles de propriétés différents? C'est probablement là que vous souhaitez utiliser une méthode de mappage de document et sélectionner natif mongodb pour cela.

	Mongoose	**Mongodb native**
Object mapping	ORM	ODM
Schema	Mandatory	Not necessary
Performance / processing time	Not bad	Excellent
Development time	Fast	Average
Default promise	No	No
Maintainability	Easy	Little hard
Learning curve	Little high	Low
Community	Good	Good

MongoDB	Mongoose
Database management system	Object document mapper
Stores giant amounts of data	Manages relationships between data
Supports multiple languages	Works only with MongoDB
Stores collections in the database	Defines schema for collections

Annexe 3 : Fonctions asynchrones

- **Promesses :** En JavaScript, une promesse représente une abstraction d'une exécution non-bloquante asynchrone. Les promesses JS sont similaires aux Futures de Java ou aux Task du C#, si vous les avez déjà rencontrées.
  Les promesses sont typiquement utilisées pour des opérations de réseau et d'I/O : par exemple, lire un fichier ou faire un appel HTTP. Au lieu de bloquer le *thread* d'exécution actuel, **nous créons une promesse asynchrone** et nous utilisons la méthode **then** pour attacher **une *callback*** qui sera activé quand la promesse sera complète. La *callback* peut elle-même retourner une promesse, et ainsi nous pouvons chaîner des promesses efficacement.**ORM ou ODM - Qu'est-ce qui est bon pour vous?** ORM ou Object Relational Mapping est basé sur le principe d'avoir des modèles ou des schémas stricts.
- Le serveur NodeJS peut recevoir plusieurs demandes (request) de différents utilisateurs. Donc afin d'améliorer le fonctionnement, tous les API du NodeJS sont conçus pour soutenir Callback. Le "callback" est une fonction (function), elle sera appelée lorsque le NodeJs achève une tâche (task) précise.
- Maix mieux : Une fonction asynchrone est une fonction précédée par le mot-clé async, et qui peut contenir le mot-clé await. **async** et **await** permettent un comportement asynchrone, basé sur une promesse (Promise), écrite de façon simple, et évitant de configurer explicitement les chaînes de promesse.
- Fonction Async : Une fonction async est un raccourci pour définir une fonction qui retourne une promesse.
- Découvrons maintenant await : Il peut uniquement être utilisé avec des fonctions async, et permet d'attendre de façon synchrone une promesse. Si on utilise une promesse en dehors d'une fonction async on doit utiliser les callback then.