VS code AS400

Table des matières

[1 Introduction 5](#_Toc84236156)

[1.1 Le language 5](#_Toc84236157)

[1.2 Ressources 5](#_Toc84236158)

[2 Installation 6](#_Toc84236159)

[2.1 Windows – Installation de GO 6](#_Toc84236160)

[2.2 Windows – Installation de Vs Code 6](#_Toc84236161)

[3 Découvrir le go 6](#_Toc84236162)

[3.1 Introduction 6](#_Toc84236163)

[3.2 Premier Programme 6](#_Toc84236164)

[3.3 Exécuter 6](#_Toc84236165)

[3.4 Les packages 6](#_Toc84236166)

[3.5 Les Imports 6](#_Toc84236167)

[3.6 Organisation des fichiers 6](#_Toc84236168)

[3.7 Tester rapidement avec Go Playground 6](#_Toc84236169)

[3.8 Explorer la documentation 6](#_Toc84236170)

[4 Les bases du langage 7](#_Toc84236171)

[4.1 Déclaration des variables 7](#_Toc84236172)

[4.2 Visibilité des variables 7](#_Toc84236173)

[4.3 Quiz des variables 7](#_Toc84236174)

[4.4 Contrôle de flux if else 7](#_Toc84236175)

[4.5 Contrôle de flux switch 7](#_Toc84236176)

[4.6 Conversion de type 7](#_Toc84236177)

[4.7 Les fonctions 7](#_Toc84236178)

[4.8 Retour de fonction multiple 7](#_Toc84236179)

[4.9 Quiz les fonctions 7](#_Toc84236180)

[4.10 Array 7](#_Toc84236181)

[4.11 Slices 7](#_Toc84236182)

[4.12 Contrôle de flux boucle for 7](#_Toc84236183)

[4.13 Range 7](#_Toc84236184)

[4.14 Gestion d’erreur en Go 7](#_Toc84236185)

[4.15 Gestion d’erreur dans un fichier 7](#_Toc84236186)

[4.16 Defer 8](#_Toc84236187)

[4.17 Quiz for err defer 8](#_Toc84236188)

[4.18 Exercice Goplace 8](#_Toc84236189)

[4.19 Corrigé Goplace 8](#_Toc84236190)

[4.20 Corrigé 8](#_Toc84236191)

[4.21 Corrigé 8](#_Toc84236192)

[4.22 Sources Go Place 8](#_Toc84236193)

[5 Structs 8](#_Toc84236194)

[5.1 Définition d’un Struct 8](#_Toc84236195)

[5.2 Re 8](#_Toc84236196)

[6 Projet : jeu du pendu 9](#_Toc84236197)

[6.1 Structure du jeu 9](#_Toc84236198)

[6.2 Re 9](#_Toc84236199)

[7 Maps 9](#_Toc84236200)

[7.1 Définition d’un map 9](#_Toc84236201)

[7.2 Re 9](#_Toc84236202)

[8 Projet Dictionnaire & Stockage Bdd 9](#_Toc84236203)

[8.1 Structure du projet dictionnaire 9](#_Toc84236204)

[8.2 Re 9](#_Toc84236205)

[9 Interface 9](#_Toc84236206)

[9.1 Interêt d’un interface 9](#_Toc84236207)

[9.2 Re 10](#_Toc84236208)

[10 Projet : Générateur de certificat 10](#_Toc84236209)

[10.1 Structure du projet générateur de certificat 10](#_Toc84236210)

[10.2 Re 10](#_Toc84236211)

[11 Goroutines & channels 10](#_Toc84236212)

[11.1 Programmation concurante 10](#_Toc84236213)

[11.2 Re 10](#_Toc84236214)

[12 Projet imgproc – Goroutines & Workers 10](#_Toc84236215)

[12.1 Structure du projet de filtres d’images 10](#_Toc84236216)

[12.2 Re 10](#_Toc84236217)

[13 Approfondir Go 11](#_Toc84236218)

[13.1 Constantes 11](#_Toc84236219)

[13.2 Re 11](#_Toc84236220)

[14 Go et le Web 11](#_Toc84236221)

[14.1 L’approche de go en Web 11](#_Toc84236222)

[14.2 Re 11](#_Toc84236223)

[15 Modules 11](#_Toc84236224)

[15.1 Pourquoi les modules en Go 11](#_Toc84236225)

[15.2 Re 11](#_Toc84236226)

[16 Projet : Goflix – http & BDD 12](#_Toc84236227)

[16.1 Structure du projet GoFlix 12](#_Toc84236228)

[16.2 Re 12](#_Toc84236229)

# Introduction

## Vs code et AS400

[Coding in RPG (IBM i/AS400). Using Deploy tool in VS Code/Code for IBM i - YouTube](https://www.youtube.com/watch?v=cynjHysp7sg)

VS / Code for IBM i

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

CREER le dossier, **c**:\WORK\AS400\deploy1

CREER sur le 400 dans l’ifs, /HOME/TVO/deploy1

Dans deploy1

**CREER les dossiers**  QRPGLESRC & .VSCODE  
dans QRPGLESRC, CREER le fichier source srcpf : uhello.rpgle

Dans .VSCODE, CREER le fichier settings.json

Dans .VSCODE, CREER le fichier actions.json

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

…

Dans deploy1, CREER le fichier makefile

GITUH/ [christianlarsen\_rpgle/makefile at main · christianlarsen/christianlarsen\_rpgle · GitHub](https://github.com/christianlarsen/christianlarsen_rpgle/blob/main/makefile)

Celui du cours avec CLV

Une image contenant texte, logiciel, Logiciel multimédia, Icône d’ordinateur

Description générée automatiquement

Créer mon makefile avec TVO, un à créer par ‘projet’, ici deploy1

Une image contenant capture d’écran, texte, logiciel, Logiciel multimédia

Description générée automatiquement

**OUVRIR** dans vs code, le DOSSIER : deploy1

**Connexion** à l’AS400, the IBMi…

A gauche : Une image contenant capture d’écran, Police, Graphique, logo

Description générée automatiquement

(il faut créer la connexion 172.24.7.3 tvo tt01)

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

…Fermer les 3 messages on le fait étape par étape

**CREER** le dossier sur l’ifs dans ma curlib /home/tvo/deploy1

Une image contenant texte, Police, logiciel, ligne

Description générée automatiquement

Faire un click droit sur le dossier de l’ifs : /home/tvo/deploy1

Et : définir l'emplacement de l'espace de travail de déploiement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Ok : c’est défini. Il sait maintenant que c’est à cet emplacement sur l’IFS (on ferme la croix)

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Revenir sur le code source et click sur Deploy

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Sélectionner : all

* Ça ve tout copier dans le dossier deploy1 sur l’ifs, dont le dosser qrpglesrc

[Coding in RPG (IBM i/AS400). Using Deploy tool in VS Code/Code for IBM i - YouTube](https://www.youtube.com/watch?v=cynjHysp7sg)

## Accès codes sources AS400 depuis VS CODE

AUTRE : idem STRPDM direct sur les Bib du 400

<https://www.youtube.com/watch?v=IRkCLZDv01Q>

Compilation dans vs code

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

<https://www.youtube.com/watch?v=IRkCLZDv01Q>  
vidéo à 7 ‘29’’

autres :

<https://www.youtube.com/watch?v=TYUK2TsxQjc>

# Node js sur IBMi

## Application todoapp1

<https://www.youtube.com/watch?v=TYUK2TsxQjc>

Application **todoapp1**, qui est une todolist

**CREER** le dossier todoapp1, sur l’ifs depuis le powerShell de Vs Code

**CHANGER** le home dir de tvo ( comme nadtvo )

WRKUSRPRF USRPRF(NADTVO ASTLVL(\*INTERMED) )

**2** NADTVO Tests iD400 - Thierry VOZELLE

Répertoire initial . . HOMEDIR /home/xxx xxx=tvo

**LANCER** le shell QP2TERM : call QP2TERM

DROITS : **chmod 755 /home/xxx**

**INITIALISER** le projet dans le dossier /home/tvo/**todoapp1**

Cd todoapp1 puis Npm init

**INSTALLER** express sur l’ifs avec qshell : npm i express ejs idb-pconnector

**CREER** le fichier serveur.js qui va être notre serveur

**DEMARRER** dans qshell QP2TERM: cd todoapp1 node serveur.js

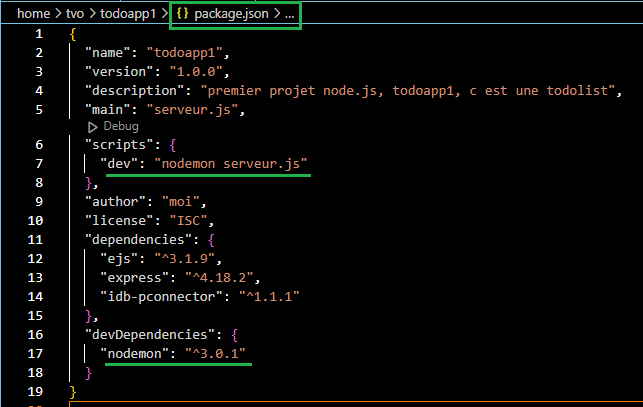
TEST : <http://172.24.7.3:3030/>

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

**INSTALLER** nodemon, dans qshell QP2TERM: npm i -D nodemon

Corriger le package.json …



**DEMARRER** dans qshell QP2TERM: cd todoapp1 node serveur.js

**DEMARRER avec nodemon** dans le qshell IBMi QP2TERM: cd todoapp1 npm run dev

Une image contenant capture d’écran, Police, vert

Description générée automatiquement

Une image contenant texte, capture d’écran, affichage, Police

Description générée automatiquement

**CREER** un index.html

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

-

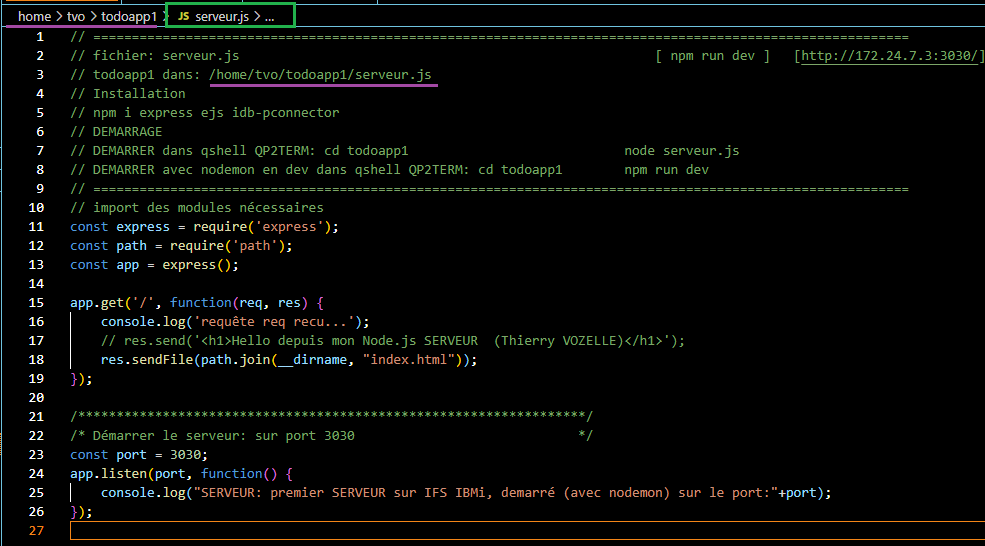
**MODIFIER** serveur.js

Béta 1

Une image contenant capture d’écran, texte, Police

Description générée automatiquement

Amélioré avec **\_\_dirname**



TEST : <http://172.24.7.3:3030/>

## Les vues EJS

View engine ‘ejs’

Les vues EJS contiennent du code "classique" : html + css + JavaScript "client" (exécuté par le navigateur), mais aussi du code JavaScript "serveur" qui sera exécuté par le serveur avant envoi au client.

sendFile() devient render() :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

**CREER** un dossier : views

**CREER** un fichier  : views/index.ejs

Ejs permet donc de passer des paramètres à la page html et d’afficher ces paramètres avec la directive :

**<%=** paramètre **%>**

Exemple :

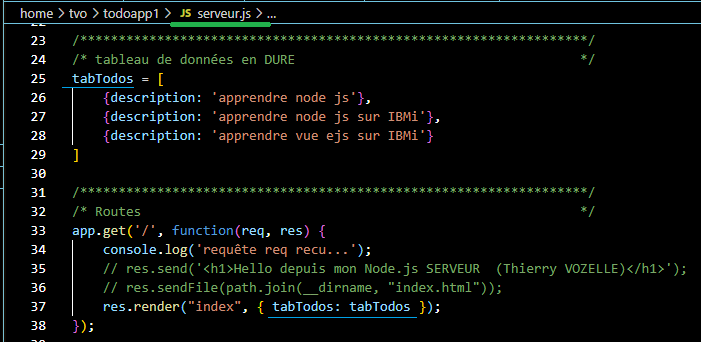
Une image contenant texte, capture d’écran, logiciel, affichage

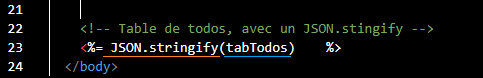
Description générée automatiquement

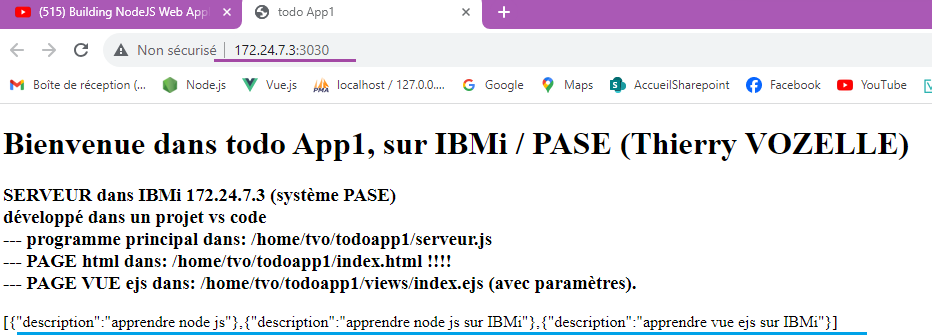
Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Exemple plus concret avec un tableau de todos : tabTodos[]







…

Utilisation de la directive **<% for(let item of tabObjet){ %> … <%}%>** dans ejs

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Une image contenant texte, Police, ligne, algèbre

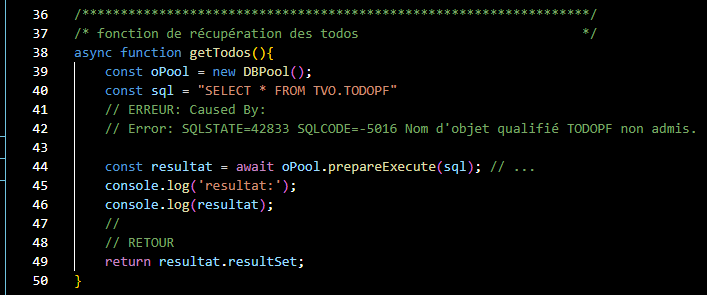
Description générée automatiquement

## fonction getTodos() avec : idb-pconnector

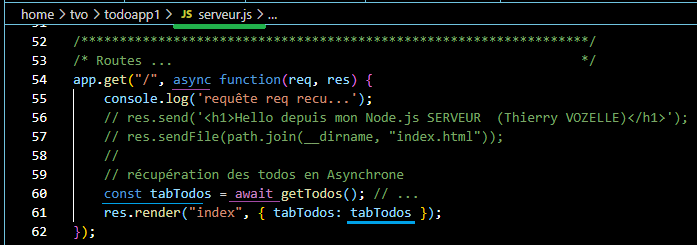
On va utiliser une connexion à la base de données IBMi DB2

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement



C’est une fonction asynchrone avec async & **await**; A l’utilisation de la fonction il faudra utiliser le mot-clé **await**

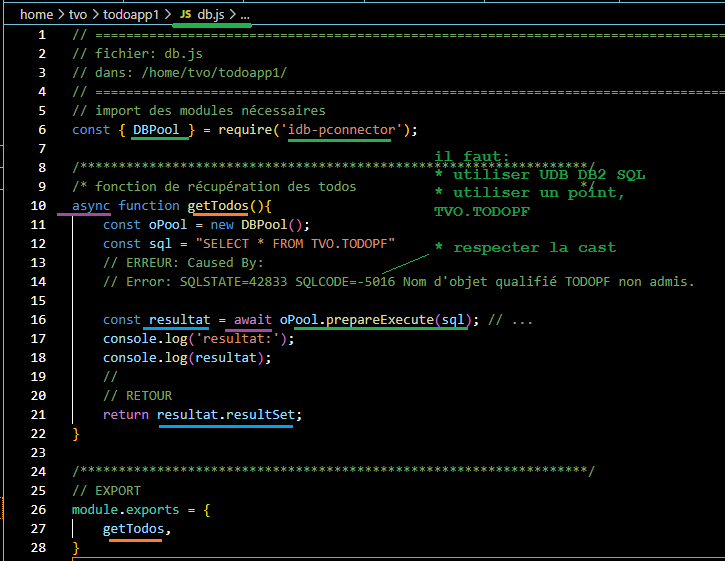


## Création d’un fichier db.js

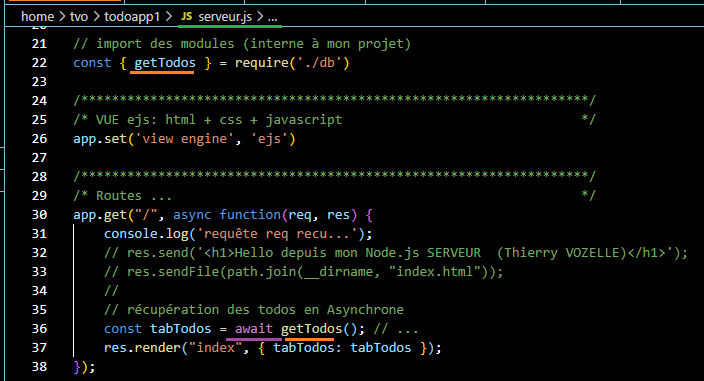
CREER sur l’ifs dans, [\\172.24.7.3\home\TVO\todoapp1](file:///\\172.24.7.3\home\TVO\todoapp1) le fichier **db.js**

Dans ce fichier on va y séparer l’accès à la base de données du serveur pour alléger le fichier serveur.js

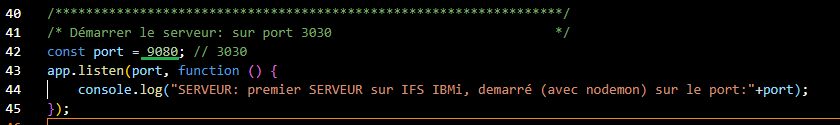
Db.js



Import et utilisation en asynchrone dans serveur.js



**Utiliser** le port web 9080



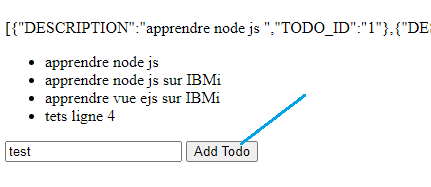
## Formulaire

Dans le fichier index.ejs

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Test : <http://172.24.7.3:9080/>



Si on click sur Add todo, l’url change : et deviens, <http://172.24.7.3:9080/add?DESCRIPTION=test>

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

ERREUR

Il faut changer la méthode en **POST**, et écrire le post dans le serveur, route app.post avec /add

Une image contenant texte, capture d’écran, Police, affichage

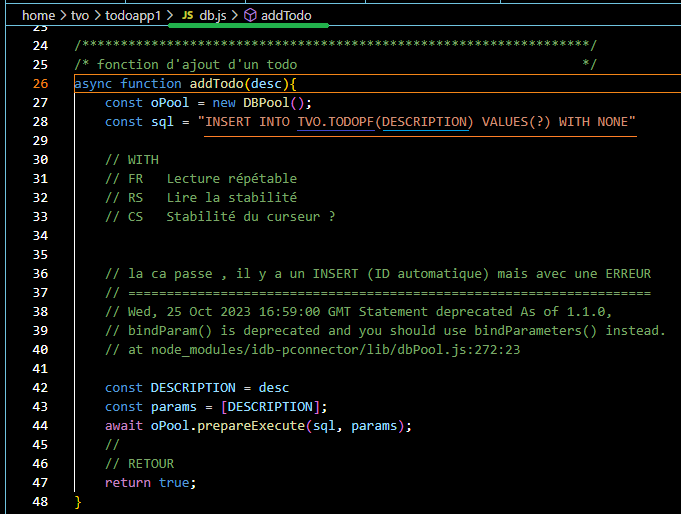
Description générée automatiquement

## Fonction addTodo

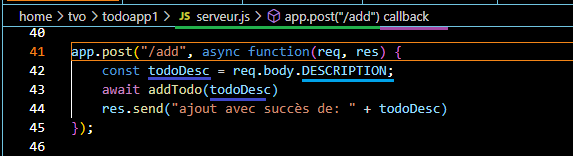
Dans le fichier db.js on ajoute la fonction addTodo() qui va ajouter à la base de données UDB DB2

Soucis avec les paramètres, par rapport au cours, il faut préciser les zones à mettre à jour entre () dans le sql

const sql = "INSERT INTO TVO.TODOPF(DESCRIPTION) VALUES(?) WITH NONE"



**Utilisation** dans le serveur, en async await (Asynchrone) dans serveur.js, la / les zone(s) fichiers doit être identique en respectant la cast MAJUSCULE sur AS400, exemple (DESCRIPTION)

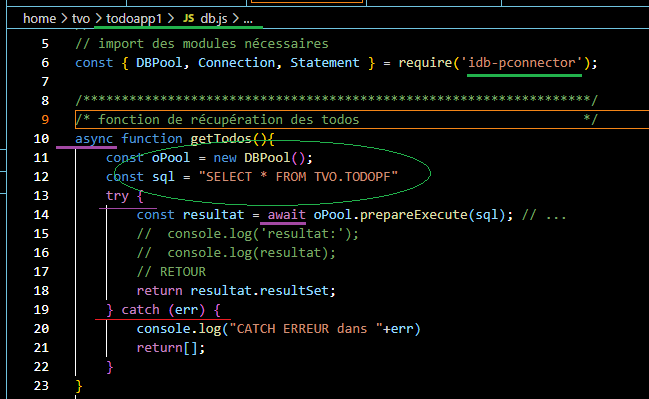


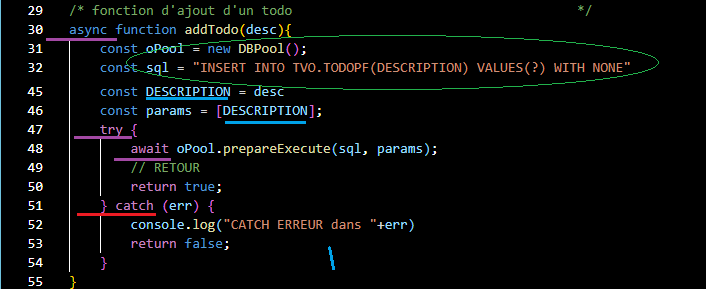
## Application todoapp1 finalisé

GITHUB : <https://github.com/yusy4code/ibmi-web-app/blob/master/db.js>

Ajout des try catch, dans les accès critique base de données

Dans db.js

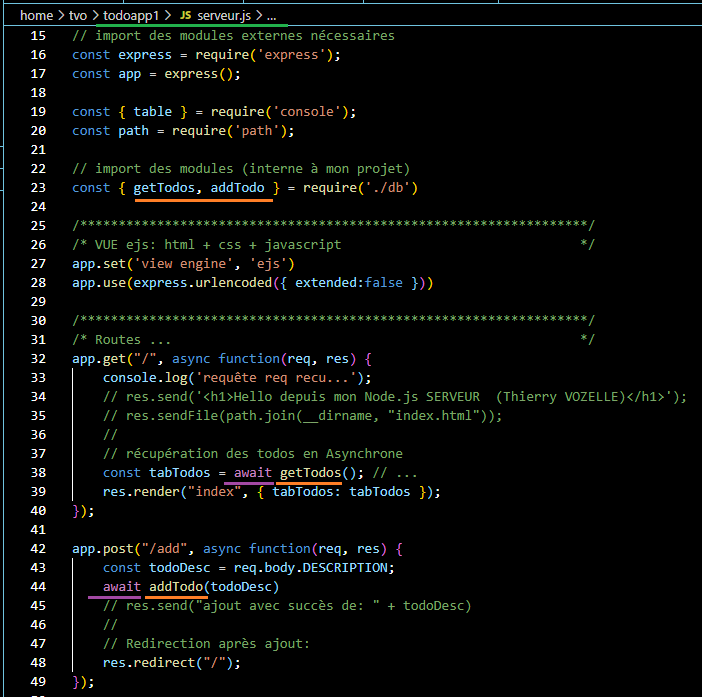




Une image contenant capture d’écran, texte, Logiciel multimédia

Description générée automatiquement

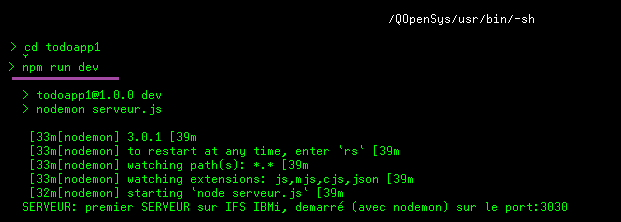
**SERVEUR** :



**Test** : Session 400 : call **QP2TERM** : cd todoapp1 + npm run dev

Une image contenant Police, vert, capture d’écran

Description générée automatiquement



**url** : <http://172.24.7.3:3030/>

Une image contenant texte, Appareils électroniques, capture d’écran, logiciel

Description générée automatiquement

# IBM i

## AS400 & HTML

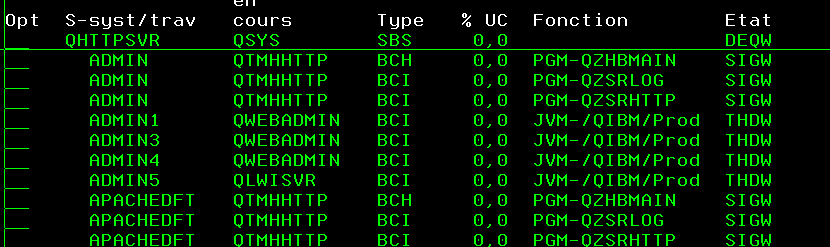
<https://www.youtube.com/watch?v=8s0RK-T2Wgg&list=PL-cQM7QhBjZQRjoWA_IM4NAc5799nKfP0&index=2>

WRKACTJOB SBS(QHTTPSVR)

Une image contenant texte, capture d’écran

Description générée automatiquement

Démarrage  
**STRTCPSVR** SERVER(\*HTTP) HTTPSVR(\*ADMIN)  
**STRTCPSVR** SERVER(\*HTTP) HTTPSVR(APACHEDFT) HTTP server starting.



<http://172.24.7.3/index.html>

wrklnk '**/www**'

5 apachedft

5 htdocs

5 index.html

…

## Comment exécuter RPGLE à partir d’un navigateur Web

Programme USAMPLECGI

-

FFilename++IPEASF.....L.....A.Device+.Keywords+++++++++++++++++++++++++++++Comments++++++++++

FLOGF O E DISK

\* ========================================================================\*

\* SDS

\* ========================================================================\*

D PSDS SDS 429

D pgmjob 244 253a . Nom JOB

D pgmuser 254 263a . Utilisateur JOB

D pgmjobnum 264 269s 0 . N° JOB

D pgmdate 276 281 . Date Système

C\*

/free

// log = N° job + pgm user + pgm job + timestamp

LOGMSG = 'Programme invoké depuis ' +

'NoJob/pgm user/pgm job: ' +

%char(pgmjobnum)+'/'+

%trim(pgmuser)+'/'+

%trim(pgmjob)+' at ' +

%char(%Timestamp());

//

// ECRITURE

WRITE LOGFMT;

//

// FIN

\*INLR = '1';

RETURN;

/end-free

\* ===FinDuSource=====

-

WRKACTJOB SBS(QHTTPSVR)

Ne fais pas du tout comme le cours, le serveur appache n’est pas le même

## Comment communiquer de l’AS400 RPGLE au navigateur Web

<https://www.youtube.com/watch?v=L1mVRUSCVqY&list=PL-cQM7QhBjZQRjoWA_IM4NAc5799nKfP0&index=3>

service programme, wrkobj QHTTPSVR/QZHBCGI   
DSPSRVPGM SRVPGM(QHTTPSVR/QZHBCGI)

-

Programme de service . . . . . . . . . . : QZHBCGI

Bibliothèque . . . . . . . . . . . . . : QHTTPSVR

Propriétaire . . . . . . . . . . . . . . : QSYS

Attribut du programme de service . . . . : CPPLE

Détail . . . . . . . . . . . . . . . . . : \*PROCEXP

Exportations de procédure :

**Procédure**

QtmhGetEnv

QtmhRdStin

QtmhWrStout

QtmhCvtDb

QtmhPutEnv

QzhbCgiParse

QzhbCgiUtils

QzhbCgiRecvState\_r

QzhbCgiSendState\_r

QzsrPutEnvCCSID

QzsrGetEnvCCSID

sendMessageCatalogError

openPrimaryMessageCatalog

-

**MODIFIER** le programme USAMPLECGI, qui va utiliser le programme de service QZHBCGI

**Directive**, créer le module puis créer le programme ( avec programme de service )

CRTPGM PGM(TVO/USAMPLECGI) MODULE(TVO/USAMPLECGI) BNDSRVPGM((QHTTPSVR/QZHBCGI))

Test : <http://172.24.7.3/tvo/usamplecgi>

Ne fonction pas !

## Comment consommer une API externe à l’aide de http\_GET dans IBMi

<https://www.scottklement.com/httpapi/> + savefile

<https://www.scottklement.com/httpapi/httpapi_savf.html>

télécharger le savf et suivre la procédure

* Log onto IBM i, and create a save file to store the store the distribution in. To do this, type:
  + **CRTSAVF** QGPL/HTTPAPI (en premier si pas de quote site namefmt 1)
* Download my [HTTP API distribution](https://www.scottklement.com/httpapi/httpapi.savf) and save it to your PC -- or to IBM i (directly into the SAVF.)
* If you sent it to your PC, FTP the distribution from your PC to IBM i. To do this:
  + Open an MS-DOS Prompt.
  + Type: **cd** \directory\where\you\put\httpapi.savf
  + Type: **ftp** your-ibmi-name-here
  + Type: (enter your username & password when asked)
  + Type: **binary**
  + Type: **put** httpapi.savf QGPL/HTTPAPI
  + Type: quit
* Back on IBM i, type:
  + DLTLIB LIBHTTP (ignore errors if library doesnt exist)
  + RSTLIB SAVLIB(LIBHTTP) DEV(\*SAVF) SAVF(QGPL/HTTPAPI)
* Build & run the install program:
  + CHGCURLIB CURLIB(LIBHTTP)
  + CRTCLPGM INSTALL SRCFILE(LIBHTTP/QCLSRC)
  + CALL INSTALL
* The install program will guide you through the rest of the process.

Bib : LIBHTTP -- Exemple : TVO/QRPGLESRC(UGETCURR)

**Erreur :**  
HTTP/1.1 404 Not Found   
Exception non interceptée à la ligne 46

46 http\_stmf(

47 'GET':

48 URL:

49 IFS

Consultation de la log : wrklnk '/tmp/htt\*'

* Github = readme : <https://github.com/fixerAPI/fixer#readme>

## Comment créer un service Web basé REST dans AS400

<https://www.youtube.com/watch?v=03eRD-JW9t8&list=PL-cQM7QhBjZQRjoWA_IM4NAc5799nKfP0&index=6>

## Moderniser votre IBMi avec une application Web

<https://www.youtube.com/watch?v=vDe7LYszzwg&list=PL-cQM7QhBjZQRjoWA_IM4NAc5799nKfP0&index=7>

appache !

## Consommer http GET à partir de RPGLE à l’aide de HTPGETCLOB

<https://www.youtube.com/watch?v=xX1eGwALKq4&list=PL-cQM7QhBjZQRjoWA_IM4NAc5799nKfP0&index=8>

using HTTPGETCLOB DB2 UDF   
select \* FROM sysfuncs

SELECT distinct(SPECIFIC\_SCHEMA) FROM sysfuncs

SPECIFIC\_SCHEMA

NADFIWOLF

NADFI0001

NADFI4444

NADFI5555

NADSPE\_W

QDBTSLIB

QGPL

QSYS2

QUSRBRM

SYSIBM

SYSIBMADM

SYSPROC

SYSTOOLS

TVO

\*\*\*\*\*\*\*\* Fin de données \*

select \* FROM sysfuncs WHERE SPECIFIC\_SCHEMA='SYSTOOLS'

....+....1....+.. ! 9....+...40....+...41...

SPECIFIC\_SCHEMA ! ROUTINE\_NAME

!

SYSTOOLS ! HTTPGETBLOBVERBOSE

SYSTOOLS ! HTTPGETBLOBVERBOSE

SYSTOOLS ! **HTTPGETCLOB**

SYSTOOLS ! **HTTPGETCLOB**

SYSTOOLS ! HTTPGETCLOBVERBOSE

SYSTOOLS ! HTTPGETCLOBVERBOSE

-

Syntaxe : Systools.hhtpgetclob(‘URL’,’’)

Select data

From **(** values

Char(SYSTOOLS.HTTPGETCLOB( ‘https://data.fixer.io/api/latest’, ‘’),4096 )

**) xx(data)**

-

Erreur

# Node JS sur IBM i – idb-**p**connector

## Rendre du HTML avec des donnes PF à l’aide de Node JS sur IBMi AS400

<https://www.youtube.com/watch?v=OJehGUgl9pE&list=PL-cQM7QhBjZQRjoWA_IM4NAc5799nKfP0&index=11>

On utilise idb-pconnector avec DBPool

Description du fichier USER1 dans TVO

Une image contenant texte, capture d’écran, horloge, Horloge numérique

Description générée automatiquement

PGM : SQLRPGLE de create table, TVO/QPRGLESRC(**UCRTUSER**)

SQL : STRSQL

insert into tvo/user1 + F4

INSERT INTO TVO/USER1(NOM, PRENOM) VALUES('VOZELLE', 'Thierry')

INSERT INTO TVO/USER1(NOM, PRENOM) VALUES(COESENS, 'Dounia')

## Initialisation du projet

CREER l’application DEMO1

**LANCER** le shell IBMi (AS400) QP2TERM : call QP2TERM

**INITIALISER** le projet dans le dossier /home/tvo/**demo1**

Changer de répertoire: **cd** **demo1**

**INITIALISER** le projet: **npm init** => cela va créer le package.json

**INSTALLER** express sur l’ifs avec qshell : npm install express ejs **idb-pconnector**

**INSTALLER** nodemon, dans qshell QP2TERM: npm i -D nodemon

**MOFIFIER** le fichier package.json pour activer le nodemon

**CREER** le fichier serveur.js qui va être notre serveur

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

**TESTER** : pour tester il faut obligatoirement passer par le qshell de IBMi c’est-à-dire : CALL QP2TERM

**DEMARRER** le serveur :

\*

CALL QP2TERM

CD demo1

NODE serveur.js

\*

TEST sur 3030: <http://172.24.7.3:3030/> => on peut utiliser le : **9080** pour être ‘visible’

## Fonction : getUsers

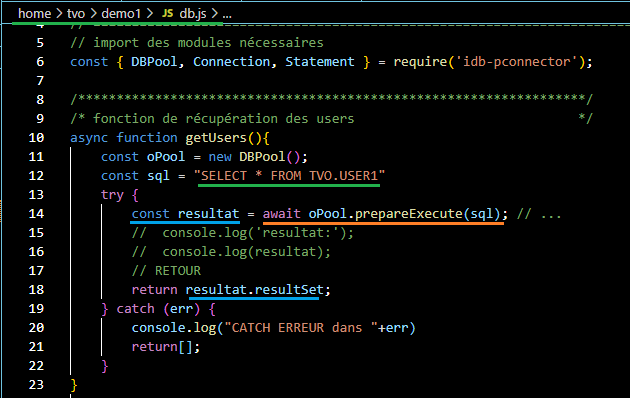
getUsers permet de récupérer tous les utilisateurs dans la table tvo/user1

syntaxe idb-pconnector :

sql = ‘SELECT \* FROM TVO.USER1’

resultat = oPool.prepareExecute(sql)

x



x

## Fonction : addUser

addUser permet d’ajouter un user dans la table tvo/user1

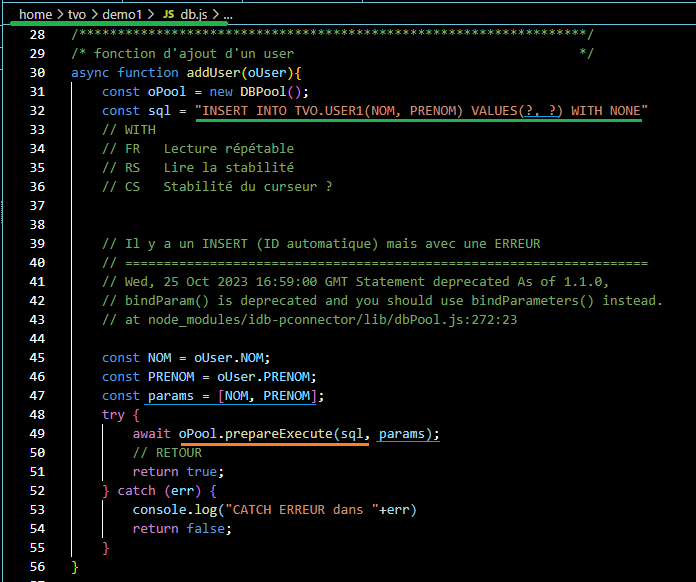
syntaxe idb-pconnector :

sql = ‘INSERT INTO TVO.USER1(NOM, PRENOM) VALUES( ?, ?) WITH NONE’

params=[NOM, PRENOM]

oPool.prepareExecute(sql, params)

x



## Fichier serveur : index

Dans le fichier serveurs.js on gère la route GET ‘/’ et le RENDU de la page index.ejs



## Fichier index.ejs

* Partie affichage de la liste des users





TEST

**DEMARRER** le serveur :

\*

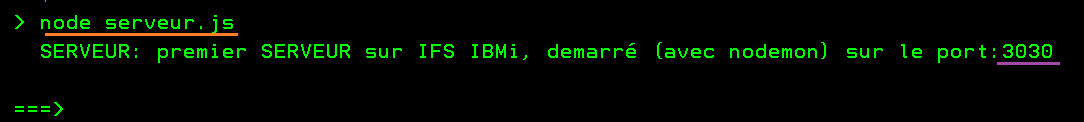
CALL QP2TERM

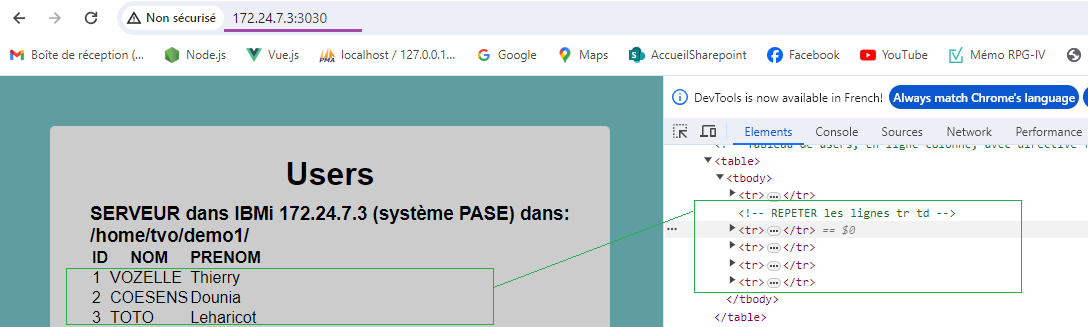
CD demo1

NODE serveur.js

\*

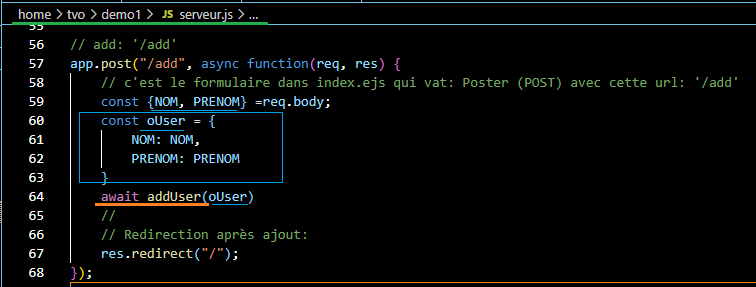
TEST sur 3030: <http://172.24.7.3:3030/>





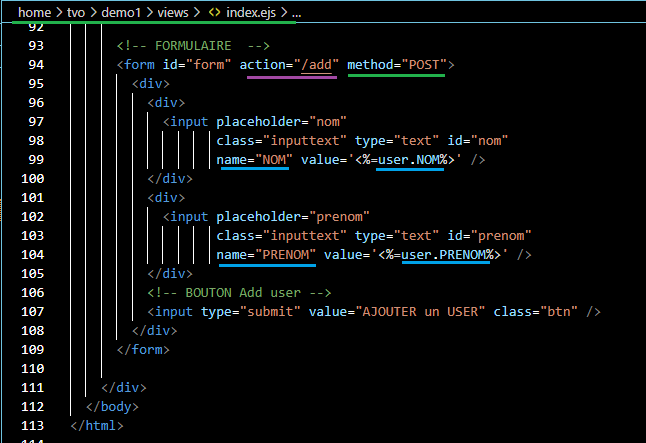
## Fichier serveur : add

Dans le fichier serveurs.js on gère la route POST ‘/add’ et le RENDU de la page index.ejs avec une redirection sur ‘/’



## Fichier index.ejs – Formulaire

* Partie affichage du formulaire avec POST ‘/add’ => ajouter un user

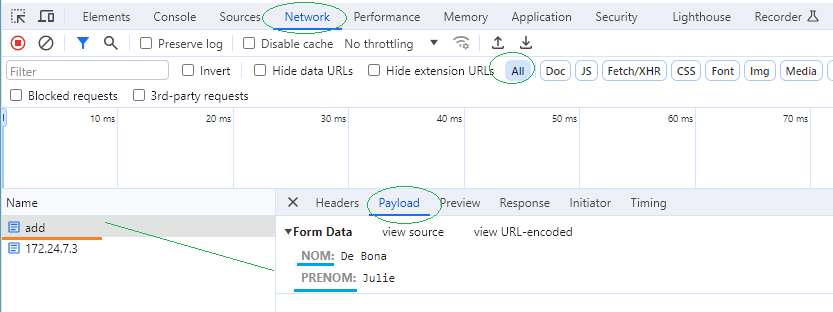


C’est la propriété name qui va donner le nom de la zone envoyé dans req.body

Name=’NOM’

Name=’PRENOM’

On peut retrouver les noms de zones dans le payload



TEST

**DEMARRER** le serveur :

\*

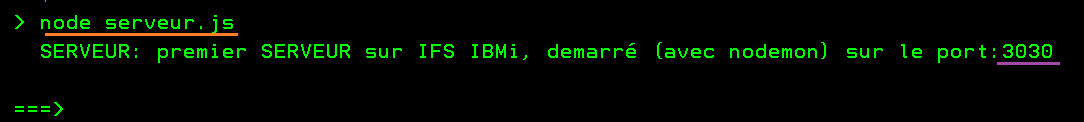
CALL QP2TERM

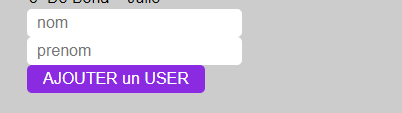
CD demo1

NODE serveur.js

\*

TEST sur 3030: <http://172.24.7.3:3030/>





x

## Fonction update user

updateUser permet de mettre à jour un user dans la table tvo/user1

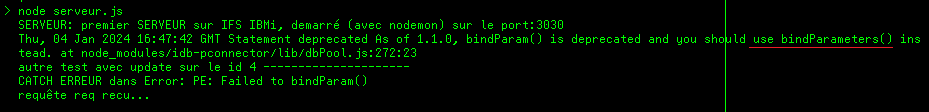
syntaxe idb-pconnector :

sql = ‘UPDATE TVO.USER1 SET NOM=?, PRENOM=? WHERE UID=?’

params=[NOM, PRENOM, ID]

oPool.prepareExecute(sql, params)

x



x

## Fonction delete user

deleteUser permet de supprimer un user dans la table tvo/user1

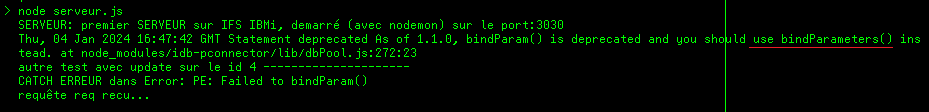
syntaxe idb-pconnector :

sql = ‘DELETE FROM TVO.USER1 WHERE UID=?’

params=[ID]

oPool.prepareExecute(sql, params)

x



X

# Node JS sur IBM i – idb-**c**onnector – demo2 - KO

## Rendre du HTML avec des donnes PF à l’aide de Node JS & idb-connector sur IBMi AS400

<https://www.youtube.com/watch?v=XMpOtjAn254>

On n’utilisera plus JAMAIS idb-connector avec : KO rien ne fonctionne.



[Cette photo](https://korben.info/appkiller-devenez-tueur-a-gages-sous-osx.html) par Auteur inconnu est soumise à la licence [CC BY-NC-ND](https://creativecommons.org/licenses/by-nc-nd/3.0/)

.

# Node JS sur IBM i – idb-**p**onnector – demo3

## Rendre du HTML avec des donnes PF à l’aide de Node JS & idb-**p**connector sur IBMi AS400

<https://testdemogitbook.gitbook.io/restify/new-tutorial-2-rest-apis-and-the-ibm-i>

On utilise idb-**p**connector avec :

Description du fichier utilisé : USER1 dans TVO

Une image contenant texte, capture d’écran, horloge, Horloge numérique

Description générée automatiquement

PGM : SQLRPGLE de create table, TVO/QPRGLESRC(**UCRTUSER**)

SQL : STRSQL

insert into tvo/user1 + F4

INSERT INTO TVO/USER1(NOM, PRENOM) VALUES('VOZELLE', 'Thierry')

INSERT INTO TVO/USER1(NOM, PRENOM) VALUES(COESENS, 'Dounia')

## Initialisation du projet

CREER l’application DEMO3

**LANCER** le shell IBMi (AS400) QP2TERM : call QP2TERM

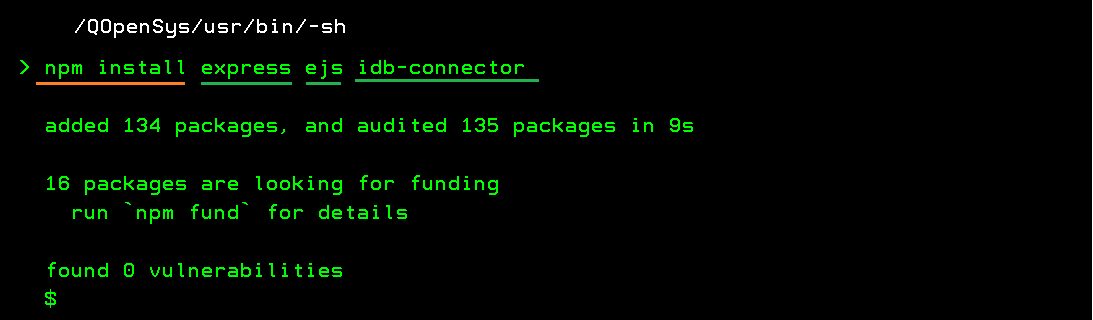
**INITIALISER** le projet dans le dossier /home/tvo/nodejs/**demo3**

Changer de répertoire: **cd** **nodejs/demo3**

**INITIALISER** le projet: **npm init** => cela va créer le package.json

**INSTALLER** express sur l’ifs avec qshell : npm install express ejs **idb-ponnector**

**INSTALLER** nodemon, dans qshell QP2TERM: npm i -D nodemon

****

**MOFIFIER** le fichier package.json pour activer le nodemon

**CREER** le fichier serveur.js qui va être notre serveur

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

**TESTER** : pour tester il faut obligatoirement passer par le qshell de IBMi c’est-à-dire : CALL QP2TERM

**DEMARRER** le serveur :

\*

CALL QP2TERM

CD nodejs/demo3

**NPM RUN** dev ou bien : **NODE** serveur.js

\*

TEST sur 3030: <http://172.24.7.3:3030/>demo3/user => on peut utiliser le : **9080** pour être ‘visible’

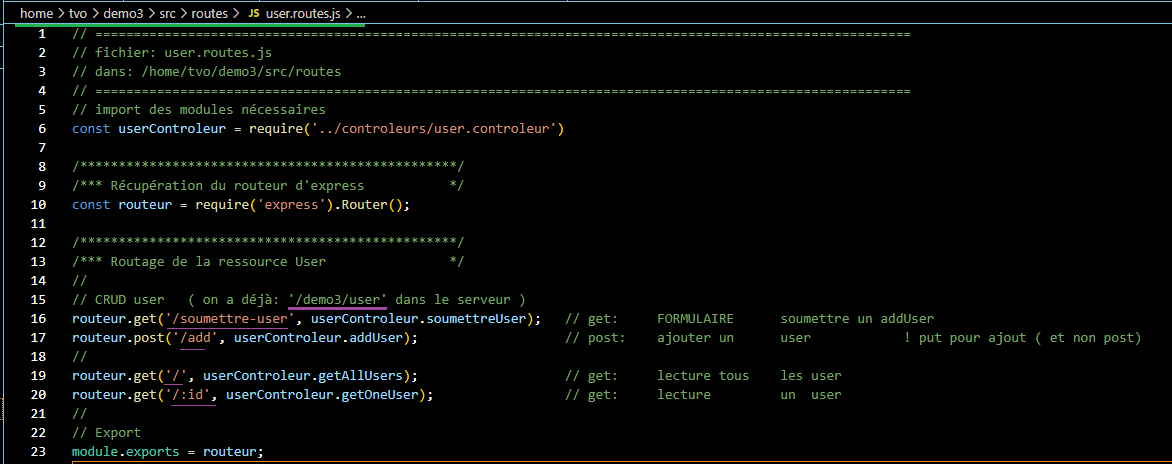
-

## Fichier: serveur.js



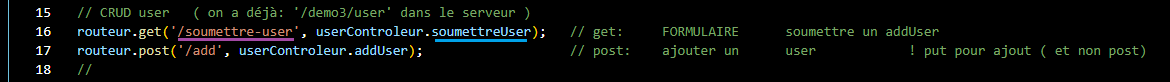
## Fichier des routes: user.routes.js

Il est dans src/routes/user.routes.js

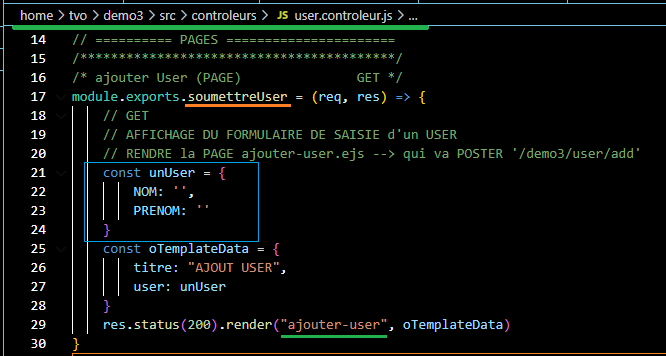


## Fichier contrôleur: Soumettre le Formulaire pour CREER un user

La route pour soumettre ce formulaire, route en GET, pour avoir le Formulaire à l’ECRAN

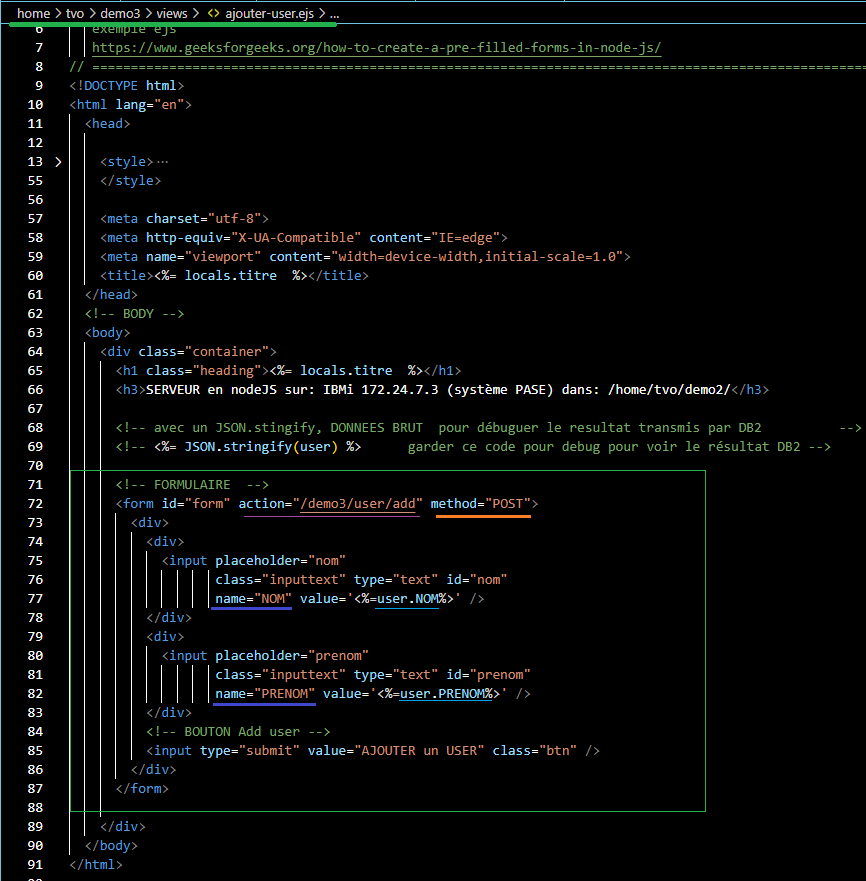


Le contrôleur : **soumettreUser**()



Ici il ne fait que rendre la PAGE du formulaire de saisie d’un user

La PAGE: **ajouter-user.ejs**



ATTENTION : bien définir la zone **name** de chaque **input** du formulaire, c’est la zone **name** qui est reprise dans le req.body & cette zone doit correspondre si possible au nom de la zone dans la table UDB DB2 IBMi, dans notre cas name : ( NOM & PRENOM en majuscule sont les zones de la table (fichier) udb db2 : tvo/user1

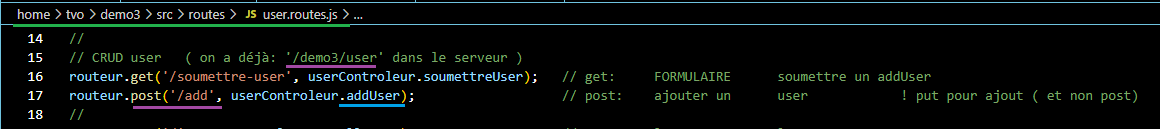
* NOM
* PRENOM

Le formulaire va soumettre en POST l’action : ‘/demo3/user/add’

## Fichier contrôleur: Ajouter un user

Le formulaire précédent va soumettre en POST l’action : ‘/demo3/user/add’

La route pour ajouter le user dans la base de données du 400



Le contrôleur : **addUser**()

Documentation sur idb-pconnector : <https://testdemogitbook.gitbook.io/restify/new-tutorial-2-rest-apis-and-the-ibm-i>

Dans ce contrôleur d’ajout user **addUser()** on va devoir :

* Contrôler la présence de données transmises par le req.body (ça doit être les champs de la table)
* Utiliser idb-pconnector avec prepare + bind + execute car il y a des paramètres



On va tester mais attention il y a une erreur

TEST\*

CALL QP2TERM

CD nodejs/demo3

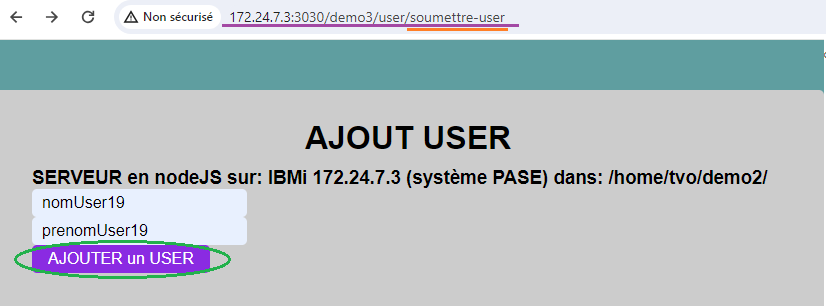
**NPM RUN** dev ou bien : **NODE** serveur.js

\*

TEST sur 3030: <http://172.24.7.3:3030/demo3/user/soumettre-user> => on peut utiliser le : **9080** pour être ‘visible’

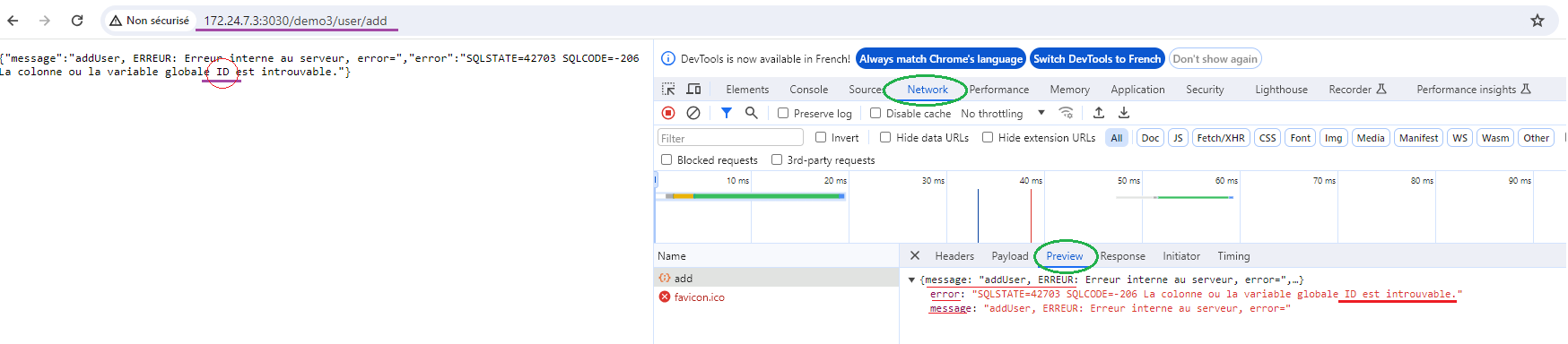
-

…

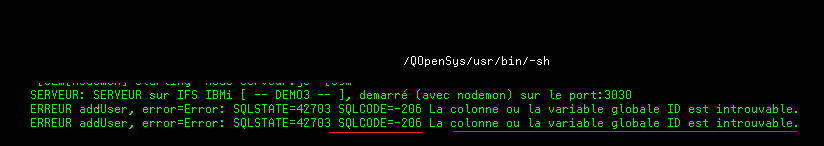


La méthode déclenche une erreur que je renvoie en console et sur la page HTML

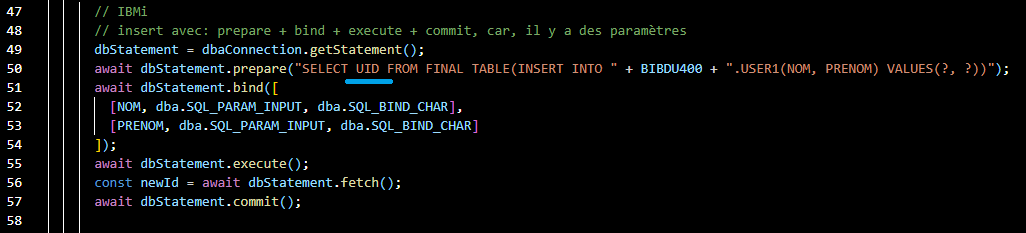
* PAGE :



* CONSOLE :



* Il faut corriger l’erreur, en remplaçant ID par UID



TEST\*

CALL QP2TERM

CD nodejs/demo3

**NPM RUN** dev ou bien : **NODE** serveur.js

\*

TEST sur 3030: <http://172.24.7.3:3030/demo3/user/soumettre-user> => on peut utiliser le : **9080** pour être ‘visible’

-

ERREUR

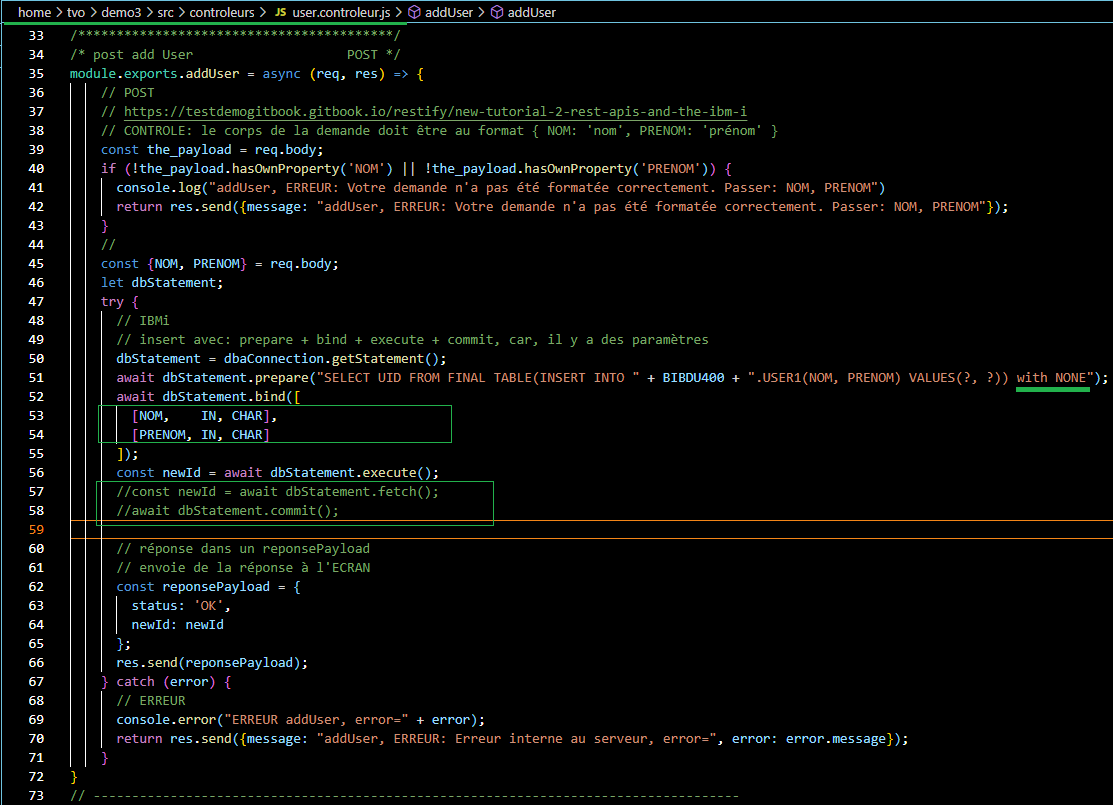
Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

CORRECTION :

* Désactiver le commit rollback : with NONE
* Déclarer les paramètres du bind IN & CHAR
* Ne pas gérer de commit si NON activé côté AS400



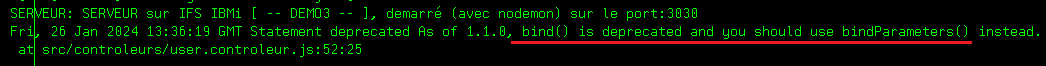


Ok : le insert fonctionne

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

!??: il reste une erreur non bloquante sur le bind des paramètres



## Fonction : soumettreUpdateUser()

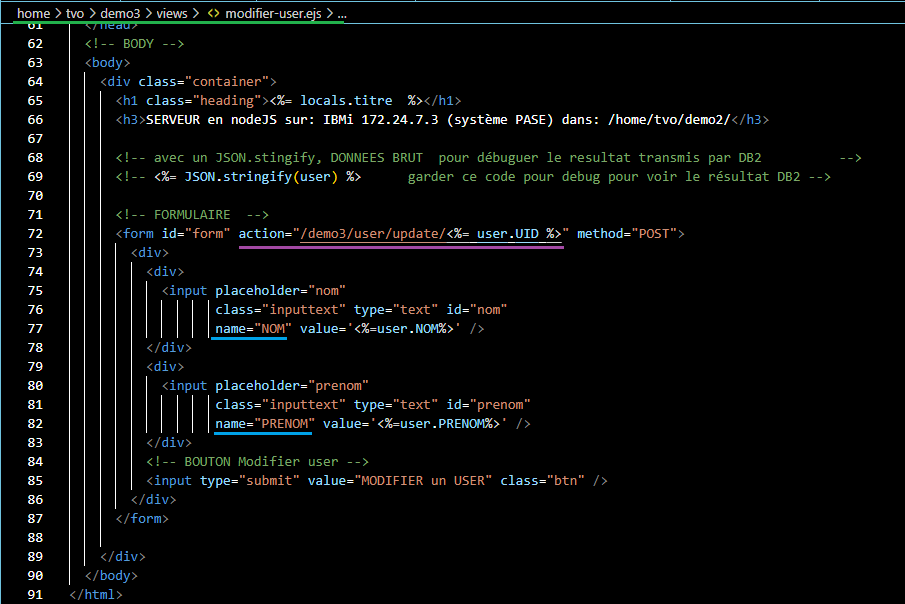
Il s’agit de la PAGE formulaire de Modification qui va poster la modification

On doit d’abord récupérer le user avec son id pour l’afficher dans la page de modification

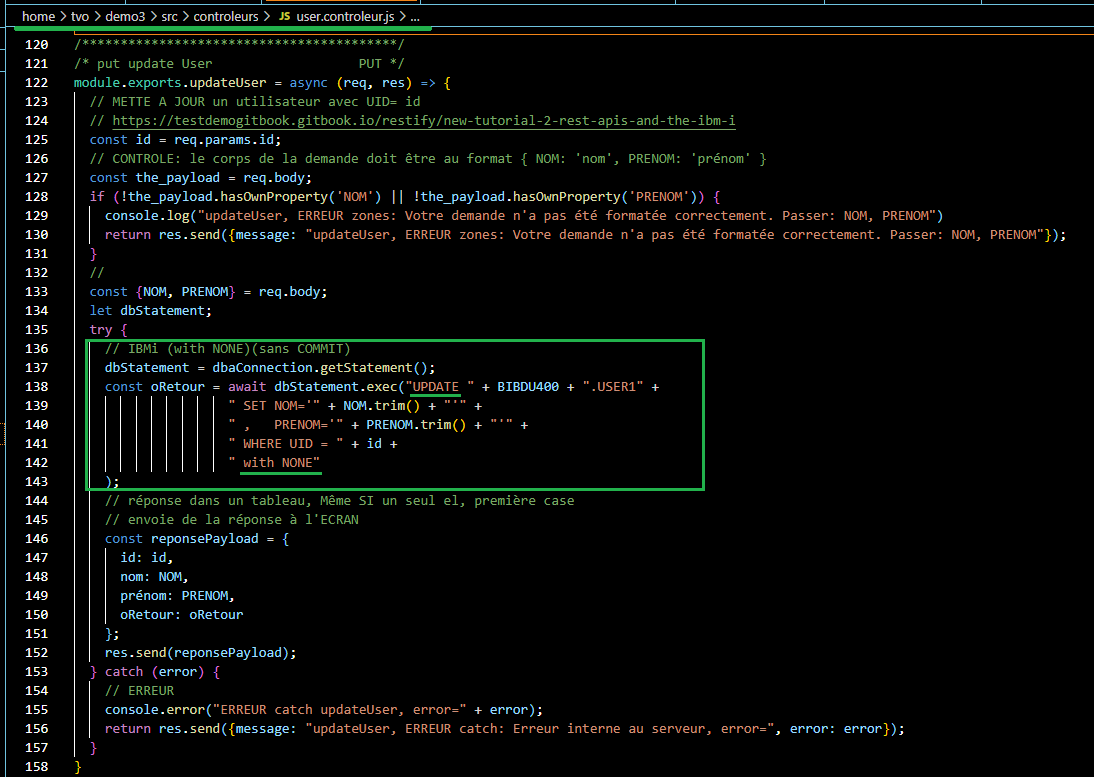
Le contrôleur



La page ejs



## Fonction : updateUser()

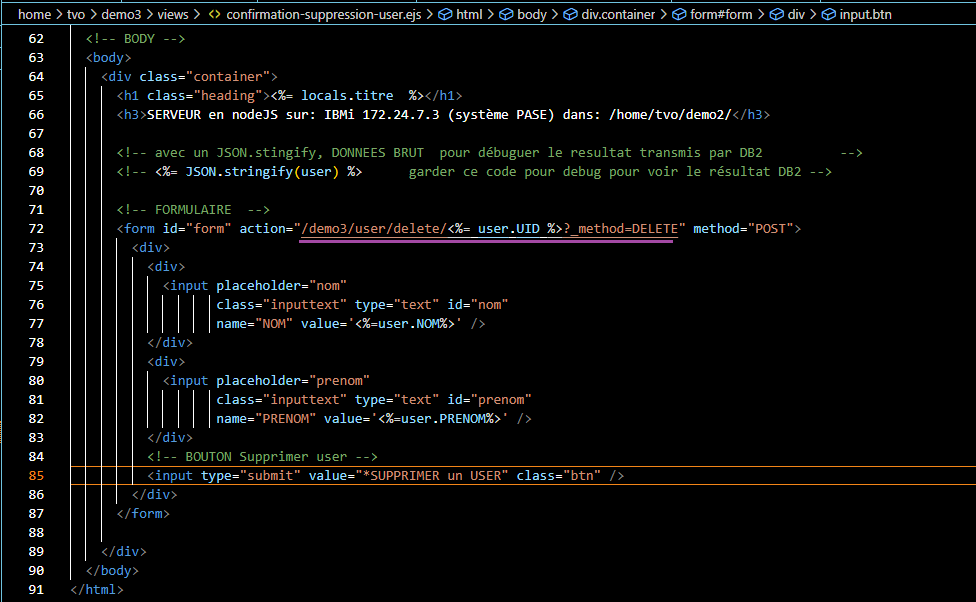


## Fonction : soumettreDeleteUser

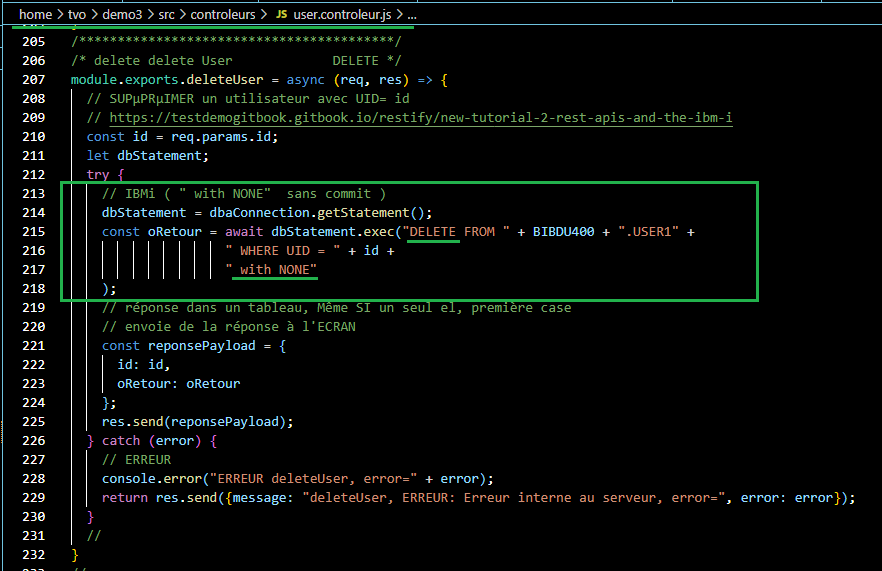
Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

La page



## Fonction : deleteUser



# Node JS sur IBM i – idb-**p**onnector – demo-user

## Rendre du HTML avec des donnes PF à l’aide de Node JS & idb-**p**connector sur IBMi AS400

<https://testdemogitbook.gitbook.io/restify/new-tutorial-2-rest-apis-and-the-ibm-i>

On utilise idb-**p**connector avec :

Description du fichier utilisé : USER1 dans TVO

Une image contenant texte, capture d’écran, horloge, Horloge numérique

Description générée automatiquement

PGM : SQLRPGLE de create table, TVO/QPRGLESRC(**UCRTUSER**)

SQL : STRSQL

insert into tvo/user1 + F4

INSERT INTO TVO/USER1(NOM, PRENOM) VALUES('VOZELLE', 'Thierry')

INSERT INTO TVO/USER1(NOM, PRENOM) VALUES(COESENS, 'Dounia')

## Initialisation du projet

CREER l’application demo-user

**CREER** le dossier sur l’ifs /home/tvo/demo-user

**CREER** le fichier application serveur (par copier-coller) /home/tvo/demo-user/app.js

**LANCER** le shell IBMi (AS400) QP2TERM : call QP2TERM

**INITIALISER** le projet dans le dossier /home/tvo/**demo-user/**

Changer de répertoire: **cd** **/demo-user**

**INITIALISER** le projet: **npm init** => cela va créer le package.json

**MOFIFIER** le fichier package.json pour activer le **nodemon**

****

**INSTALLER** express et tout le nécessaire sur l’ifs avec qshell (application ejs avec un **layout** dans **main.ejs**) :

mpm install express dotenv ejs

mpm install **idb-pconnector**

layout :

npm install express-session connect-flash express-fileupload cookie-parser

npm install express-ejs-layouts express-fileupload method-override

**INSTALLER** nodemon, dans qshell QP2TERM: npm i -D nodemon

**Une image contenant capture d’écran, texte, Police

Description générée automatiquement**

**COPIER COLLER** les dossiers nécessaires :

* Dossier : public : on garde tel quel
* Dossier : views (on va garder le même principe de base )
* Dossier : serveur (on va garder les mêmes principe )
  + Dossier : config (gérer la configuration dans : .env )
  + Dossier : models (peut être inutile ici)
  + Dossier : routes ( à mettre à jour )
  + Dossier : controleurs ( à mettre à jour )

**ARCHITECTURE** du projet NodeJS EJS views layout (résumé)

[app.js]------------[routes] ---------------- [controleurs] ---------------- [models] ---------------- [views]

Démarre le serveur CRUD Méthodes Class PAGES

Listenou sql/AS400

Et

Met en place

Le routage

## Fichier: **app.js** – le serveur nodeJS qui va écouter sur un port

On le garde tel quel en remplaçant client par user

* Route de base : ‘/demo-user’
* Le fichier des routes : userRoutes
* Le port : 3000
* Serveur nom: DEMO-USER

On commence par garder le fichier app.js tel quel, on change juste, pour que sa puisse compiler :

* Le nom des routes : user-routes
* Dans la config, vérifier : /serveur/config/.env (port & serveur nom)
* L’accès/démarrage de la base de données : à supprimer, l’Accès se fait par SQL

~~require("./serveur/models/database")~~

* Dans user-routes.js mettre user-contoleurs & userControleur pour chaque route du CRUD

Dans user-controleurs.js mettre chaque méthode du CRUD comme ci-dessous (méthode vide pour compiler) :

* Dans les routes on peut garder la structure du CRUD tel quel
  + soumettreAddFormulaire
  + addUser
  + homepage
  + getOne
  + soumettreUpdateFormulaire
  + updateOne
  + deleteOne
  + rechercheUsers
  + about()
* Dans le contrôleur, on créé chaque méthode à vide pour compiler
  + soumettreAddFormulaire ()
  + addUser()
  + homepage()
  + getOne()
  + soumettreUpdateFormulaire()
  + updateOne()
  + deleteOne()
  + rechercheUsers()
  + about()
  + Mettre à jour la class User – models/users.js
* Dans models/database.js Ne rien mettre on est en AS400 SQL AS400
* Dans models/user.js Ne rien mettre on est en AS400 SQL AS400
* La sa devrait compiler : prendre ensuite modèle sur demo3 & EJS02\_CRUD
* Prendre un copier coller de cette maquette.

**TESTER** : pour tester il faut obligatoirement passer par le qshell de IBMi c’est-à-dire : CALL QP2TERM

**DEMARRER** le serveur :

\*

CALL QP2TERM

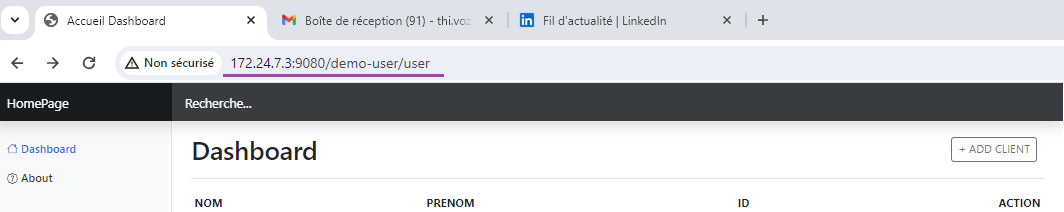
CD nodejs/demo3

**NPM RUN** dev ou bien : **NODE** serveur.js

\*

TEST sur 3030: <http://172.24.7.3:3030/>demo3/user => on peut utiliser le : **9080** pour être ‘visible’

Ça doit afficher la maquette vide



## REMARQUE : String vide en NodeJS

TEST en debug :

console.log('------- aaa date modification:');

const aaa = oTabUnUser[0].DATE\_MODIFICA;

console.log(aaa);

if(aaa === null){ console.log("c est null"); }

if(aaa === ""){ console.log("BLANC ce n'est pas VIDE"); }

if(aaa.trim() === ""){ console.log("c est VIDE"); }

if(aaa === undefined){ console.log("c est undefined"); }

console.log('taille:');

console.log(aaa.length);

console.log('------------------------------');

const derniereMAJour = (oTabUnUser[0].DATE\_MODIFICA.trim() === "") ? oTabUnUser[0].DATE\_CREATION : oTabUnUser[0].DATE\_MODIFICA;

x

ATTENTION : en nodeJS BLANC ce n’est pas vide ; il faut rajouter un .trim()

Donc 10 BLANCS n’est pas égale à ‘’

Il faut faire unString.trim() === ‘’

## REMARQUE 2 : retrouver la data retourné par idb-pconnector

TEST en debug :

try {

// IBMi

// compter

let dbStatement;

dbStatement = dbaConnection.getStatement();

const tabxxxx = await dbStatement.exec("SELECT COUNT(\*) AS **NOMBRE** FROM " + BIBDU400 + ".**USER1**");

console.log('-------------- tabxxxx');

console.log(tabxxxx);

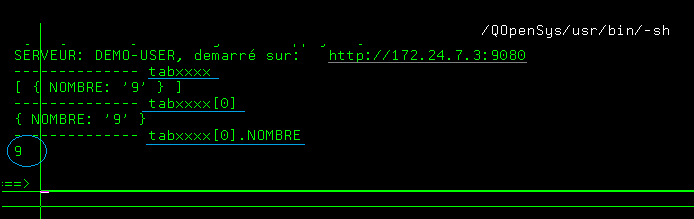
console.log('-------------- tabxxxx[0]');

console.log(tabxxxx[0]);

console.log('-------------- tabxxxx[0].**NOMBRE**');

console.log(tabxxxx[0].NOMBRE);

Visu du résultat :



X

x

## Pre