TP Chaîne d'acquisition – 3 – IR
NODE-JS et WEB-SOCKET

Objectif :
Un nano-ordinateur sous Raspberry OS (Linux) doit recevoir des informations issues d'un système externe (Arduino) via sa liaison série, traiter ses informations pour les afficher en HTML ou les enregistrer dans une base de données.

On vous fournit :

- Une platine Arduino avec un logiciel embarqué qui génère des trames GPS, ou qui est capable de recevoir des commandes/
- Un nano-ordinateur Raspberry et sa carte SD préchargée.
- Un câble de liaison USB

Pré-requis : Le TP Chaine d'acquisition 1.

Sommaire

1 NODE JS : UN MOTEUR JAVASCRIPT POUR SERVEUR : A LIRE !

Node-JS n'est pas un serveur Web !
C'est un « moteur d'exécution » du langage « JavaScript orienté serveur », beaucoup plus complet que le JavaScript réservé aux navigateurs.

Il est conçu pour recevoir des « node-modules » qui augmentent ses compétences. Certains modules sont installés de base (comme le module **http** grâce auquel il devient serveur Web).

Dans ce TP, nous allons utiliser les node-modules ***http***, ***serialport***, ***websocket*** pour que notre application Node-JS sache lire le port série, et dialoguer en Websocket (ws) avec un navigateur.

On dira donc provisoirement « au revoir » au serveur Web *Apache2* … Cela veut dire qu'on dit aussi provisoirement « au revoir » au PHP …

> Note : Pour faire du *WebSocket*, on n'est pas obligé d'utiliser un serveur *NodeJS*.
> Il existe d'autres solutions, par exemple en PHP (la plus connue est ***Ratchet***), ou les modules WebSocket de Java. *NodeJS* permet de rester dans un environnement full-JavaScript.

2 *WEBSOCKET* : LE MODE PUSH SUR VOTRE NAVIGATEUR. A LIRE !

Un TP précédent vous a fait tester une solution AJAX.
AJAX est une transmission par **socket TCP** avec un Navigateur, au travers du protocole **http**.

Un **socket** désigne un point de connexion entre 2 processus.

Particularité d'AJAX : c'est une communication « ***one shot*** » comme disent nos amis anglais :
Le Navigateur ouvre en arrière-plan une connexion à un service **http**, échange des données, et attend la fin de la connexion pour traiter les données reçues. Il faut ensuite recommencer pour obtenir de nouvelles informations. C'est donc le Navigateur qui est à l'origine de l'échange. AJAX ne prévoit pas le mode « ***push*** », c'est-à-dire quand le service distant prend l'initiative d'envoyer une donnée (comme une notification).

WebSocket est aussi une transmission socket TCP, mais il ajoute le mode « ***push*** » à votre Navigateur. C'est un protocole (http://ws:) qui s'appuie sur une partie du http.
Avantage : Le Navigateur ouvre un canal *WebSocket* avec un serveur. Ensuite, le serveur et le Navigateur peuvent échanger des informations à n'importe quel moment. Si c'est le serveur qui envoi, le Navigateur reçoit un évènement de notification. C'est le mode « push ».

> Note : La plupart des langages ont des bibliothèques pour faire de la communication socket. Ces bibliothèques sont plus ou moins compliquées à utiliser.
> Le protocole *WebSocket* est reconnu comme un moyen « simple » de faire de la communication socket. Il a donc été intégré à certains langages (Ex : Java).

3 INSTALLER UN MOTEUR NODE-JS SUR LE RPI

Un tutoriel très correct est à cette adresse :
https://thisdavej.com/beginners-guide-to-installing-node-js-on-a-raspberry-pi/#install-node

TRAVAIL :
On note que l'installation se fait en 2 temps :

1. Téléchargement d'un package hors magasin : ATTENTION : Installer la dernière version. Aidez vous du site https://thisdavej.com/upgrading-to-more-recent-versions-of-node-js-on-the-raspberry-pi/

   ```
   curl -sL https://deb.nodesource.com/setup_17.x | sudo -E bash -
   ```

2. Installation

   ```
   sudo apt install -y nodejs
   ```

3. Vérifiez les versions installées de NodeJS et des gestionnaires de paquets NPM et NPX :

   ```
   pi@pi3os:~ $ node --version
   v17.4.0
   pi@pi3os:~ $ npm --version
   8.3.1
   pi@pi3os:~ $ npx --version
   8.3.1
   pi@pi3os:~ $
   ```

4. Faites un petit test avec un JavaScript « de base » :
   Créer un fichier **minitest.js** contenant :

   ```
   var j = 10;

   while(j--) {
         let ligne = " ";
         let i = 10;
         while (i--) {
               let num = j*i;
               if (num < 10) num = "0" + num;
               ligne += " " + num;
         }
         console.log(ligne);
   }
   ```

   Et testez-le en tapant : **node** minitest.js
   Il devrait afficher une table de multiplication inversée.

5. Effectuez ce test qui vous permettra de valider le module **http** et l'accès par le réseau :

Fichier hello.js :

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

// Configure our HTTP server to respond with Hello World to all requests.
var server = http.createServer(function (request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.end("Hello World\n");
});

// Listen on port 8000, IP defaults to 127.0.0.1
server.listen(8000);

// Put a friendly message on the terminal
console.log("Server running at http://127.0.0.1:8000/");
```

Lancez le serveur : node hello.js
Et utilisez un navigateur pour accéder à votre Raspberry sur le port 8000 :
http://192.168.1.xx:8000

OBSERVATION : Ce code ne donne pas envie de faire du HTML … On regrette déjà notre bon gros Apache ! Il existe cependant des packages qui permettent de retrouver un environnement plus classique pour le html/css : Plus d'info sur le tuto : https://thisdavej.com/create-a-web-server-in-node-without-any-code/

4 AJOUTER LE NODE-MODULE WEBSOCKET

Il faut ajouter la fonctionnalité « Web Socket » à Node JS.

Question existentielle :
Actuellement, un package très utilisé est « socket.io » qui permet de « faire » toutes sortes de sockets. Il y a des aussi packages spécialisés « websocket » (websocket, express-ws) qui semblent plus faciles à utiliser. Des discussions intéressantes se trouvent ici :

- https://stackoverflow.com/questions/10112178/differences-between-socket-io-and-websockets
- https://qastack.fr/programming/10112178/differences-between-socket-io-and-websockets
- https://davidwalsh.name/websocket
- https://www.npmjs.com/package/websocket

➔ **Choix du prof pour ce TP : Package WebSocket.**
Pourquoi ? A cause du code JS côté navigateur. Les navigateurs récents possèdent tous la technologie WebSocket. Avec Socket.io, on doit ajouter au code un lien externe vers un serveur CDN qui fourni la technologie « socket.io » au navigateur. C'est gênant pour les applications embarquées. C'était pratique pendant la période floue où les navigateurs n'étaient pas tous à jour. Socket.io gérait bien ça.

1/ Créez un dossier (avec la commande « mkdir ») pour votre projet NodesJS
2/ Placez-vous dans ce dossier avec la commande « cd »
3/ Installez le module WebSocket à l'emplacement du projet :

```
npm install websocket
```

Vérifiez : il doit maintenant y avoir un dossier node_modules dans votre dossier de projet.

3/ Testez votre serveur : utilisez les fichiers fournis avec ce TP :
Pour simplifier le test, nous testerons la partie websocket sans la partie http.
serveurWS1.js : le serveur exécuté par NodeJS
clientWS.html : simple fichier HTML + JS que vous exécuterez en local sur votre machine Windows, après avoir indiqué l'adresse IP et le port d'écoute de votre serveur WS.

5 AJOUTER LE NODE-MODULE PORT SERIE

Ne oublions pas le port série …
Node-JS doit être capable de lire le /dev/tty ??? pour y récupérer les informations GPS.
Il faut donc ajouter le module Serial-Port. Un tuto en parle :

https://serialport.io/docs/guide-usage/

1) Installation :
   Dans le dossier où se trouvent vos programmes js :

```
npm install serialport
```

2) Tester :
   Créez un fichier **serial.js** et mettez le code suivant :

```
const SerialPort = require('serialport') ;
const Readline = require('@serialport/parser-readline') ;

const port = new SerialPort('/dev/ttyUSB0', {
     baudRate: 4800
}) ;

// On redirige le port série vers un "parser"
// pour détecter les fins de ligne (\n)
const parser = port.pipe(new Readline({ delimiter: '\n' })) ;

// On n'utilise pas la méthode "port.on"
// puisqu'on passe par le parser ainsi :
parser.on('data', reception) ;

// Fonction « callback » qui reçoit en argument
// le résultat du parser
function reception(ligne) {
     console.log("lu:", ligne);
}
```

Testez ce code : vous devriez recevoir les trames GPS.

```
$ node serveur.js
lu: ▯Simulateur GPS/MODEM
lu:  GPS : Entrée D2 à GND
lu:  MODEM : Commandes SN (Ex : SNLED1 pour alumer D13)
lu:
lu: $GPGGA,124037.289,4352.8207,N,00545.4549,E,1,04,3.2,200.2,M,,,,0000*0E8207
lu: $GPAAM,A,A,0.10,N,WPTNME*32
lu: $GPGGA,124038.289,4352.8373,N,00545.4958,E,1,04,3.2,200.2,M,,,,0000*0E8373
lu: $GPAAM,A,A,0.10,N,WPTNME*32
lu: $GPGGA,124039.289,4352.7930,N,00545.4772,E,1,04,3.2,200.2,M,,,,0000*0E7930
lu: $GPAAM,A,A,0.10,N,WPTNME*32
lu: $GPGGA,124040.289,4352.8144,N,00545.4778,E,1,04,3.2,200.2,M,,,,0000*0E8144
lu: $GPAAM,A,A,0.10,N,WPTNME*32
```

6 FINALISER LA LECTURE DU PORT SERIE

Votre travail est de terminer le programme **serveur.js** pour qu'il affiche la trame GPGGA complète, sous forme de message JSON :

Pour découper la trame, vous utiliserez la méthode ***split*** de la variable « ligne », qui est en fait un objet ***String*** (***chaine de caractère***) :

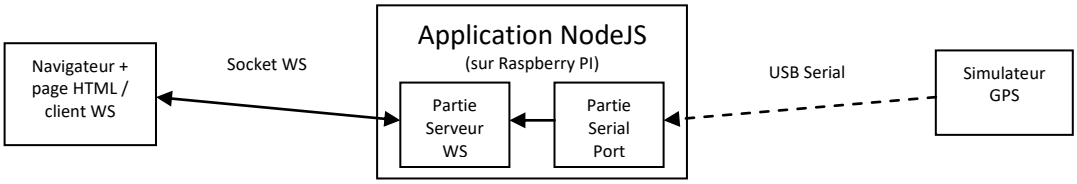
Ex : `var champs = ligne.split(',');`

« champs » est un tableau (array) contenant les mots isolés de la trame. Le découpage se fait par la virgule.

Pour transformer un *array* en Json, utiliser `JSON.stringify()`

7 LE GROS BOULOT : L'ASSEMBLAGE FINAL !!

Objectif : Envoyer cette trame JSON par WebSocket, à un Navigateur qui pourra ainsi mettre à jour son affichage en « temps reel », sans qu'il soit besoin d'un bouton, d'un rechargement de page, ou d'un setInterval … A VOUS DE JOUER !!!



8 LANCEMENT AU DEMARRAGE

Vous notez que votre application n'est pas un service « *daemon* » de Linux qui pourrait être activé par une commande **systemctl.**

Pour démarrer votre service NodeJS au boot du Raspberry, il vous faudra utiliser une des 2 techniques suivantes :

- Soit créer un nouveau service (Exemple complet basé sur un script php indépendant : https://medium.com/@benmorel/creating-a-linux-service-with-systemd-611b5c8b91d6)
- Soit ajouter le lancement de votre serveur dans le fichier /etc/rc.local

9 CONCLUSION : ON REGRETTE DEJA LE APACHE ET PHP ?

On constate que la programmation NodeJS n'est pas aussi limpide que le bon vieux couple Apache + PHP.
Et on n'a vu ici que le module websocket : on n'a pas développé un site web avec le module http !!!!
D'ailleurs, la plupart du temps pour un site Web, on associe des modules comme EXPRESS, ExpressGenerator, et des générateurs de template comme PUG ou JADE pour « faciliter » le travail de création du site.
Il existe des environnements de développement intégrés comme ANGULAR-JS ou REACT-JS qui savent s'interfacer avec NodeJS. Mais ça, c'est une autre histoire…