communication I²C avec Node.js et Raspberry Pi

affichez les données de vos capteurs dans un navigateur

Franck Bigrat (France)

Le bus I²C existe depuis les années 80 et est pris en charge par les plates-formes populaires actuelles, notamment le Raspberry Pi. Grâce à Node.js, il est possible de créer un serveur Web en JavaScript et de transmettre des données à un réseau local ou à Internet. Cet article vous explique comment connecter facilement un capteur I²C à un Raspberry Pi et afficher les résultats dans un navigateur Web.

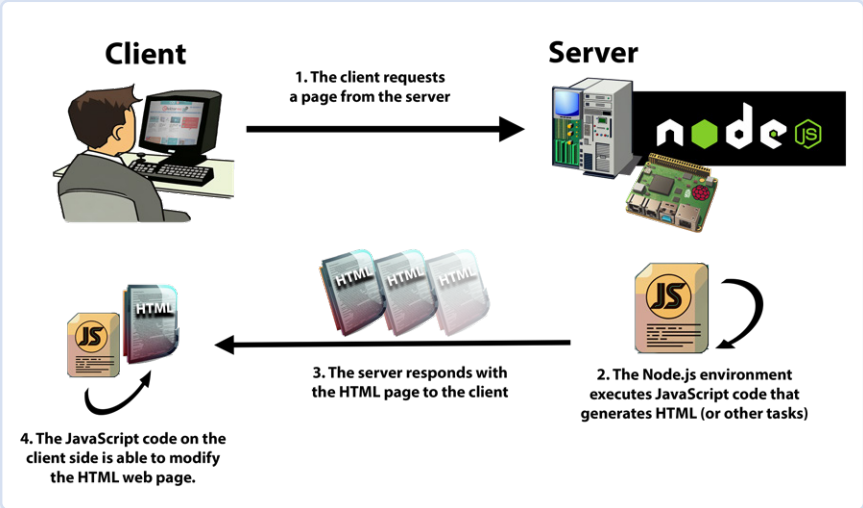

Figure 1. Flux de communication du serveur Node.js. (Source : sdz.tdct.org)

Dans ce projet, nous utilisons un simple Raspberry Pi Zero et le capteur de température I²C DS1621. Le résultat de la mesure est affiché sur une page web et est accessible sur une tablette ou un smartphone connecté au même réseau wifi.
Node.js dispose de modules tels que *http* et *express*, qui permettent au développeur de créer des serveurs web en toute simplicité. Nous n'avons donc pas besoin de recourir au serveur Apache qui est très répandu.
Bien qu'il existe un module dédié au contrôle du capteur DS1621, il est plus intéressant d'utiliser le module *i2c-bus* de Node.js, qui nous permet d'étendre le projet et d'y intégrer une grande variété d'appareils I²C.

Qu'est-ce que Node.js ?

Expliquer en quoi consiste Node.js en quelques mots n'est pas une tâche facile.

De nombreux livres et sites web sont consacrés à Node.js. Souvenez-vous simplement que Node.js est du *JavaScript exécuté par le serveur web, il n'est donc pas exécuté par un navigateur web lancé côté client* (cependant, le navigateur peut également exécuter du code JavaScript si nécessaire). Il est basé sur le moteur JavaScript de Google Chrome (V8). En termes simples, le développeur crée des applications JavaScript, mais celles-ci s'exécutent sur le serveur. La **figure 1** en donne un aperçu.

Pour apprendre (presque) tout sur Node.js, je vous recommande vivement de visiter les sites web [1] et [2].
Vous pouvez également vous référer à ces livres :

- *Node.js in Practice,* Alex Young et Marc Harter (Manning)
- *Beginning Node.js,* Basarat Ali Sayed (Apress)
- *The Node Beginner Book,* Manuel Kiessling
- *Instant Node.js Starter,* Pedro Teixeira (Packt Publishing)

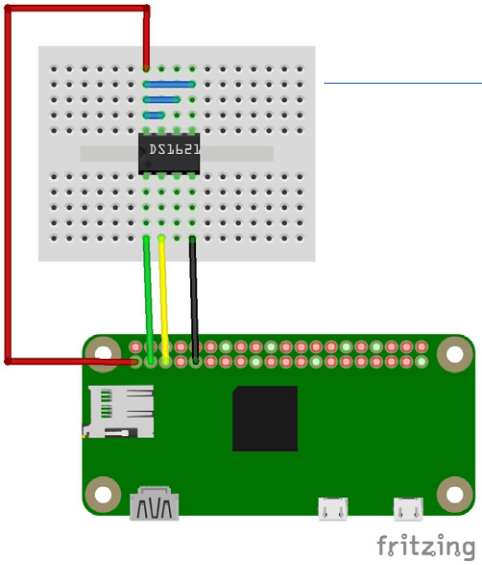

Figure 2. Schéma de connexion du DS1621 au Raspberry Pi Zero W.

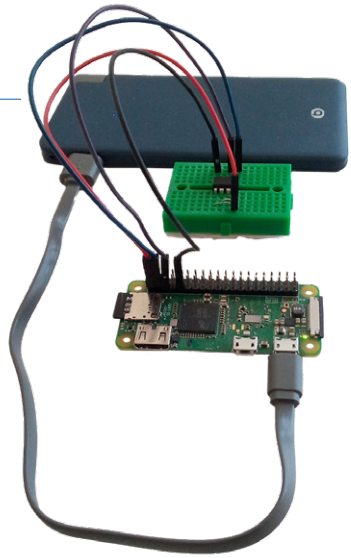

Figure 3. Le circuit est monté, avec la batterie.

I²C et SMBus

Dans ce paragraphe, je supposerai que vous êtes déjà un peu familier avec le protocole I²C (*Inter-Integrated Circuit*). Le plus important à savoir est que l'I²C a été développé par Philips dans les années 80 pour minimiser le nombre de fils entre un microprocesseur et un tas d'autres circuits intégrés. C'est un protocole de transmission de données synchrone, série, à deux fils (lignes de données et d'horloge). Il est possible de connecter plusieurs circuits, simultanément, aux mêmes lignes de données et d'horloge.

Les détails des fonctions du module *i2c-bus* de Node.js sont disponibles sur [3].

Si vous lisez attentivement, vous pouvez voir que certaines fonctions ont un préfixe *smbus* ou sont incluses dans la section SMBus. SMBus permet la communication entre le processeur d'un ordinateur et différents périphériques, notamment des appareils de gestion de l'énergie, des capteurs de température ou des ventilateurs.

Au niveau matériel, le Raspberry Pi prend en charge ce protocole : le bus de gestion du système est dérivé du bus I²C, et les deux protocoles sont tout à fait compatibles. Mais il existe quelques différences entre le SMBus et l'I²C ; elles sont expliquées dans le document de Texas Instruments [4].

Connexion et configuration des composants

Il est temps de connecter le capteur de température DS1621 au Raspberry Pi comme le montre la **figure 2**.

N'oubliez pas de relier les broches 5, 6 et 7 (A0, A1, A2) du DS1621 à V_{CC} pour définir l'adresse I²C du capteur à *4F* (hexadécimal). Le circuit réalisé est illustré dans la **figure 3**.

Configuration du Raspberry Pi

1. Téléchargez et installez *Raspberry Pi Imager* sur : *https://raspberrypi.com/software*

2. Lancez *Raspberry Pi Imager* (**figure 4a**), cliquez sur *CHOOSE OS* puis cliquez sur *Raspberry Pi OS (Other)* dans la liste qui s'affiche (**figure 4b**), puis *Raspberry Pi OS Lite* (32-bit) dans le sous-menu.

3. Cliquez sur *CHOOSE STORAGE*, puis sélectionnez votre carte microSD dans la boîte de dialogue *Storage* (**figure 4c**).

Figure 4a. Raspberry Pi Imager.

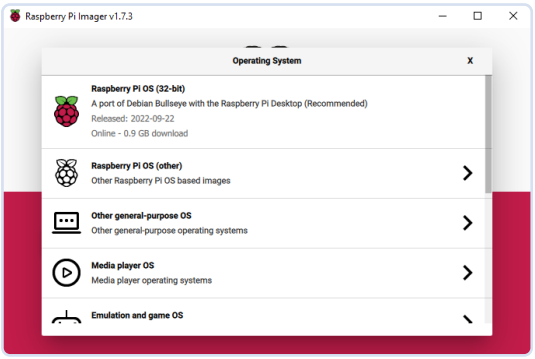

Figure 4b. Choix du système d'exploitation.

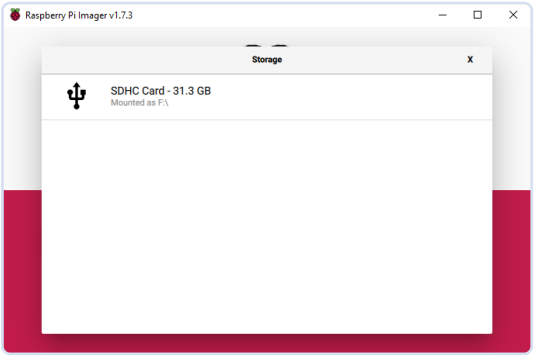

Figure 4c. Choix du support de stockage.

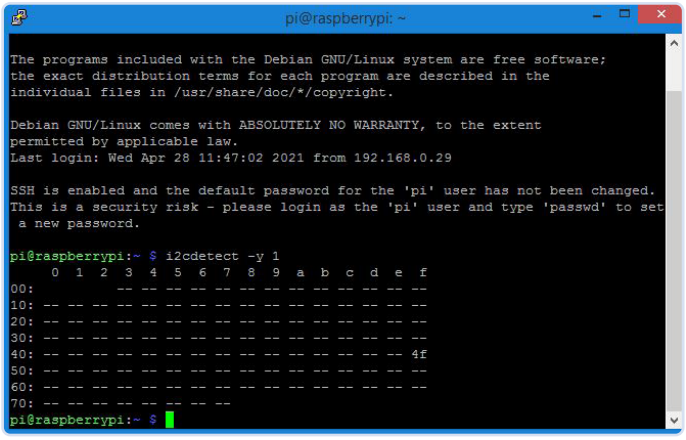

Figure 5. Sortie du programme i2cdetect.

4. Cliquez sur l'icône de paramètres (rouage) sur le côté droit dans l'application et définissez les options de configuration initiale du Raspberry Pi :
   - Cochez *Set hostname* : et entrez le nom d'hôte que vous voulez ; *raspberry* fera l'affaire.
   - Cochez la case *Enable SSH* et sélectionnez le bouton d'option *Use password authentication*.
   - Cochez la case *Set username and password*, puis saisissez le nom d'utilisateur (*pi* convient) et le mot de passe (choisissez un nom facile à retenir, par exemple *raspberry*).
   - Cochez la case *Configure wireless LAN* et saisissez le SSID et le mot de passe de votre réseau wifi.
   - Cochez la case *Set locale settings* et choisissez votre fuseau horaire (*Time zone*) et la disposition du clavier (*Keyboard layout*) si nécessaire, ou laissez les valeurs par défaut.
   - Cliquez sur le bouton *SAVE*.

5. Installez le système d'exploitation en cliquant sur *WRITE*, puis confirmez par *YES*.

6. Quand le processus d'écriture est terminé, retirez la carte microSD.

7. Insérez la carte dans la fente microSD du Raspberry Pi et mettez le Pi sous tension.

8. Trouvez l'adresse IP du Raspberry Pi sur le réseau avec un outil tel que Advanced IP Scanner ou, mieux encore, à partir des paramètres du bail DHCP de votre routeur wifi.

9. Utilisez un client SSH tel que Putty pour vous connecter au Raspberry Pi en utilisant son adresse IP et son nom d'utilisateur/mot de passe (*pi* et raspberry, respectivement, dans notre exemple).

10. Pour utiliser la totalité de l'espace de stockage de la carte microSD, étendez le système de fichiers :
    - Lancez l'outil *raspi-config* : `sudo raspi-config`
    - Choisissez *Advanced Options*, puis l'option *Expand Filesystem*.
    - Valider par la touche Entrée.
    - Passez à l'onglet <*Finish*> et confirmez par *Entrée*, puis confirmez le redémarrage en passant à l'onglet <*Yes*> et en appuyant sur Entrée.

11. Pour activer le bus I²C :
    - Ouvrez l'outil *raspi-config* : `sudo raspi-config`
    - Sélectionnez *Interfacing Options* et ensuite I²C *(Enable/disable automatic loading of I2C kernel module)*.
    - Confirmez par la touche *Entrée*.
    - Confirmez à nouveau en passant au bouton <*Yes*> et en rappuyant sur la touche Entrée.
    - Terminez par <*OK*> et ensuite <*Finish*>.

12. Connectez le Raspberry Pi au DS1621 selon notre schéma dans la **figure 2**.

13. Vérifiez la communication I²C :
    - Commencez par faire une mise à jour globale : `sudo apt update`
    - Installez les outils de diagnostic I²C : `sudo apt install i2c-tools` (vous devrez peut-être redémarrer le Raspberry Pi d'abord).
    - Exécutez la commande : `i2cdetect -y 1`
    - Dans la liste affichée, le capteur doit indiquer l'adresse que nous avons câblée, *4f* (**figure 5**).

14. Maintenant, installez Node.js et le gestionnaire de paquet NPM (*Node Package Manager*) : `sudo apt install -y nodejs npm` (cela peut prendre un peu de temps).

15. Vérifiez les versions avec `node -v` suivi de `npm -v`

16. Installez les modules *express*, *i2c-bus*, et *sleep* :
    `npm install express`
    `npm install i2c-bus`
    `npm install sleep` (peut générer des avertissements)

17. Créez un répertoire nommé *Documents* et un sous-répertoire nommé *NodeJS.*

18. Avec un client FTP tel que FileZilla, copiez les fichiers *DS1621_V3.js* et *DS1621_V4.html* dans le dossier *NodeJS*.

Et voilà, faisons un test !

Une fois que vous avez connecté tous les éléments, configuré le Raspberry Pi, installé Node.js et les différents modules, et copié les fichiers *.js* et *.html* sur le Raspberry Pi, ouvrez le répertoire *Documents/NodeJS* et exécutez le script avec `node DS1621_V3.js`.
Il suffit de connecter un PC, une tablette ou un smartphone à votre réseau wifi, d'ouvrir votre navigateur préféré et de saisir l'adresse : *[Your_Raspberry_Pi_IP_address]:8080*
Si vous avez fait les étapes précédentes correctement, le résultat devrait ressembler à la **figure 6**.

Figure 6. En attente de données.

Figure 7. Mesure et affichage de la température.

Après quelques secondes, la température sera affichée (**figure 7**). À partir de ce moment, la valeur de la température sera rafraîchie toutes les cinq secondes.

Explication des scripts Node.js et HTML

Avec Node.js, ce qui est important, mais difficile, c'est de comprendre l'exécution du script. En Node.js, l'exécution d'un script n'est pas séquentielle. Voir la **figure 8** : les instructions de la fonction `Server.get('/',...)` ne sont pas exécutées avant les instructions de la fonction `Server.get('/temp',...)`, qui ne sont pas exécutées avant les instructions suivantes.
Les instructions de la fonction `Server.get(«/»,...)` ne sont exécutées que si le serveur reçoit la requête *[Rpi_IP_Address]:8080* du client. Les instructions de la fonction `Server.get(«/temp»,...)` ne sont exécutées que si le serveur reçoit la requête *[Rpi_IP_Address]:8080/temp* du client. C'est ce qu'on appelle « la programmation événementielle ».
Dans les cinq premières lignes du script Node.js (partie gauche de la **figure 8**), nous importons les modules Node.js nécessaires et créons différents objets :

- Nous créons un objet `I2C` en utilisant le module `i2c-bus` :

```
const I2C = require('i2c-bus');
```

- Nous créons un objet `express` avec le module `express`. Ce module est un *framework* destiné à la création d'applications web avec Node.js :

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

- Nous créons un objet `fs` avec le module `fs` (*file system*). Cela nous permet de lire et d'écrire des fichiers sur la carte microSD :

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

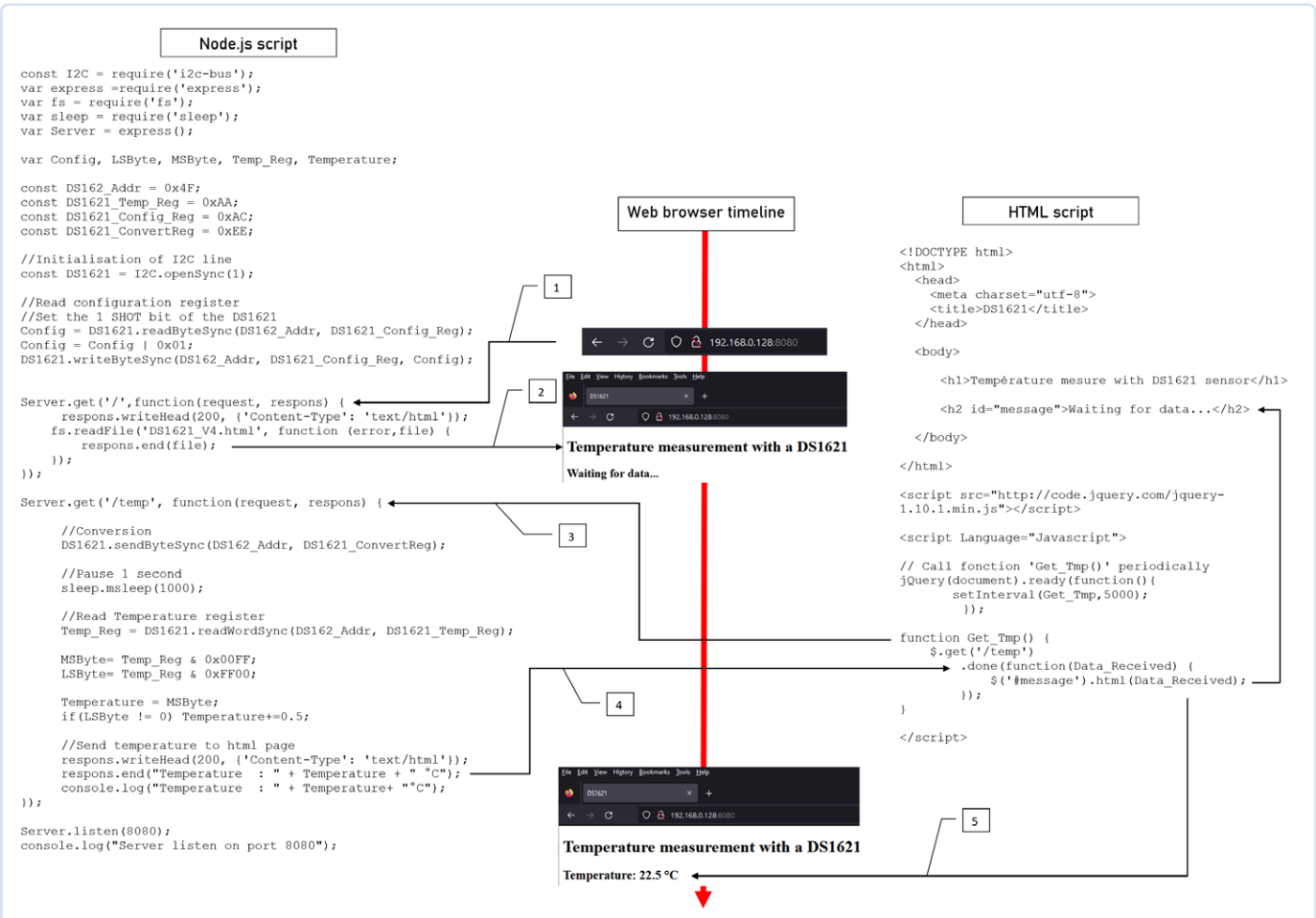

Figure 8. Ligne de temps des évènements.

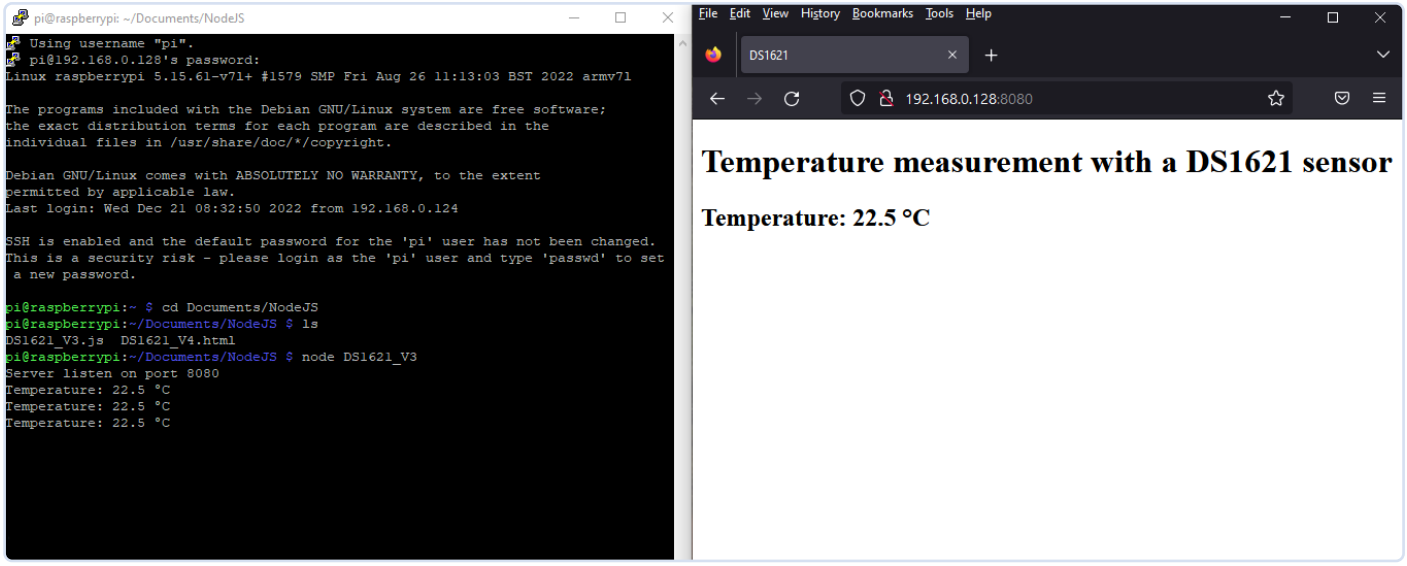

Figure 9. Terminal Linux et navigateur web.

› Nous créons un objet `sleep` avec le module `sleep`. Ce module nous permet de créer des délais :

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

› Enfin, nous créons un objet `Server` :

```
var Server = express();
```

Dans les cinq lignes suivantes, nous définissons 5 variables et 4 constantes. Ces constantes contiennent les adresses respectives des registres du DS1621.

```
var Config, LSByte, MSByte, Temp_Reg, Temperature;

const DS162_Addr = 0x4F;
const DS1621_Temp_Reg = 0xAA;
const DS1621_Config_Reg = 0xAC;
const DS1621_ConvertReg = 0xEE;
```

Ensuite, nous ouvrons le bus I²C et créons un objet appelé `DS1621`. Nous utiliserons cet objet pour communiquer avec le capteur via I²C.

```
const DS1621 = I2C.openSync(1);
```

Les trois lignes suivantes définissent le bit `1SHOT` du DS1621. Ainsi, la température est mesurée et convertie une fois (uniquement) lorsque le maître I²C (le Raspberry Pi) exige une conversion.

```
Config = DS1621.readByteSync(DS162_Addr,
         DS1621_Config_Reg);
Config = Config | 0x01;
DS1621.writeByteSync(DS162_Addr,
         DS1621_Config_Reg, Config);
```

Maintenant, observons ce qui se passe lorsque le navigateur envoie une requête au serveur (**figure 9**).
Lorsque le script Node.js est exécuté, le message « *Server listen on port 8080* » s'affiche dans le terminal Linux après quelques secondes. Ouvrez un navigateur web sur votre PC ou votre tablette connectée au réseau wifi. Dans la barre d'adresse, saisissez *http://* et l'adresse IP du Raspberry Pi et le port sur lequel le serveur écoute ; par exemple *http://192.168.0.128:8080*. Le port 8080 a été choisi pour être différent de celui utilisé par défaut par le serveur web Apache (Port 80).

1. Lorsque vous soumettez l'adresse IP, le navigateur envoie une requête *HTTP get* (obtenir HTTP) au serveur. Que se passe-t-il ?
   a. Le serveur détecte la requête avec `Server.get('/'...)` et exécute la fonction de rappel anonyme, `function(request, respons)`.
   b. L'objet `request` contient la requête envoyée par le navigateur.
   c. Dans l'objet `respons`, nous envoyons deux en-têtes comme paramètres :
      i. Un code http *200*, signifiant que la requête a abouti
      ii. La chaîne « *'Content-Type' : "text/html"* », signifie que le navigateur est prêt à recevoir des données texte de type HTML.
   d. L'appel de la fonction `respons.writeHead()` renvoie les données des paramètres au navigateur.

2. La fonction `fs.readFile()` lit le fichier DS1621_V4.html. La fonction de rappel anonyme, `function(error, file)`, est exécutée et, en cas de succès (nous le supposerons), le fichier est lu et placé dans l'objet *file*. L'appel de la fonction `respons.end(file)` envoie le code HTML au navigateur, qui affiche la page Web. En cas d'erreur, un code d'erreur est placé dans l'objet `error`.

La page Web est désormais au format HTML dans le navigateur. Que se passe-t-il ensuite ? Jetons un coup d'œil au script HTML (côté droit de la **figure 8**).
La première chose que nous voyons est la ligne `<h2 id="message">Waiting for data...</h2>`. Il s'agit d'un *placeholder* d'en-tête dans lequel la température sera écrite. Nous découvrirons comment plus tard.
Ensuite, nous voyons le bloc qui suit. Il s'agit d'un script écrit en JavaScript, mais il est du côté client, ce qui signifie qu'il sera exécuté par le navigateur :

```
<script src="http://code.jquery.com/jquery-1.10.1.min.
js"></script>

<script Language="Javascript">
```

```
  // Call function Get_Tmp() periodically:
  jQuery(document).ready(function() {
         setInterval(Get_Tmp, 5000);
          });

  function Get_Tmp() {
      $.get('/temp')
          .done(function(Data_Received) {
              $('#message').html(Data_Received);
          });
  }
</script>
```

D'abord, nous importons la bibliothèque *jQuery* du Web, qui nous offre des fonctions intéressantes. Grâce à jQuery, nous pouvons configurer un *timer* qui appellera la fonction `Get_Temp()` toutes les 5 secondes :

```
  jQuery(document).ready(function(){
       setInterval(Get_Tmp, 5000);
       });
```

Enfin, il y a la fonction `Get_Tmp()` :

```
function Get_Tmp() {
$.get('/temp')
     .done(function(Data_Received) {
          $('#message').html(Data_Received);
     });
}
```

Et c'est la partie intéressante du script !
Examinons comment elle interagit avec le script Node.js du côté serveur (voir la chronologie au centre de la **figure 8**) :

1. La fonction `Get_Tmp()` envoie une requête au serveur lorsqu'elle est appelée. Que se passe-t-il ensuite ?
   a. Le serveur détecte la demande avec `Server.get('/temp')` et exécute la fonction de rappel anonyme, `function(request, respons)`.
   b. Le maître I²C (Raspberry Pi) communique avec le DS1621 via le bus I²C, demande une conversion de données, les lit et les convertit en °C.
   c. L'objet de la requête contient la requête envoyée par le navigateur.
   d. Dans l'objet `respons`, nous stockons à nouveau nos deux paramètres :
      i. La valeur 200, est un code signifiant que la requête est réussie.
      ii. La chaîne « 'Content-Type' : "text/html" » signifie que le navigateur recevra du texte à interpréter comme du code HTML.
   e. La fonction `respons.writeHead()` renvoie ces données au navigateur.

2. L'appel de fonction `respons.end("Temperature : " + Temperature + " °C")` envoie le message « *Temperature : XX °C* » (qui est une chaîne ASCII) au navigateur. *XX* est la température mesurée par le DS1621.

3. Cette chaîne est placée dans l'objet `Data_Received`.

4. Vous vous rappelez la section d'en-tête `<h2 id="message">Waiting for data...</h2>` ? Nous voyons que cette balise d'en-tête comporte l'`id` du `message`, donc lorsque la chaîne est reçue, grâce à la ligne `$('#message').html(Data_Received) ;`, la chaîne envoyée par le serveur est placée dans la section d'en-tête et la page Web affiche la température.

Pour finir, expliquons un peu les deux dernières lignes du script Node.js :

```
Server.listen(8080);
console.log("Server listen on port 8080");
```

La ligne `Server.listen(8080) ;` lance le serveur, et la ligne `console.log("Server listen on port 8080") ;` écrit le message « *Server listen on port 8080* » dans la console Linux (shell).
Examinez bien la **figure 8**, je pense qu'une ligne de temps vous facilitera la compréhension.

210367-04 — VF : Asma Adhimi

Des questions, des commentaires ?

Contactez Elektor (redaction@elektor.fr).

LIENS

[1] Introduction to Node.js: A Beginner's Guide to Node.js and NPM: https://elektor.link/udaynode
[2] What is Node.js: A Comprehensive Guide: https://simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs
[3] Module i2c-bus : https://npmjs.com/package/i2c-bus
[4] Compatibilité du SMBus avec un appareil I²C : https://ti.com/lit/an/sloa132/sloa132.pdf