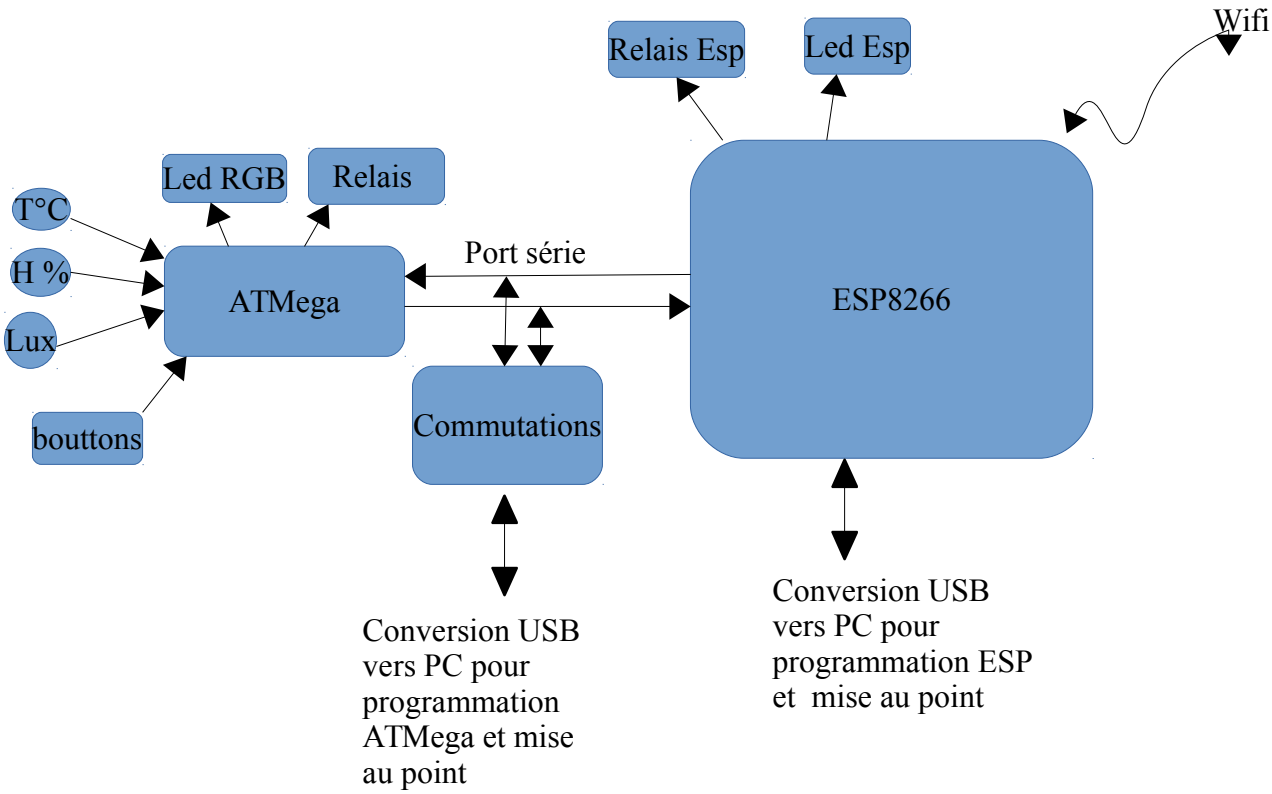
CNAM 2018
Objet Connecté

Ce tp a pour but de créer un objet connecté de démonstration offrant un serveur web permettant de visualiser les valeurs de différents capteurs (température,humidité,luminosité,…) et de commander des actions (par l'intermédiaire de relais).

Synoptique

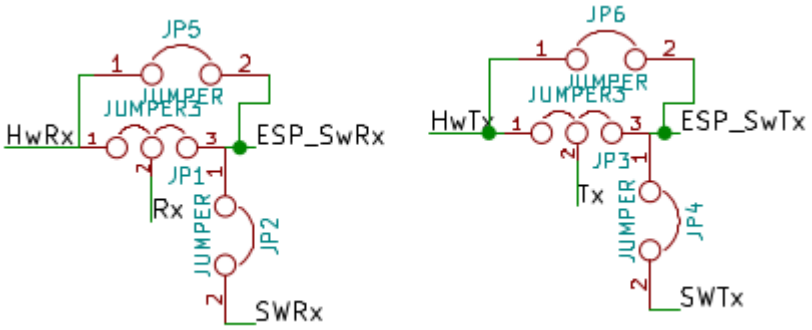


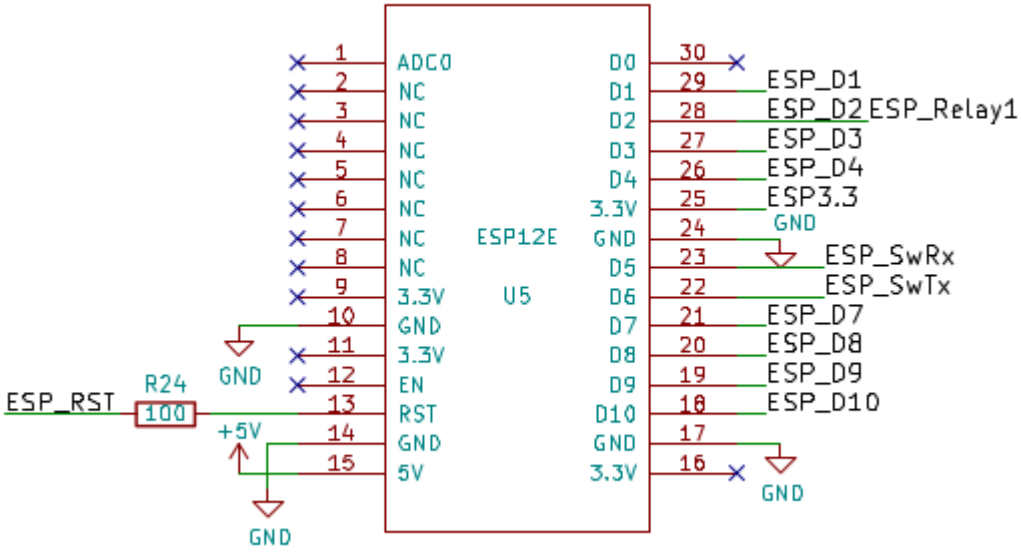
L'Atmega a été programmé dans le tp précédant pour relever périodiquement les valeurs des capteurs et pour répondre aux diverses commandes reçues sur son port série. Il est important de bien respecter le protocole décrit dans le tp précédent notamment les caractères de début '#' et de fin de trame '\r'.

Le principe général du programme de l'ESP8266 est le suivant :
Au démarrage, il va chercher à se connecter à un réseau wifi connu.
S'il n'en trouve pas il va se placer en mode "acces point".
Il va ensuite démarrer un serveur web (port 80) et un serveur websocket (port 81)
(une page sera disponible pour entrer les paramètres du réseau wifi à utiliser lors d'un prochain redémarrage)

Sur le serveur web une page permettra de visualiser l'état des capteurs et de commander les différents actionneur.
Pour cela il va communiquer par le port série connecté à l'ATMega pour lui passer des commandes et connaître l'état des capteurs.

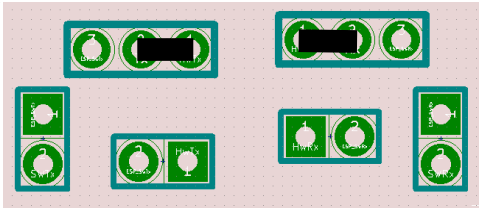
L'ESP8266 possède deux ports séries. Le premier (UART0) est utilisé pour programmer la carte de développement par le PC. Il sera aussi utilisé pour afficher des messages de debug.
La broche de réception du second (GPIO8 UART1) est utilisée pour commander la mémoire Flash, ce qui fait que sur notre carte l'UART1 ne peut que émettre des données.
Comme il faut communiquer dans les deux sens avec l'ATMega, nous allons émuler un port série sur une broche GPIO avec la librairie *espsoftwareserial* (https://github.com/plerup/espsoftwareserial)





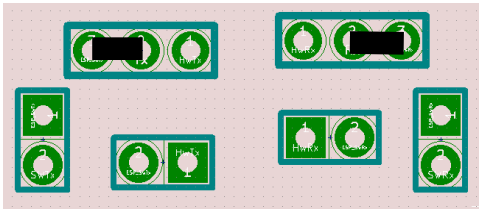
Sur notre carte des "jumpers" permettent différentes configuration pour la mise au point et pour le fonctionnement normal :

Programmation et mise au point ATMega : Liaison PC USB <-> ATMega
Jumpers : HTx <-> Tx et HRx <-> Rx

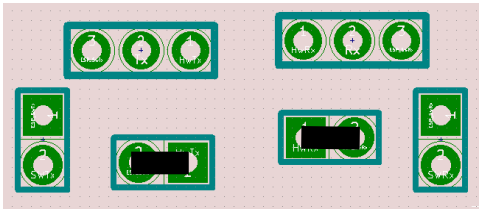


Mise au point ESP softwareSerial : On connecte le PC en USB sur le port série émulé de l'ESP. On peut ainsi visualiser avec un émulateur de terminal ce qu'envoie le port soft ESP et lui répondre au clavier comme le ferait l'ATMega.

ESP soft est configuré ainsi : ESP_SwRx=D5 et ESP_SwTx=D6 (voir schéma structurel de la carte)
Jumpers: ETx <-> Tx ERx <-> Rx



Fonctionnement normal : Liaison ATMega ↔ ESP8266 Software
Dans le programme de l'ESP, il faut inverser l'affectation des lignes ESP_SwRx et ESP_SwTx.



1. Mise au point de la liaison ATMega ↔ ESP8266

1.1/ Communiquer avec l'ATMega

Installer la librairie *espsoftwareserial* et adapter l'exemple Communication->SerialEvent.

Avec cette librairie, il n'y a pas de fonction automatiquement appelée, il faut créer une fonction `softSerialEvent()` sur le même modèle de ce que vous avez fait dans le tp précédent et l'appeler dans la boucle principale.

Pour tester votre programme connecter le PC en USB au port série soft de l'ESP.
Attention avec l'émulateur gtkTerm, le caractère de fin est obligatoirement '\r'.

L'ESP va envoyer des commandes puis attendre et interpréter la réponse de l'ATMega.
Pour la mise au point, il faut répondre dans l'émulateur de terminal comme le ferait l'ATMega.

Quand cette fonctionnalité est *parfaitement* au point, connecter l'ESP à l'ATMega (en inversant Tx et Rx dans le programme de l'ESP) et vérifier que vous obtenez des réponses correctes aux différentes commandes.

1.2/ Extraire les informations d'une commande !I

La commande d'information périodique envoie périodiquement la trame :

`#I;période;température;humidité;luminosité;tension;boutons;ok`

Exemple : `#I;30;25;45;123;12004;0;ok`

On veut extraire ces informations et les placer dans des variables globales qui seront utilisées par la suite : `int temperature,humidity,light,voltage,buttons;`

Pour cela on peut utiliser la fonction[1] ci dessous :

```
String getValue(String data, char separator, int index)
{
 int found = 0;
 int strIndex[] = {0, -1};
 int maxIndex = data.length()-1;
 for(int i=0; i<=maxIndex && found<=index; i++){
   if(data.charAt(i)==separator || i==maxIndex){
       found++;
       strIndex[0] = strIndex[1]+1;
       strIndex[1] = (i == maxIndex) ? i+1 : i;
   }
 }
 return found>index ? data.substring(strIndex[0], strIndex[1]) : "";
}
```

Analyser et commenter la fonction ci-dessus. Utilisez la pour mettre à jour les variables globales en fonction de la trame d'information reçue.

Programme "final" de cette partie :
Au démarrage, L'ESP envoie la commande de reset (!Z) puis la commande d'information périodique "!I10". Ensuite il met à jour les variables globales temperature, humidity,… à chaque réception d'une trame d'information.

2. Un premier serveur web

- Modifier l'exemple `ESP8266WebServer->HelloServer` pour mettre les paramètres du Wifi de la salle. Charger le et vérifier son fonctionnement.

Remarques : 1. Les paramètres du wifi sont écrits ici "en dur" dans le programme. C'est acceptable pour un tp d'initiation mais pas pour une application commerciale. Pour résoudre le problème de la connexion initiale, l'Esp démarre d'abord son propre réseau wifi (mode acces point) avec un portail captif. Une page permet alors de rentrer les paramètres de son réseau wifi. L'Esp sauve cette configuration et redémarre en se connectant au résau wifi configuré (mode STA). Ce problème ne sera pas traité dans ce tp. Une librairie très complète pour gérer ce type de problème est WiFiManager : //https://github.com/tzapu/WiFiManager

2. Il n'y aucune autre sécurité que la sécurité du réseau wifi. Toute personne connectée au réseau aura accès au serveur de l'Esp. C'est parfaitement acceptable pour une application

1Voir : https://arduino.stackexchange.com/questions/1013/how-do-i-split-an-incoming-string

domestique locale. Si on veut connecter directement l'Esp sur l'internet il est préférable de passer par un reverse-proxy.

- Modifier le nom DNS local pour qu'il soit de la forme **cnamXXYY.local** où XXYY sont les deux octets poids faibles de l'adresse MAC de votre ESP.
(utiliser la méthode macAddress() de la classe Wifi, attention le paramètre de la méthode begin() de la classe MDNS doit être une chaîne c (char *) et pas une String. Utiliser la méthode c_str() pour transformer une String en chaîne c).

-Modifier la page d'accueil pour afficher l'état de la led bleue (D1) et du relais relié à l'ESP (D2).

- Ajouter la possibilité de commander la led bleue et le relais relié à l'ESP par une requête de type GET de la forme : **Ip.De.Votre.Esp/execute?relay=0&led=**1
Si elle est correcte, le serveur exécute la commande et retourne une page simple indiquant l'action effectuée et redirige vers l'index au bout de quelques secondes.

3. Une page HTML qui affiche les informations reçues de l'ATMega

En vous inspirant de l'exemple `ESP8266WebServer->AdvancedWebServer`, ajouter une fonction qui génère une page HTML affichant les valeurs de température, humidité, luminosité et tension d'alimentation.

4. Une courbe avec les dernières valeurs mesurées

On souhaite afficher un graphique avec les n (n=10 par exemple) dernières valeurs de luminosité mesurées.
Pour cela il faut d'abord stocker les n dernières valeurs mesurées. Une structure de données adapté est un "buffer circulaire" pour implémenter un comportement de type FIFO.

- Installer et étudier la libraire CircularBuffer : https://github.com/rlogiacco/
Comment déclarer un buffer pour stocker les données de luminosité ? , comment écrire une donnée ? Comment lire les données de la plus ancienne à la plus récente ?

-Stocker les n dernières valeurs de luminosité dans une fifo. Vérifier le bon fonctionnement en affichant le buffer à chaque insertion.

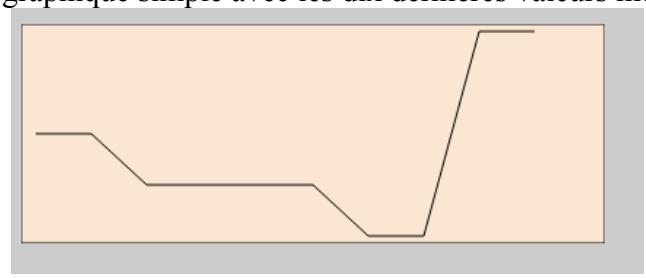
Il existe plusieurs manières pour générer un graphique dans un système embarqué. Certaines méthodes très puissantes impliquent une connexion internet (google chart,…), on trouve également des bibliothèques java script open source (chart.js,….). La génération du graphique est alors reportée côté client. L'objet connecté ne faisant que transmettre les données.
Nous allons utiliser une méthode très simple qui ne nécessite pas de connexion extérieure ni une grande puissance de calcul pour générer une image côté serveur (dans l'objet).
Il s'agit simplement de créer une image de graphique souhaité au format ***svg***.

- Étudier la fonction drawGraph() de l'exemple :
`ESP8266WebServer->AdvancedWebServer`

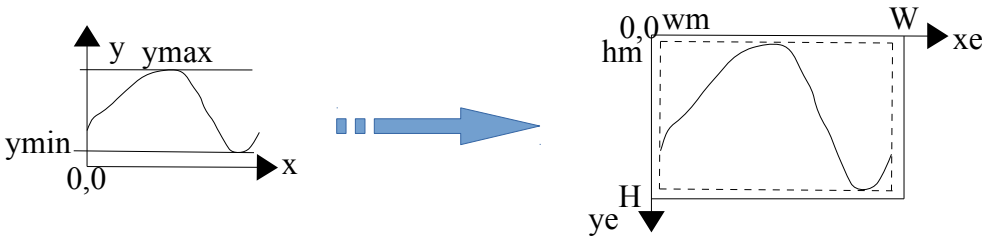
En vous aidant d'une documentation du format svg, expliquer la construction de l'image de l'exemple.

On souhaite un graphique autoadaptatif montrant les n dernières valeurs mesurées.

Par exemple voici un graphique simple avec les dix dernières valeurs mesurées :



Il faut trouver les relations entre les données d'origine (x,y) et les coordonnées (xe,xe) sur l'image de taille (W,H) avec une marge (wm,hm).



Pour les y, il y a une mise à l'échelle et un retournement, pour les x il y a seulement une mise à l'échelle.
- Exprimer xe en fonction de x,W,wm et du nombre n de points du graphique.
(x représente ici le numéro du point entre 0 et n-1).

- Exprimer ye en fonction de y, ymin, ymax, H et hm.

- Quel sera le problème si les valeurs sont toutes identiques ? Dans ce cas on prendra un écart fixe de ± 10 %.

- Écrire la fonction qui produit le graphique avec en paramètre la source de donnée, le nombre de point, la taille de l'image et les marges.

5. Une page (statique) en Flash

La page précédente était stockée dans la RAM de l'ESP. Il est ainsi facile d'en modifier le contenu mais on va vite être à court de RAM avec ce principe. D'autre part la mise au point est difficile car le code html doit être intégré directement dans le code.
L'ESP partage sa mémoire Flash en deux : une partie pour le code et une autre partie pour un système de fichier appelé SPIFFS. Généralement avec 4Mo de flash, on choisit soit la répartition 1M code /3M SPIFFS soit l'inverse.
Procédure de chargement en SPIFFS :
Même si vous utilisez un autre IDE pour développer votre projet, le plus simple est d'utiliser l'IDE arduino pour charger des fichiers. Il faut d'abord installer un commande additionnel à l'IDE arduino :https://github.com/esp8266/arduino-esp8266fs-plugin

Créer un sketch arduino quelconque et créer un répertoire **data** dans le répertoire de ce sketch. Les fichiers présent dans ce répertoire seront chargés dans la mémoire Flash de l'ESP par la commande "Sketch data upload" de l'IDE arduino.

- Étudier attentivement l'exemple "A-SPIFFS_File_server" du "ESP8266-beginer-guide" (https://github.com/tttapa/ESP8266) et le document "ESP8266 arduino Server Web SPIFFS et WebSocket" partie 1.

- Ajouter la possibilité d'accéder à un fichier index.html statique à votre programme.

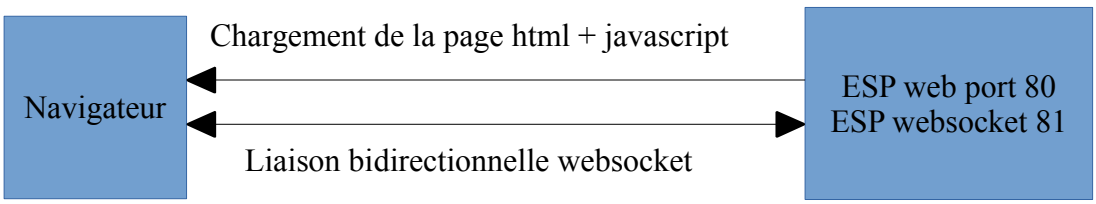
6. Un serveur webSocket

Jusqu'à présent les pages étaient soit "statiques" stockées en Flash (code ou SPIFFS), soit générées en RAM sur demande par le serveur. La seule manière de rafraîchir la page est de la recharger en totalité. C'est acceptable pour une petite page mais ce n'est pas très fluide ni très réactif. Nous allons mettre en place un serveur *webSocket* de manière à pouvoir demander des information et passer des commandes sans avoir à recharger la page entière. Un programme javascript embarqué dans la page demandera périodiquement des mises à jour et pourra modifier uniquement les éléments nécessaires de la page. De plus les différentes modifications pourront être répercutées automatiquement aux différents clients connectés.

6.1 Étude de la librairie arduinoWebSockets

- Installer la librairie https://github.com/Links2004/arduinoWebSockets
- Charger et étudier l'exemple : https://gist.github.com/bbx10/667e3d4f5f2c0831d00b
- Modifier le pour contrôler la LED bleue de l'ESP.
- Vérifier que si vous chargez la page dans un ou plusieurs autres navigateurs (éventuellement sur des machines différentes), l'état de la led est automatiquement mis à jour.

Principe de fonctionnement :



Le navigateur obtient la page qui contient le html et un code javascript par une requête sur le port 80. Une fonction javascript s'exécute (sur le navigateur) au chargement de la page pour établir une connexion avec le serveur websocket sur le port 81. Le navigateur peut alors communiquer avec le serveur websocket par des fonctions javascript et modifier éventuellement le contenu de la page. Le serveur websocket peut envoyer des informations à tous les clients connectés pour effectuer les mises à jour de l'état des capteurs/actionneurs.
On peut ainsi avoir un affichage fluide et très réactif.

Un mot sur la sécurité : Ici la sécurité du système repose sur l'accès au réseau wifi. Dès qu'on a accès au réseau wifi, on a accès aux différentes pages de l'ESP et au serveur websocket. Le serveur websocket ne vérifie pas l'origine de la connexion. Il est donc illusoire de mettre en place un accès par mot de passe à la page web. Ce fonctionnement est tout à fait acceptable pour une utilisation domotique privée dans lequel tout les membres d'une famille ont accès aux différents objets connectés dès lors qu'ils ont accès au réseau wifi. Il ne faut surtout pas

que l'ESP soit directement visible depuis l'internet (par une redirection de port sur la box par exemple) car dans ce cas n'importe qui peut y avoir accès. Si on veut y accéder depuis l'extérieur de son réseau on peut utiliser le support SSL de l'ESP mais celui ci est assez limité. La solution la plus utilisée semble être de mettre en place un reverse proxy.

6.2 Mise au point du serveur webSocket

Pour mettre au point le serveur websocket, nous allons utiliser l'utilitaire ***`wscat`*** qui permet de se connecter en ligne de commande à un serveur. On pourra ainsi envoyer des données au serveur comme le fera plus tard notre programme javascript et vérifier le résultat.

- Installer l'outil ***`wscat`*** (https://www.npmjs.com/package/wscat)
- Vérifier que vous pouvez vous connecter par :

```
$wscat -c ws://ip_du_serveur:port
```

-Contrôler la led de l'exemple en ligne de commande

Comme nous l'avons fait pour le port série, nous allons mettre en place un protocole très simple pour communiquer avec le serveur. Nous allons avoir deux types de commandes :
- des commandes qui sont destinées à être exécutés directement par l'ESP
- des commandes qui sont destinées à être exécutes par l'ATMega.

Pour ces dernières on va simplement les préfixer par le caractère **'+'**. Elles sont communiquées à l'ATMega et le serveur renverra le résultat. Les autres seront préfixées par **'='**. Elles seront interprétées et exécutées par l'ESP qui répond en commençant par **'&'**.

Commandes "ATMega" :

Par exemple sur réception de "+!P1" le serveur transmet "!P1" à l'ATMega, attend la réponse (max 200ms) et retourne la réponse de l'ATMega ou "#?" si timeout.

Commandes "ESP" ("&?" si inconnue):

Fonction	**Commande**	**Réponse**	**Commentaire**
Lire upTime	**=U**	**&U;01:54:34;ok**	Temps écoulé depuis le reset
Relais ESP	**=R1**	**&R;1;ok**	Change état du relais ESP
Led ESP	**=L1**	**&L;1;ok**	Change état de la LED

- Ajouter un serveur websocket sur le port 81 au programme de la question 4.

- Implémenter les différentes commandes.
  Les réponses sont envoyées à tous les clients connectés, pour qu'ils puissent mettre à jour l'état du système (sauf pour les commandes upTime et Info).

6.3 Un page en spiffs qui communique avec le serveur websocket

En vous inspirant de l'exemple, écrire une page html et un code javascript qui informe de l'état de capteur et qui permet de commander les différents actionneurs.
Cette page sera stockée en flash dans le SPIFFS.

Nb : pour la mise au point, il n'est pas nécessaire de charger la page en flash. Lors de la création de l'objet WebSocket en javascript, il suffit de remplacer l'adresse de connexion par l'adresse de l'ESP pour que la communication s'établisse même si la page se trouve sur votre machine de développement :
remplacer :

```
websock = new WebSocket('ws://' + window.location.hostname + ':81/');
```

par :

```
websock = new WebSocket('ws://' + '192.168.0.44' + ':81/');
```