GUIDE TO SETTING UP WEB APPLICATION FOR
PROJECT: ECOHOME (not)™

‘

Course: TELECOMMUNICATION PROGRAMMING PROJECTS WITH ARDUINO

Course number: 34338

DTU, the Technical University of Denmark

Autumn semester, January 3rd - 22nd, 2022.

Web Application & WebApp Documentation written by August Valentin

Content

Introduction:

This document will take you from start to finish of setting up the server on your local network, viewing the webpage, and letting the Arduino talk with your local server. I tried to make the guide as clear and thorough as possible but might have ended up making the document too lengthy.

With regards to the 'local' part of a 'local server', it is possible to make the server public. This means that your PC can be connected to your home wifi, while microcontrollers could be connected to any other wireless network and still communicate with your server.

There exists some free (unpaid) resources to 'expose' your local server, but for most of them, you will have to change the IP every time you restart the service, and the connection will be slow. Examples of this kind of service is ngrok.com/ and github.com/localtunnel/localtunnel.

What You Need:

(Part 1) To run the server and webpage, you need:

- PC running either Windows, MacOS or Linux.
- A terminal capable of running Git commands (for Windows, this could be Git Bash)
- Installation of Python 3.0 or newer.
- A browser capable of running JavaScript (Google Chrome, Edge, Mozilla, etc.)
- Access to ECOHOME Github (github.com/mar-le-ne/eco-home)
- Access & connection to Wireless network (name & password).
- Text editor, like Windows Notepad, Notepad++, etc.

(Part 2) To connect the server with a MicroController, you will also need:

- An ESP8266 Chip connected to the MicroController (For example the NodeMCU 0.9)
- A USB connection to the microcontroller.
- Installation of the Arduino IDE.
- An Arduino Sketch for communicating with the server (or a sketch you wish to connect with the server)

Summary:

The main steps we will go through in part 1 are:

1. Setting up the repository.
2. Find your local IP.
3. Set up the server to use your local IP.
4. Running the server and viewing the webpage.
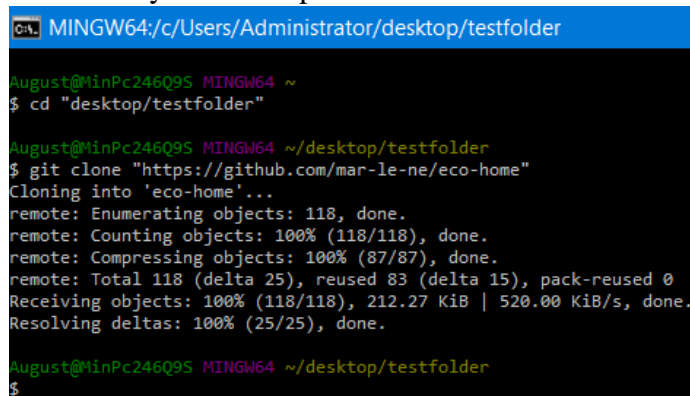
In part 2, the steps are:

5. Find the library file for communicating with the server.
6. Calling the right functions.

And that's it!

Part 1: Running the server

Step 1: download the server implementation from our github.

    a. If you do not already have the repository cloned on your local pc, this can be accomplished by opening the terminal of your Operating System. For Windows however, we need to use Git Bash or an alternative.

    b. Choose a place for the repository to be located on your pc. Let's call this <repository_path>

    c. Change the current working directory of the terminal, by using the following command:
cd <repository_path>

    d. After changing the directory, use: *git clone "https://github.com/mar-le-ne/eco-home"*
The terminal should look something like this, if your folder was called "testfolder" and located on your desktop:

    e. Now, a new folder will have been created in the directory, with the name "eco-home". This folder contains the main branch. Inside you will see some different folders, the relevant one being the "Server" folder. This folder contains the 2 things:
A python script for running the server called "server.py", and a folder with the resources for the webpage, called "webpage_lib".

    a. If you already have cloned the repository to your PC, simply make sure that you are on the "main" branch. If you have changes on your current branch that you want to integrate with the server, consider merging *from main* and *into your branch*. (For example, if you are working on an Arduino script that needs to communicate with the server).

## Step 2: Finding the IP of your Wireless Network

It's important to note that both your computer and ESP8266 must be connected to the same wireless network for them to communicate. **Make sure this can be done before continuing.**

I will go through the steps for finding the IP of your connection to the wireless network in Windows 10. If you are using MacOS or Linux, use these guides instead:

Mac: https://www.avg.com/en/signal/find-ip-address#find-your-local-on-mac (also has a guide for Windows).

Linux: https://opensource.com/article/18/5/how-find-ip-address-linux

   a. Open the Settings app for windows (Danish: "indstillinger").
   b. Choose the "Network & Internet" (Danish: "Netværk og internet").
   c. On the left-hand side, click on the "Wi-Fi" tab.
   d. Click on the name of the Wi-Fi you are currently connected to.
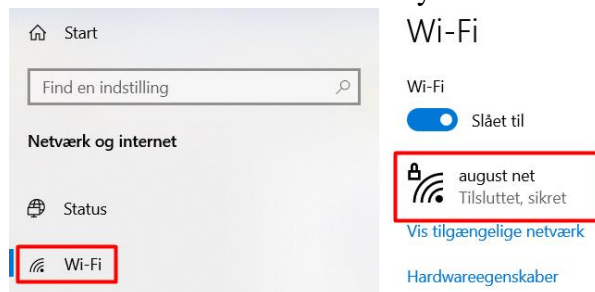
   e. Scroll down the page until you see a title "Properties" (Danish: "egenskaber").
      The IP of your computer in the local network should be listed to the right of "IPv4 address".
      The IP will be used a couple of times, so you should write it down.
   f. Verifying that this is a local IP address:
      A local IP is one of three types:
      Class A: the IP is within the ranges of **10.0.0.0 — 10.255.255.255**
      Class B: the IP is within the ranges of **172.16.0.0 — 172.31.255.255**
      Class C: the IP is within the ranges of **192.168.0.0 — 192.168.255.255**
      Source: www.avast.com/c-ip-address-public-vs-private
      If the IP is inside any of these ranges, it is a local IP. (I am not sure if it's even possible for the IP listed to not be local…)

Step 3: Modifying the files to match your local IP

To make the webpage work, a single file must be modified.

   a. Go to the main directory. Then navigate to "Server/webpage_lib/". Find the file called "ip.txt". If it doesn't exist, simply create it yourself.
   b. In the file, write the local IP you found before. Do *not* include quotation marks.
   c. That's it. The "Server.py" and "updatePage.js" files will read from this text file automatically.

Step 4: Running the server and viewing the webpage

The server can now be run from either a Python IDE, or a terminal. There are many different IDEs,

but I will go through the simple steps of executing it in PyCharm, and then also running it using the Windows terminal.

   a. Returning to the Server folder, open up the "Server.py" file. Alternatively, Start PyCharm up and click "file -> open", then navigate to the "Server.py" file and choose it.
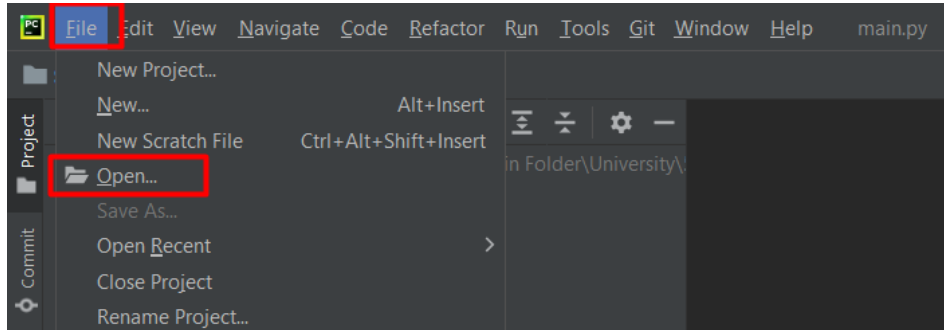
   b. Now, in PyCharm either click "Run -> Run 'Server' ", or scroll down until you see the line "if __name__ == '__main__':", then click the green arrow next to it, and then "Run 'Server' ".

       Third option is the keyboard shortcut, Shift + F10.
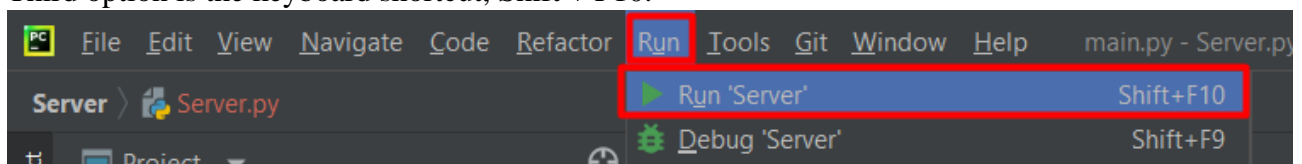
   c. PyCharm's terminal should give a message "SERVER BEGINS".
   d. Running the server using the Windows terminal is simple:
       Start the terminal (cmd.exe).
       change working directory to the server folder (cd "desktop/testfolder/eco-home/Server").
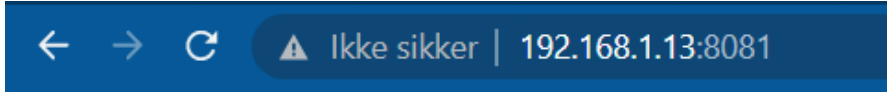   Run the python script using python3 (python3 Server.py).

e. Now, it's time to view the webpage. Open your browser-of-choice (I used Google Chrome), and in the search bar, type your local IP followed by ":8081". For example, if your IP is "192.168.1.13", your search bar should look like this:

If everything was done correctly, you should be greeted by the EcoHome webpage :D
You can check that the webpage is communicating with the server if you return to your IDE and look at the terminal. Several messages should be printing approximately every 15 seconds, representing the requests coming in.
Clicking the button with the "I've left the house" text will send a POST request to the server, which can also be seen by a message in the IDE's terminal.

You can also see the debugging messages of the webpage by right clicking the page, then selecting "Inspect element" (Danish: "Undersøg"). A white pane jumps up to the right. Click on the "Console" panel, and some messages similar to "Fetching data" and "Setting Data" should appear continuously.

f. Congratulations! You have now set up the server and webpage application for the EcoHome project. If your phone is connected to the same network as the server, you can access it the same way you did with your PC, using a browser.

(Optional): if you know how to, you can also try sending a POST or GET request using cURL. It probably won't be successful since you need to follow our established protocols, but the request should still appear in the server terminal.

Part 2: Extending ESP8266 programs to communicate with the Server.

This part is for "creating" the client that talks with our server. More than creating, it is focused on extending previously written Arduino scripts so that they interact with the server by sending sensor values, and possibly retrieving data from other distant sensors also connected to the server.

Step 5: Locating the "client_lib.h" file

a. Switch to the main branch of the repository (if it wasn't already your active branch). If you have an Arduino script on your current branch that you want to extend with the library, consider merging *from main* and *into your branch*.

b. Return to the main directory of the repository (where the "Server" folder is located)

c. Find and open the folder "lib".

d. Inside, a file "client_lib.h" is located. The ".h" extension means the file is a "header" file.

e. To extend the capabilities of the Arduino sketch, *copy* (do **not** move it) the "client_lib.h" into the folder your sketch ".ino" file is located.

f. Open the sketch with the Arduino IDE. Notice that a tab has opened alongside the sketch called "client_lib.h"
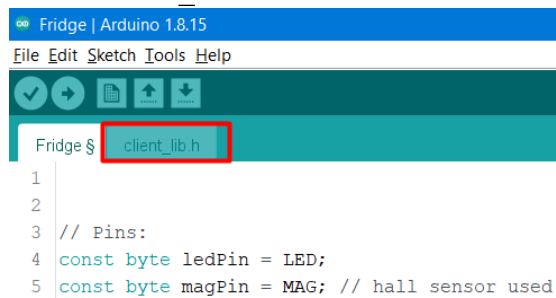
g. If you click on the tab, you can see the code inside the file.

h. Some extra information on header files:
Header files are commonly used in C and C++ to define the signatures of functions, defining values and some other auxiliary duties, typically for the sake of the compiler and the IDE. However, in Arduino, the header files are used to move any code from the ".ino" sketch that serves as the main access point of your program. Arduino can only use header files located in either your global Arduino library folder, or located in the same folder as the ".ino" sketch.
Header files from the global Arduino library are referenced by enclosing the name in inverted chevron and chevron (< and >). Header files from the same folder as the sketch are referenced by being enclosed in quotation marks (" ").
For example, loading the WiFiClient.h file from your global Arduino library:

```
4  #include <WiFiClient.h>
```

But loading a header file called "pins.h" from the same folder looks like this:

```
2  #include "pins.h"
```

It is due to Arduino's limitations of where header files can be stored, that we must copy the library file into each sketch-folder that requires it.
Official information on header files from Arduino.cc:
docs.arduino.cc/hacking/software/LibraryTutorial

Step 6: Using the library:

a. Now we have done **all** the tedious work. It is time to use the library in our sketch.

b. To interact with the server, our device must connect to the same network.

The steps for letting microcontroller connecting to the WiFi are:

the client library has a function called setupWIFI() (notice capitalization). The function needs three Strings in the order:

WiFi name, WiFi password and IP.

Make sure the WIFI password is not used elsewhere, such as for facebook, netflix, other personal accounts etc.!

The function setupWIFI() should be called *after* you have set the baudrate for the Serial device. This is because setupWIFI() has some printed messages.

Also, It's important that the microcontroller connects to the same network as your server, or they won't be able to communicate together.

Example seen below:

```
13 void setup() {
14   // Set baud-rate of Serial connection(?)
15   Serial.begin(115200);
16
17   String wName = "farligt wifi";
18   String wPass = "august1234";
19   String IP = "192.168.152.233";
20   setupWIFI(wName, wPass, IP);
21 }
```

c. Now, your microcontroller can connect to the WiFi. But it also needs to send something to the server, right?

A couple functions have already been defined for this. Their name and purpose is listed below.

name	Return type	Parameters	Description. True = first statement, False = second statement.
GEThome()	Bool	None	Ask the server whether user is home or not.
POSTfridge()	Void	Bool	Tell the server the fridge is open or closed.
POSTlight()	Void	Bool	Tell the server the light is on or not.
GETwaitTime()	int	Void	Ask the server for the wait time (to wait before turning off the light).
POSTforgotLight()	Void	None	Tell the server that the user forgot to turn off the light.
POSTautolight()	Void	None	Tell the server that the light was automatically turned off.
POSTfaucet()	Void	Bool	Tell the server the faucet is running or not.
POSTshower()	Void	Bool	Tell the server the shower is running or not.

d. An example of the library being utilized can be seen in the "eco-home/ArduinoSketches/Fridge/Fridge.ino" file.

e. Of course, more functions can be written, and other functions from the client library can be used. But keep in mind that these wrapper functions have very clear purposes and are therefore easier to use. Secondly, keep in mind that the lib_client.h file is simply copied. If you extend the functionality of the header file, it becomes difficult to keep track between different versions of the file.

If you want to send or retrieve some new information from the server, then you'll also have

to modify the Server.py script, updatePage.js and possibly the webpage.html, to reflect these changes.

Because of these reasons, **please refrain from modifying any of the files in the "Server" folder** without discussing it with the Product Owner. This part of course excludes the "ip.txt" file, which you have to modify.