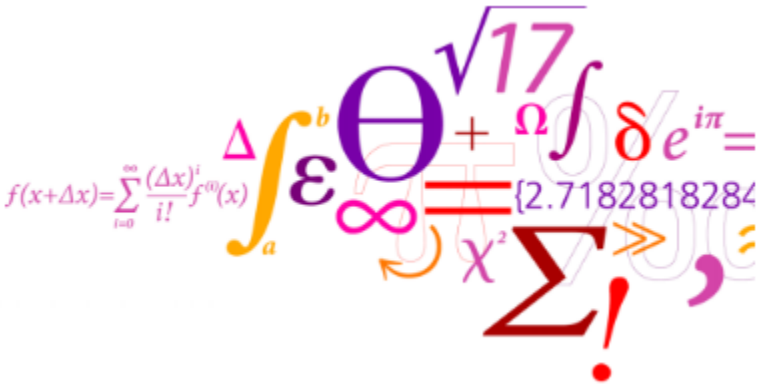
Technical University of Denmark

Implementation of an Internet-of-Things node using the Adafruit Feather Huzzah32 – ESP32 WiFi board and MicroPython

Simon Tobias Lund - s184459
Sara Maria Guijarro Heeb - s184484
Thomas Fayad - s184440

December 6, 2019



Technical University of Denmark
Course: 02135 - Introduction to Cyber Systems

Introduction

In this report is explained how a simple server running on the Adafruit Feather Huzzah32 - ESP32 WiFi board was build to be able to provide the following services :

- Output an `HTML` page describing the status of the server, when addressed by a standard browser.
- Provide a Web API to be used by a control program running on a laptop.

As a matter of fact some definitions have to be firstly defined.

`HTTP` stands for `Hypertext Transfer Protocol`, where hypertext documents include static elements (e.g. text, images and sounds), dynamic elements and embedded links (*hyperlinks*). It is a stateless protocol, meaning that each command is executed independently. These commands correspond to storing and retrieving hypertext documents. Most commonly, `HTTP` is used to transfer data over the web. Indeed, it applies a server-client model. In other terms, it consists of a sequence of exchanges between the servers and clients. Here, the servers are the providers of a resource or service, and the clients are services that request a certain action or method to be performed on a given resource (such as transferring a file).

`HTML` stands for `Hypertext Markup Language`. As in the previous definition, hypertext corresponds to static elements, dynamic elements and embedded links (*hyperlinks*) that an `HTML` page may contain. It is the standard language used to create web pages. Indeed, it defines the structure of a web content.

`JSON` stands for `JavaScript Object Notation`. JavaScript is a high-level, object oriented programming language. Being a high-level language means that it has more built-in functions so that it becomes easier to use (but sometimes also slower to compute). Moreover, object-oriented programming is organized around data or objects (eg. attribute-value pairs, array data types), rather than functions and logic. Finally, `JSON` is used primarily to transmit data between a server and web application, in a human-readable format.

1 First task

For this first task, an explanation of the initial provided code was redacted. This translation of the code lines was principally supported by the usage of the following website : `https://docs.python.org/3/library/socket.html`.

This exercise permitted a better understanding of the basics to which the rest of the tasks were based on.

2 Second task

For the second part, the program reads the value of the 0 pin, corresponding to the button. If the button value is 0, then the code will run the following:

The program gets the temperature from the temperature sensor via Wifi connection If the measured temperature is under 26 degrees Celsius, the LEDs will turn green. If the temperature is not bellow 26 but bellow 27, the LEDs will turn yellow or if none of those criteria are matched(when the temperature is larger than 27). Also, the program will create a table with an `HTML` source code that will display the temperature value and the button value.

This task is documented by two short videos, in the file "Tasks_videos.zip".

3 Third task

`API` stands for `Application Programming Interface`. A Web API facilitates interaction between two machines over a network. It is basically used as a framework that enables development of HTTP services to reach out to clients. These clients can be browsers, devices or tablets.

The `API` are the possible requests that can change values in the server or return a resource form the server. In this case, our server has a support for the three following `API` requests : `GET`, `SET` and nothing.

- If the request is empty, the `HTML` document will be returned. The empty path specifies that the requested resource is the `HTML` document. Other paths will give a value from the dictionary called `"API"`.
- A `GET` request returns a specific resource in the server, which must be specified by a path.
- The `SET` request changes a value in the `API`-dictionary.

The `JSON` scheme presents a response from the server has the format : `"HTTP/1.1 200 OK\nContent-Type:"`, succeeded by the content type, and then `"\ n\ n"` with the message body following. Where content type is `text/html` if the response is a `HTML` document, and `JSON` if the response is a value. Moreover, the python values were formatted to `JSON` with `json.dumps()`.

For this task, the server was adapted by sorting its information into a dictionary. Furthermore, code was added for parsing requests, and functionality too to respond with `JSON` if the request was `GET`.

4 Fourth task

First, the program creates a text file called `"data.txt"`. Next, it sends requests to the board via Wifi connection in order to read the temperature sensor.

If the status response is the value 200, that means no errors have been encountered, the program will write in the recently created text file the time and the temperature data received from the board. If the status response is different from 200, the program will print in the terminal: `"Data not obtained, server response"` with the

response code. After receiving the status code, the program will hold for 0.5 second and reenter the loop until it is manually terminated.

With the collected data, the following plot can be made

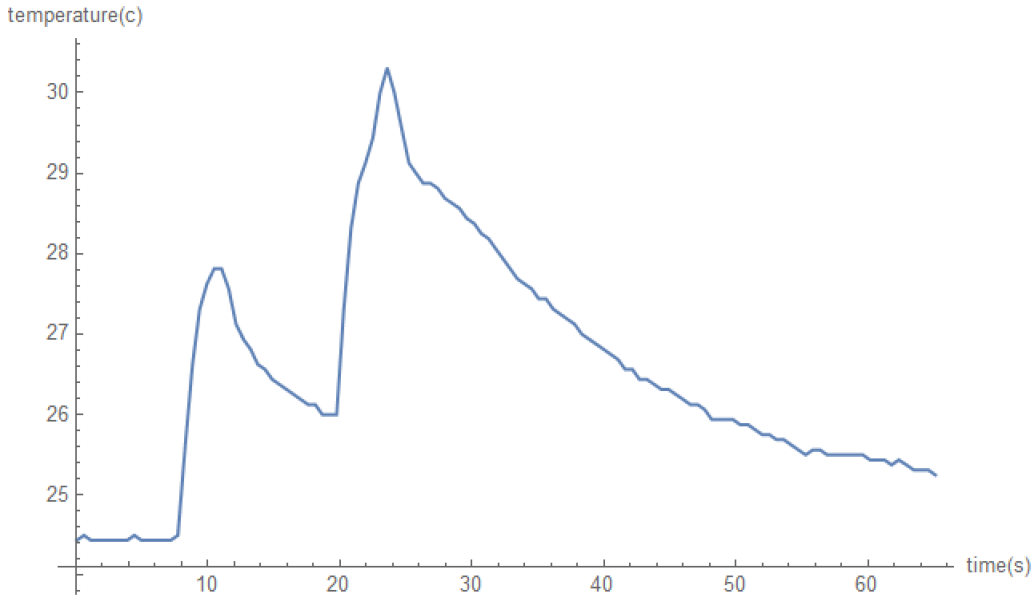


Figure 1: Plot representing the Temperature against the Time

Similarly as the second task, the functionality of the code is documented by a short video, in the file "Tasks_videos.zip".

5 Fifth task

For this final exercise, extensions were made to the initial API. Indeed, the `API`-dictionary was split into `input` and `output` to process `SET` and `GET` requests. Moreover, the parsing of requests was optimized, as to read message body from `SET` requests. Lastly, we made the LED value be dependant on the `API`, instead of the temperature.

The control program takes user input from the console, sends an appropriate request to the server, and displays responses to the console.

The user command can be either "`col`" (for changing led color) or "`get`" (for getting a resource). To have a better understanding, examples of valid commands are "`col red`", "`get temp`" and "`get button`".

Conclusion

To conclude this project permitted us to develop a program from very bareback communication, to a comprehensive communication protocol. In task 2 we had a server capable of recognizing the existence of a client request, and sending a `HTML` page in return. By task 5 we had a fully working server capable of parsing `HTTP` requests, and using the information to return or edit internal memory.