02135 Assignment 4

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

Hans Henrik Løvengreen and Luca Pezzarossa [`lpez@dtu.dk`]

November 8, 2021

1 Introduction

This document describes the fourth assignment of the course 02135 - Introduction to Cyber Systems. The purpose of this assignment is to let you apply network notions and protocols to control the hardware setup you have worked with in Assignment 3 (Micropython and I/O Lab) using the Adafruit Feather Huzzah32 – ESP32 WiFi board in order to get an idea on the way Internet-of-Things devices are working. In this description, the term *board* will refer to the Adafruit Feather Huzzah32 – ESP32 WiFi board that is part of the kit you have been provided with.

In this assignment, you are going to build a very simple server running on the board. The server should provide two services:

- It must act as a usual *Web server* when addressed by a standard browser resulting in an HTML page describing its status.
- It should provide a *Web API* to be used by a *control program* running on a laptop. The *Web API* allows the control program to interact with the board (read the status and control the board I/Os).

Similarly to assignment 1 – 3, this assignment should be carried out in **groups of two or three people**. The groups can be different to the ones of Assignment 1 – 3 and you need to re-register your group. In addition to the developed programs, you are required to prepare a short report. Further details are provided in Section 4. All the material related to this assignment can be found on the DTU-Learn course page at the following location:

`DTU-Learn/Course Content/Content/Assignments/Assignment 4`

The assignment is divided into **five tasks**. You are expected to complete these tasks and deliver the implemented programs and the report before the deadline. The deadline for this assignment is **Friday 3rd December 2021 at 23:59**. By this date, you have to hand-in an electronic version of the short report in

PDF format and the source files as ZIP archive using the assignment utility in the DTU-Learn course page at the following location:

`DTU-Learn/Course Content/Assignments/Assignment 4`

As usual, you also need to give a DEMO of the working tasks to the teacher or to the TAs.

This document is divided into 4 sections:

- Section 2 describes how to set up a WiFi-based Web server and lists the related tasks to be carried out (1 – 2).
- Section 3 provides the background on Web API and lists the related tasks to be carried out (3 – 5).
- Section 4 lists the assignment tasks and the report requirements.

As we are not enforcing any particular approach on the implementation, it may be difficult for the teachers to understand your implementation and thus, help to debug your code by taking into account the following hints:

- Structure your code such that it is clear what the different parts are doing.
- Think about testing your code while planning and make sure that you test and debug the individual parts.
- Document your code with comments and, if necessary, a README file with instructions on how to run it.
- Include pictures and screenshots proving the correct functionality of the system.

Feel free to ask or send an e-mail to [`lpez@dtu.dk`] if you have questions regarding practical matters or the assignment in general.

2 Web server

For communication, the board should be set up to provide its own private network in the form of an *access point* to which your laptop may connect using WiFi. Note that you cannot use the internet while your laptop is connected to the board (unless you have other network interfaces like a cabled one).

The following program, sets up the board as WiFi access point and starts a simple Web server (HTML based) that reports the status of the board pins. Before running the program, change the WiFi SSID (i.e. the WiFi network name) `ESP32-WIFI-NAME` and the WiFi password `WiFi-password`. In addition, remember to change the pin list (line 10) with the ones you want to be monitored by the program. If you want to see the HTTP request received by the server, you can uncomment the *print* command (line 35).

```
1  import machine
2  import network
3  import socket
4
5  ap = network.WLAN (network.AP_IF)
```

```
ap.active (True)
ap.config (essid = 'ESP32-WIFI-NAME')
ap.config (authmode = 3, password = 'WiFi-password')

pins = [machine.Pin(i, machine.Pin.IN) for i in (0, 2, 4, 5, 12, 13, 14, 15)]

html = """<!DOCTYPE html>
<html>
    <head> <title>ESP32 Pins</title> </head>
    <body> <h1>ESP32 Pins</h1>
        <table border="1"> <tr><th>Pin</th><th>Value</th></tr> %s </table>
    </body>
</html>
"""

addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]

s = socket.socket()
s.bind(addr)
s.listen(1)

print('listening on', addr)

while True:
    cl, addr = s.accept()
    print('client connected from', addr)
    cl_file = cl.makefile('rwb', 0)
    while True:
        line = cl_file.readline()
        #print(line)
        if not line or line == b'\r\n':
            break
    rows = ['<tr><td>%s</td><td>%d</td></tr>' % (str(p), p.value()) for p in pins]
    response = html % '\n'.join(rows)
    cl.send(response)
    cl.close()
```

As usual, you may use the WebREPL or `ampy` (recommended) to run programs. When you run the program, you should now be able to access the server from a browser at the address `192.168.4.1:80` and see a page with the current status of the ports.

TASK 1

Give a detailed explanation of the program such that the purpose of every line of code is accounted for.

TASK 2

Adapt the program to show the status of the inputs and sensors that you set up in the previous assignment, this includes the buttons, the temperature sensor and, optionally, the potentiometer.

Give a brief description of your changes. Document your test of the program with a screenshot.

OPTIONAL: If you know how, you may improve the graphical appearance by using *in-line styling*, i.e. style options attributed to the HTML elements.

Note that the given server is very primitive. Almost any line received will be interpreted as a status request. Also, the response lacks the usual HTTP response format (with error code and header lines) relying on the browser to show the HTML page anyhow.

3 Web API

The Web server must now be extended to make it possible to control the board from remote programs. Whereas HTML pages may be fine for a human recipient, a site should provide its information in a more direct way when accessed by other programs. This is the purpose of a Web API (Application Program Interface).

3.1 Web API notion

When we talk about the Web, we typically think of a number of services accessed using the HTTP protocol and connected via hyperlinks in HTML documents.

Since HTTP is the dominant application protocol on the internet, almost all firewalls will allow HTTP traffic even if other traffic is blocked. Therefore, it has become customary to use the HTTP protocol as a carrier for other forms of services typically known as *Web services*.

Therefore, the term Web service can be used in a very broad sense as any service provided by a server accessed via the HTTP protocol. This is illustrated in figure 1 where HTTP POST commands are used.

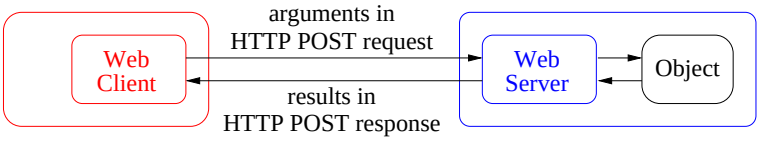

Figure 1: Basic protocol scenario

The Web service notion may also refer to a more specific setting where the services are described in a standardized version of XML called SOAP (Simple Object Access Protocol) [4, 3].

For this assignment, we are going to use a simpler form of Web service most often referred to as a *Web API*. This is a form of protocol, where the basic HTTP requests are used to operate upon objects on the server [5].

3.2 Web API guidelines

There are no rules for how Web APIs are defined, but it is common practice to adhering to a set of principles known under the acronym REST [2]. For this assignment, we merely focus on the use of the HTTP protocol. In the following, you can find a list of the basic principles and requests for an HTTP-based RESTful Web API.

- A site holds a set of *named resources* identified by (the path part of) URIs. Resources are typically *collections* of *elements* which may be identified hierarchically. For example, there may be a collections of pins named `/pins/` of which a specific pin may be named `/pins/pin5`.
- The contents (state) of a resource may be interrogated with an HTTP `GET` request. The state may be returned in various forms, but often the JSON format is used (see below). `GET` requests should never change the contents of the resource.
- The contents (state) of a resource may be replaced using the HTTP `PUT` request supplying the new contents in its body.
- The contents (state) of a resource may be modified (rather than totally replaced) using the HTTP `PATCH` request.
- A new element of a collection may be created with the HTTP `POST` request.
- A resource may be deleted by the HTTP `DELETE` request.

Although these are the principles, they are not always followed. For example, a particular element may be identified by a *query parameter*, e.g. as `/pins/pin?no=5` or new contents may be included in the query part in order to avoid sending a request body.

3.3 JSON

In order to transfer (larger amounts) of information in a Web API, the JSON data interchange notation [1] may be used. JSON (pronounced as the name Jason) originates from JavaScript, as a means of representing data to be transferred between a client-side Web application and the Web server.

JSON is a very simple textual format. A JSON element belongs to one of the types:

- **Numbers** with values like `-17`, `3.14159`, `2.99792458E8`
- **Booleans** with the two values `true` and `false`
- **Strings** with values like `"Temp1"` and the empty string: `""`
- **Null** with a single value denoted by `null`
- **Lists** (aka arrays) which are ordered sequences of elements of the form `[` e_1`,` e_2`, ...,` e_n `]` The size n is arbitrary and the elements need have the same type.

- **Maps** (aka objects) which are (unordered) sets of key-value pairs of the form `{` key_1`:`val_1`,` key_1`:`val_1`,` $\ldots$`,` key_n`:`val_n `}` The size n is arbitrary as are the types of keys and values.

White-space (blanks, tabs and newlines) is allowed almost everywhere allowing for an indentation-structured presentation.

For a particular application a *data scheme* is often used, containing a list of elements belonging to the same type. The scheme may be expressed in a formal notation (even JSON itself) or just stated in words.

The JSON data types correspond naturally to Python types. Often JSON text can be generated directly with a few lines of codes, but for general conversion, the `json` package may be used.

TASK 3

> **Define a simple Web API for interrogating the state of pins and sensors. Extend your Web server such that it returns the requested state information in JSON form. In other words, when the server is interrogated with a certain query (see the suggested resource paths below), instead of responding with an HTML page, it should respond with a JSON-formatted information.**
>
> **Describe the API, the JSON scheme used, and how your server has been adapted.**

Note that the Web API will have to be served on port 80 as well. Therefore, your server must distinguish normal Web requests identified by an empty resource path from the Web API requests where a specific resource path is given. TIP: you can print out the received request line in order to see the actual `GET` requests in order to better understand how they are formatted.

You e.g. may use the following resource paths:

Path	**Returns**
`/pins`	List of pin names
`/sensors`	List of sensor names
`/pins/`*`pin_name`*	Value of pin (number or boolean)
`/sensor/`*`sensor_name`*	Value of sensor (floating point number)

Also, the server should return proper HTTP-responses with suitable error codes.

TIP: some browsers know how to show the JSON result in a readable form if you use a `Content-Type` response header with the MIME type `application/json`. Otherwise, you may use your simple HTTP client program developed at a previous exercise to print the JSON response.

TASK 4

> **Write a laptop monitor program in Python that will enable you to log the board temperature over a specific period of**

time. The program should periodically request the temperature, print it to the terminal and record it to a simple text file (with a time stamp). From the file, it should be possible to generate a graph that shows how the temperature has developed. For this, you may use any visualization tool you know of (e.g., MS Excel).

Describe the format of the log file and how your monitor program works.

TASK 5

Enhance your Web server to read and write the value of (selected) I/O pins upon request. Write a simple Python control program from which read the status of the buttons and from which you can control the LEDs remotely. More specifically, form the Python script running on your laptop, you should be able to read the status of the buttons, turn on and off the LEDs and change the colour of the NeoPixel LEDs.

Describe the extension made to the API, the changes in the server and how your control program works.

OPTIONAL: If you have time, you can extend your programs to be able to read the temperature sensor and the potentiometer upon request.

4 Assignment tasks and report requirements

In the following, we summarize the steps that you should perform in this assignment and the requirements for the short report. Remember to hand-in all the source code for the tasks (preferably divided into folders) and, if necessary, a README file with instructions on how to run the programs. Also, hand-in screenshots proving the correct functionality of the programs you implemented.

These are the tasks you should perform in this assignment:

1. Read carefully the entire assignment document in order to get a full picture of what is expected from you.
2. Research and acquire information about HTTP, HTML, and JSON.
3. Carry out and document the five tasks.
4. Prepare a short report according to the instruction provided in the following.

The short report should not be longer than 6 pages (everything included) and should provide a description of what you did and the functionality of each of the five tasks. There are no specific requirements on the template to be used for the short report. Do not include the full code in the report. As usual, you

can include some code snippets if these are relevant to explain certain aspects of your programs.

References

[1] *JSON*.
`https://en.wikipedia.org/wiki/JSON`.

[2] *REST*.
`https://en.wikipedia.org/wiki/Representational_state_transfer`.

[3] *Soap*.
`https://en.wikipedia.org/wiki/SOAP`.

[4] W3C. *Simple Object Access Protocol (SOAP) 1.1*, May 2000.

[5] *Web API*.
`https://en.wikipedia.org/wiki/Web_API`.