

02135 Assignment 4, group 49

Jacopo Ceccuti s215158 and Andrea Fratini s215177

Introduction

In this assignment we focused on building a very simple server that runs on the Huzzah32 board. This server should provide:

- Web server that can be accessed by a browser to visualize an HTML page
- Web API that allows us to interact with the board from a browser

The code to use the board as an access point was provided, but first we had to change the WiFi SSID and the WiFi password. To run the program and modify our code we used Thonny, an IDE for Micropython which allowed us to operate in a single environment and have everything under control in an easier way.

Task 1

The first task asked us to give a brief explanation of every line of code that was provided in order to show that we have an understanding of the different parts of the program. To do so we went line by line in the code and looked up the syntax to understand how everything works. The results of this task can be found in “Code description” in the attached material.

```
rows = ['<tr><td>%s</td><td>%d</td></tr>' % (str(p), p.value()) for p in pins] # creates the updated rows with the certain p
response = html % '\n'.join(rows) # joins the rows and saves it in the variable "response"
cl.send(response) # sends a certain response to the client socket
cl.close() # close the created socket
```

Snippet of the work done in “Code description”

Task 2

The second task asked to show the status of the inputs and sensors that we set up in assignment 3. In particular we decided to focus on task 1 and task 3, also known as the blinking LED and the traffic lights with the use of the temperature sensor.

For task one (referring to assignment 3) the same disposition of cables, resistor and buttons was used with the addition of a new wire that connects to the board's pin 12 to the LED, also the *interrupt* method was implemented in the code in order to have the LED in a continuous status of blinking even when the request was received:

```
timer_1= machine.Timer(0)
timer_1.init(period=500, mode=machine.Timer.PERIODIC, callback=toggle_led)
```

Snippet of the interrupt method

The result of the blinking LED can be seen on the web browser page, in particular focusing on the state of Pin 12 that goes from a 0 to a 1 (Demo video uploaded as a separate content). Script is attached as “server_1 Task1.py”.

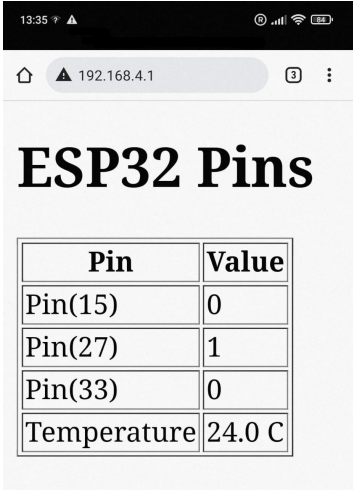
For task three (referring to assignment 3) we used the same layout as the traffic light assignment but again we added three more cables to check the led status, pin numbers: 12-27-33.

```
if temperature<25:
    Green.on()
    Yellow.off()
    Red.off()
if temperature>=25 and temperature<=28:
    Green.off()
    Yellow.on()
    Red.off()
if temperature>28:
    Green.off()
    Yellow.off()
    Red.on()
```

Snippet where the thresholds for the temperature can be seen

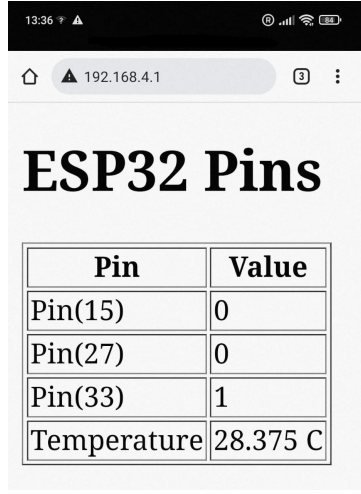
In the browser it is also possible to visualize the current temperature measured when the request arrives. Once again the demo video is uploaded as a separate content. Script is attached as “server_2 Task3.py”.

We are now reporting two images of the traffic lights Web page, proving the lights and temperatures changing.



Pin	Value
Pin(15)	0
Pin(27)	1
Pin(33)	0
Temperature	24.0 C

24°C and Green LED on



Pin	Value
Pin(15)	0
Pin(27)	0
Pin(33)	1
Temperature	28.375 C

28°C and Red LED on

Web API and JSON

The purpose of a Web API (Application Program Interface) is to give the site's information in a more direct way, often less human friendly, to a program or application that requests them. A Web API is a simple form of web service; with web service we describe any service provided by a server that uses the HTTP protocol as a communication protocol. The web API uses HTTP requests to interact with objects on the server.

Often a lot of informations are required in a Web API and one way to transfer them is with the JSON data representation. JSON is a type of notation used to transfer data between a Web application and a Web server, it supports different text formats such as: number, booleans, strings, null, lists and maps.

Task 3

To complete the third task of the assignment, we were asked to familiarize with the JSON format. This format is different from the HTML we are used to working with. We decided to keep things simple and we replicated the functions used in “Task 2” but instead of using the HTML format we gave as output a JSON format in our web page.

We had to make some changes to our server in order to change the output format, and we will now explain the changes we made. Firstly we had to import the library that allowed us to print a JSON output.

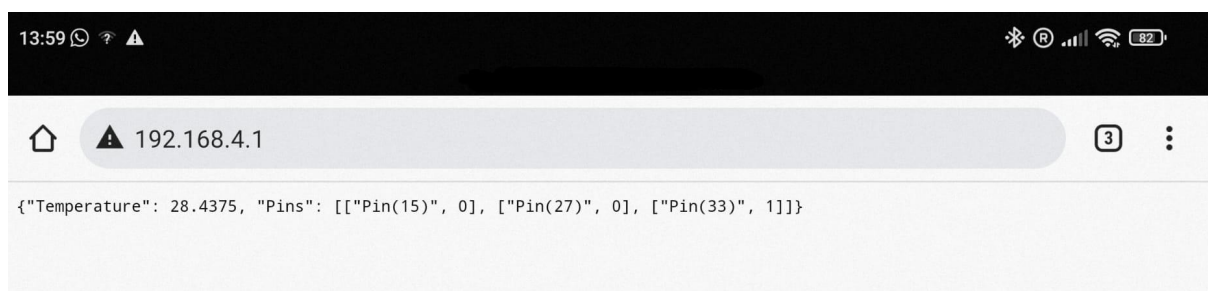
```
JSON_VALUES = {  
    "Pins": p,  
    "Temperature": temperature}
```

Fig. 1

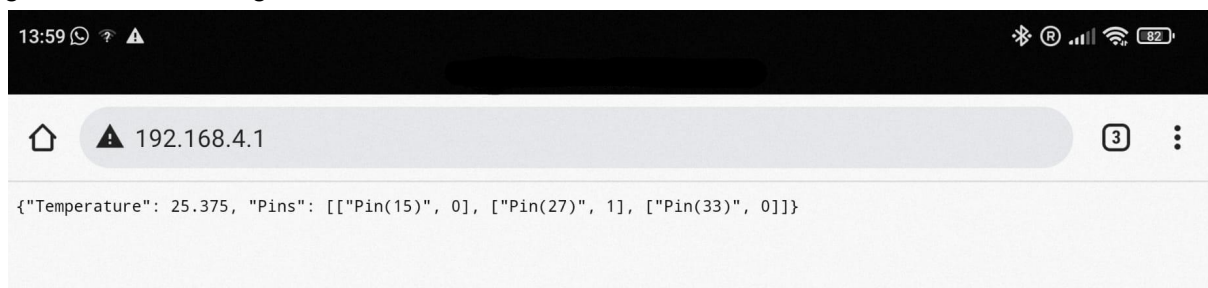
```
response = json.dumps(JSON_VALUES)  
cl.send(response)  
cl.close()
```

Fig. 2

As we can see from the two snippets of code, we have used two important functions, taken from the JSON library. The `JSON_VALUES` associates to strings, real variables as seen in Fig.1 or in our “server_1 Task3.py”. The second snippet is the `json.dumps()` function, this allows us to turn the `JSON_VALUES` into proper JSON formatted strings that can be sent to the client. Now we will show two different requests made while testing this task.



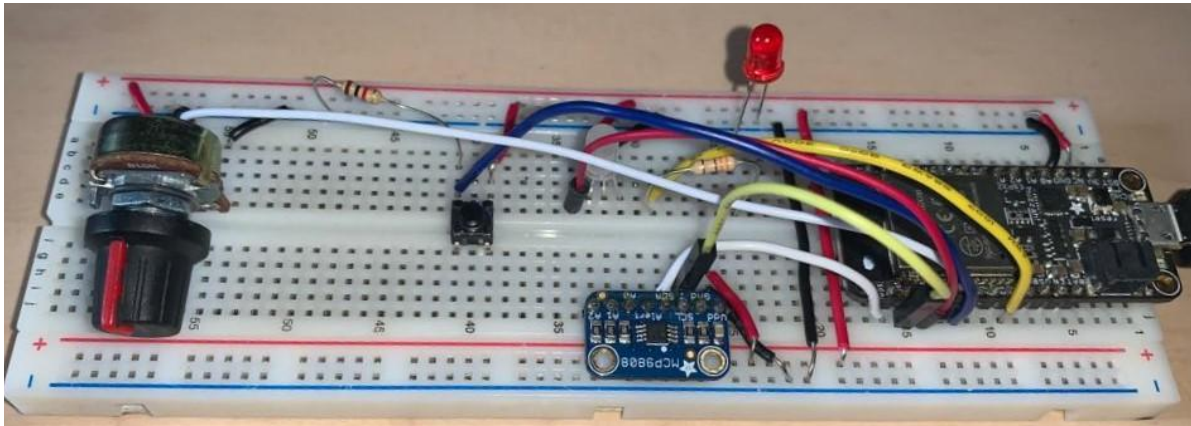
As we can see in the image above, our temperature value is 28.4375°C, as seen previously in our code snippet, for a value higher than 28°C the red LED must turn on, since we have connected the red LED to the Pin n.33 we know from the web page that it is on while the green and the orange LEDs are off.



In a different temperature situation we notice that the Pin n.27, connected to the green LED turns on when the temperature is below 25.5°C.

Task 4

In the last task of this assignment we are asked to summarize the knowledge we acquired during the whole assignment and work with the Huzzah32 and the WebAPI as best as we can. We can now implement a new function to our server, we can take very specific requests and for example “read the value from the potentiometer” or “change color to the NeoPixel”. For the final task we wanted to implement as many functions as possible on the breadboard, before analyzing the server’s code we are going to take a look at the new configuration of the board.



We can see that we have connected a potentiometer, a NeoPixel, the temperature sensor, a button and a red LED.

As stated before, we are now going to look at our new code, focusing on the crucial parts that allow us to perform new “GET” commands. The server now needs to be able to read the path written in the webpage. We take this command as example:

“192.168.4.1:80/pins/pin14/sethigh/”

We now have to create a specific server able to handle this type of requests. In this case turning on the LED connected to “pin14”. To do so we implemented the following snippet of code in our “server_4”.

```
web_path = ""

while True:
    line = cl_file.readline()

    web_path = web_path + str(line)

    print(line)
    if not line or line == b'\r\n':
        break

JSON = {"LED": LED.value(), "Button": button, "Temperature": temp, "Potentiometer": potentiometer, "NeoPixel": NeoLED[0]}

State = JSON
web_path_sti = web_path.split('/')
```

We can now focus on the “web_path” variable, this function is the one creating a proper web path. Our next focus will be the “path handling”, once we can add paths to our requests, we need to change the server in order for it to answer this new type of requests properly. We are now going to show only one of the if-statements that we have created for every possible path requestable.

```

# Changes the NeoPixel's color
if "pins" in web_path_sti and "pin14" in web_path_sti and "green" in web_path_sti:
    State = JSON

    NeoLED[0] = (255, 0, 0)
    NeoLED.write()

    del State["Button"]
    del State["Temperature"]
    del State["Potentiometer"]
    del State["LED"]

    State = list(State.values())

```

This is only a small piece of code that gets repeated multiple times in order to handle multiple colors and sensor's values.

To conclude our report we are going to list all of the possible paths that we have implemented, followed by a short description of each one.

Request		Functionality
/pins/pin12/sethigh/	→	Turns the LED on
/pins/pin12/setlow/	→	Turns the LED off
/pins/pin14/sethigh/	→	Turns the NeoPixel on
/pins/pin14/setlow/	→	Turns the NeoPixel off
/pins/pin14/green/	→	Changes the NeoPixel color to green
/pins/pin14/red/	→	Changes the NeoPixel color to red
/pins/pin14/blue/	→	Changes the NeoPixel color to blue
/pins/pin14/pink/	→	Changes the NeoPixel color to pink
/pins/pin14/purple/	→	Changes the NeoPixel color to purple
/pins/button/	→	Shows the status of the button
/pins/potentiometer/	→	Shows the status of the potentiometer
/sensors/temperature/	→	Shows the temperature