IoT exercises – Week 10-11

This week (plus next week if needed), we will use NodeMCU/ESP8266 as a platform with a http and web based server built on it. Simple user interfaces will be created in combination with basic sensor control, file read and write, and . The net module will be further explored (based on your exercise in Week 6) in this practical together with the http module. The file module will also be used to write and read for access to files in NodeMCU/ESP8266.

The official documentation is provided here again for your reference. Please remember, when you are building your own IoT project in the future, always refer to the bespoke built-in modules and read their documentation first, which will be helpful!!!

https://nodemcu.readthedocs.io/en/master/

The details of net, http and file module can be found in

https://nodemcu.readthedocs.io/en/master/modules/net/

https://nodemcu.readthedocs.io/en/master/modules/http/

https://nodemcu.readthedocs.io/en/master/modules/file/

Exercise 1:

In this exercise, you will need to use your NodeMCU/ESP8266 as a web crawler to get the information from websites. Different from previous exercises, we will see the limitation of our NodeMCU/ESP8266 tool in dealing with the http page.

Get online first and then try to crawl the sample page and the amazon page. Check the failure and think about the reason why it fails for non-json format html. **(Optional Challenge)** Figure out how to avoid the failure you have met.

==========

```
function crawl()

url='http://httpbin.org/ip'

--url='http://www.amazon.co.uk'
```

```
--url='http://wttr.in/'

--try other urls and see why they can work or why not

print(url)

headers={['user-agent'] = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.117 Safari/537.36'}

--headers to avoid the website to recognise you as a robot

http.request(url,'GET',headers,'',function(code, data)

        if (code<0) then

            print("HTTP request failed")

            print(code)

        else

            print(code, data)

        end

    end)

end
```

==========

Exercise 2:

Now you have seen the request function in http module. In this exercise, you will use more functions/methods of delete, put and post with the following codes slightly changed from the official documentation of the module. And please check the format of the data and then refer to the url of http://httpbin.org/put, http://httpbin.org/post and http://httpbin.org/delete directly in an internet browser to see if the data can be accessed. If not, read the message presented to you and check what a HTTP request method is.

Note that GET is to retrieve the data on the server, POST is to modify the data on the server, PUT is to add data to the server, Delete is to remove data from the server. Normally GET and POST are used more.

==========

```
function post()

    http.post('http://httpbin.org/post',

        'Content-Type: application/json\r\n',
```

```
        '{"IoT":"2020","This is":"Json Format","Please check":'..
        '"How the data are shaped"}',
        function(code, data)
          if (code < 0) then
            print("HTTP request failed")
          else
            print(code)
            print(data)
          end
        end)
end

function put()
      http.put('http://httpbin.org/put',
        'Content-Type: text/plain\r\n',
        'IoT 2020 plain text, please check how the data are formatted',
        function(code,data)
          if (code < 0) then
            print("HTTP request failed")
          else
            print(code)
            print(data)
          end
        end)
end

function delete()
      http.delete('http://httpbin.org/delete',
```

```
    "",
    "",
    function(code,data)
      if (code < 0) then
        print("HTTP request failed")
      else
        print(code)
        print(data)
      end
    end)
end
```

==========

Exercise 3:

In this exercise, you will use the file module, write what you get from an HTTP request in a local file, and then retrieve the data from the local file by reading and post it to the test site with the appropriate HTTP request method. Exercise 1 and 2 will be combined with your file operation. Here only a limited number of functions are listed. More to be found in https://nodemcu.readthedocs.io/en/master/modules/file/. Don't forget to close the file after use.s

==========

```
fileList = file.list()
print(type(fileList))
for name,size in pairs(fileList) do
    print("File name: "..name.." with size of "..size.." bytes")
end


--mode can be 'w' 'r' 'a' 'r+' 'w+' 'a+'
fobjw = file.open('samplefile.txt','w')
fobjw:writeline('IoT first string')
```

```
fobjw:write('second string')

fobjw:write('third string')

print(fobjw.read())

print(fobjw.readline())


fobjr = file.open('samplefile.txt','r')

fobjr:writeline('IoT first string')

fobjr:write('second string')

fobjr:write('third string')

print(fobjr.readline())

fobjr:close()


fobjr = file.open('samplefile.txt','r')

print(fobjr.read())

print(fobjr.read())

fobjr:close()

fobjr = file.open('samplefile.txt','r')

print(fobjr.read())


fobjr:close()

fobjr = file.open('samplefile.txt','r')

print(fobjr.seek("cur",11))

--"set": base is position 0 (beginning of the file)

--"cur": base is current position (default value)

--"end": base is end of file

print(fobjr.read())

--skip 11 charactres from current position

print(fobjr.seek("cur",-5))
```

```
--output the last 5 characters

print(fobjr.read())
```

==========

Exercise 4:

Recall the web page based DHT11 temperature and humidity display in Week 6. In this exercise, you will need to create an HTTP based server and listen to the client. The control button will be added to turn ON/OFF of the LED.

==========

```
pinLED = 4

gpio.mode(pinLED,gpio.OUTPUT)

svr = net.createServer(net.TCP)

function htmlUpdate(sck,flag)

--update the html file for display in your browser

        html = '<html>\r\n<head>\r\n<title>LED LAN Control</title>\r\n</head>\r\n'

        html = html..'<body>\r\n<h1>LED</h1>\r\n<p>Click the button below to switch LED on and off.</p>\r\n<form method=\"get\">\r\n'

--method is get here, listener will try to find the get info

        if flag then

--compare the boolean logic here and below in the receiver

        strButton = 'LED_OFF'

        else

        strButton = 'LED_ON'

        end

        html = html.."<input type=\"button\" value=\""..strButton.."\" onclick=\"window.location.href='/"..strButton.."'\">\r\n"

    -- add the different button

        html = html.."</form>\r\n</body>\r\n</html>\r\n"

        sck:send(html)
```

```
    end
    function setMode(sck,data)
      print(data)
--check what is the data received, and figure out why we find the match pattern in the string
      if string.find(data, "GET /LED_ON")  then
       htmlUpdate(sck, true)
       gpio.write(pinLED, gpio.HIGH)
      elseif string.find(data, "GET / ") or string.find(data, "GET /LED_OFF") then
       htmlUpdate(sck, false)
       gpio.write(pinLED,gpio.LOW)
      else
--if no match found then close the connection after sending a notice using the socket for the last will
       sck:send("<h2>Error, no matched string has been found!</h2>")
       sck:on("sent", function(conn) conn:close() end)
      end
    end


if svr then
      svr:listen(80, function(conn)
--listen to the port 80 for http
--when the event of 'data is received' happens, run the setMode
      conn:on("receive", setMode)
      end)
end
```

==========

Please browse "192.168.4.1/LED_ON" and "192.168.4.1/LED_OFF", the IP address of "192.168.4.1" could be different and changed in your settings. And see if the light is controlled. Then look into the code and think about what is the control signal here in this example. The button or some other hints? If you are still confused, try

“192.168.4.1/LED_ONdalin”, “192.168.4.1/LED_OFFiot”

(Optional Challenge) Exercise 5:

Now you are expected to create the logging system based on the previous exercises of the IoT unit attendance using an HTTP based server. It allows the student to browse an url/IP address where they can sign in their attendance. The signed-in results will be written and saved in a local file on your nodeMCU containing 2 properties of the Student ID and the Student Name. The differences from Exercise 4 are that you would need to add the text boxes to type in the data for file recording/writing, and the button function requires slight modification before use. **Regular Expression** is likely to be applied to your application to filter out the name and id you need.

Further support could be referred to in https://www.w3schools.com/.

The following minimalistic libraries and built servers can be found for you to use to simplify your implementation:

Back-end

nodemcu_http_server https://github.com/borischernov/nodemcu_http_server

Front-end

Zepto.js https://github.com/madrobby/zepto

Spectre.css https://github.com/picturepan2/spectre

(Optional) Exercise 6:

Further extending the Exercise 5, please include constraints to your system that a unique device can only be used to sign in the attendance for once. MAC address is recommended for use.