Jorge Vicente Camuñas Heredia **04/11/20**

Práctica 5. Servidores REST y representación de la información. JSON y CBOR

En esta práctica se empezó desarrollando el ejemplo del GitHub sobre servidores REST y viendo su funcionamiento. Una vez visto lo básico y haber experimentado con ello se pide como primera tarea añadir al código original un nuevo endpoint que devuelva el valor de temperatura en grados Fahrenheit. Como resultado de dicho cambio se obtuvo el siguiente resultado:

```
I (7166) example_connect: Got IP event!
I (7666) example_connect: Got IPv6 event!
I (7666) example_connect: Connected to jorgeswifi
I (7666) example_connect: IPv4 address: 192.168.43.209
I (7666) example_connect: IPv6 address: fe80:0000:0000:0000:260a:c4ff:feea:3cc8
I (7876) example: Partition size: total: 1920401, used: 1622213
I (7876) esp-rest: Starting HTTP Server
I (263716) esp-rest: File sending complete
I (265236) esp-rest: File sending complete
I (265326) esp-rest: File sending complete
I (267836) esp-rest: File sending complete
I (268556) esp-rest: File sending complete
I (268956) esp-rest: File sending complete
I (278206) esp-rest: Light control: red = 160, green = 160, blue = 160
I (283136) esp-rest: Light control: red = 58, green = 160, blue = 160
I (291146) esp-rest: File sending complete
I (367256) esp-rest: File sending complete
```

Figura 1: Monitorización ESP32 rest_server

```
ubuntu@ubuntu2004:~$ curl http://192.168.43.209/api/v1/temp/raw
{
        "raw":  9
}ubuntu@ubuntu2004:~$ curl http://192.168.43.209/api/v1/temp/raw
{
        "raw":  9
}ubuntu@ubuntu2004:~$ curl http://192.168.43.209/api/v1/temp/fahrenheit
{
        "fahrenheit":   50
}ubuntu@ubuntu2004:~$ curl http://192.168.43.209/api/v1/temp/fahrenheit
{
        "fahrenheit":   66
}ubuntu@ubuntu2004:~$ curl http://192.168.43.209/api/v1/temp/fahrenheit
{
        "fahrenheit":   39
}ubuntu@ubuntu2004:~$ curl http://192.168.43.209/api/v1/temp/fahrenheit
{
        "fahrenheit":   51
```

Figura 2: Valores de temperatura en grados Celsius y grados Fahrenheit

Como tarea siguiente y una vez visto como trabajar con los archivos JSON se pide realizar los cambios necesarios en el endpoint Farenheit para que devuelva un valor real. El resultado puede verse en la siguiente figura:

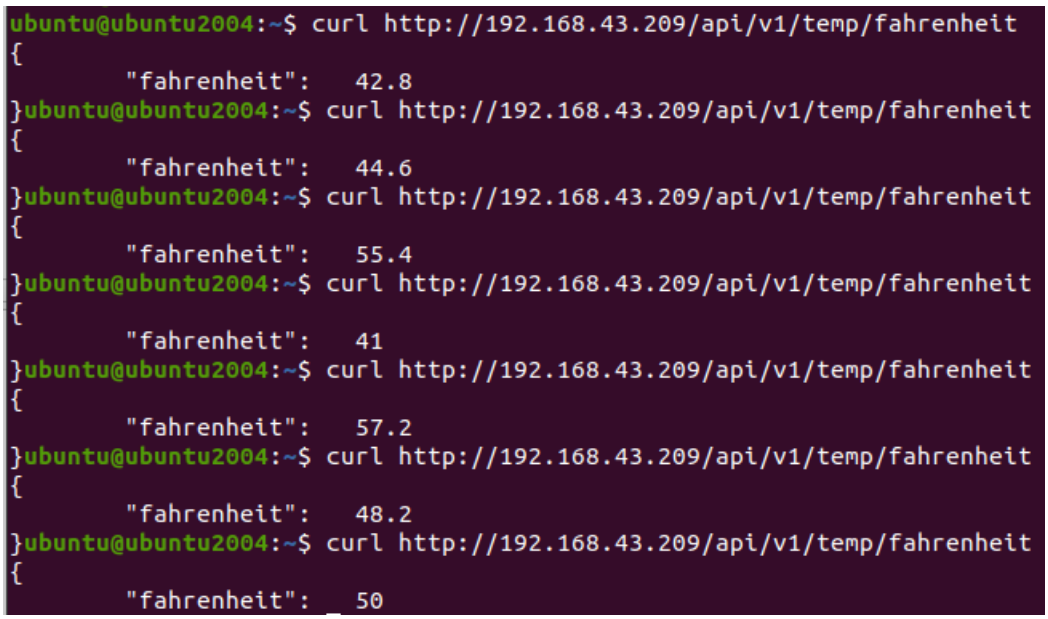

Figura 3: Valores reales de temperatura en grados Fahrenheit

Para seguir profundizando en los JSON se pide como siguiente tarea añadir un nuevo endpoint con datos de distintos tipos que emulen el comportamiento de un entorno IoT. Para ello se ha creado el endpoint “datos” que incluye tres valores:

1. Distancia: valor real en cm que emula el comportamiento de un sensor infrarojo.

2. Temperatura: valor entero en grados celsius que pretende emular el comportamiento de un sensor de temperatura.

3. Luz: valor booleano que emulará la presencia o no de luz frente a un sensor óptico.

Con estos valores generados de forma aleatoria ante la petición de “datos”, ya sea usando curl o via pagina web, se mostrará al cliente dicha información como puede verse en las siguientes imágenes:

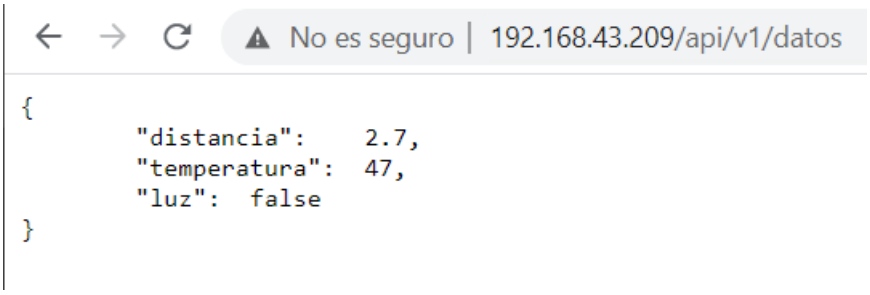

Figura 4: Datos obtenidos en página web

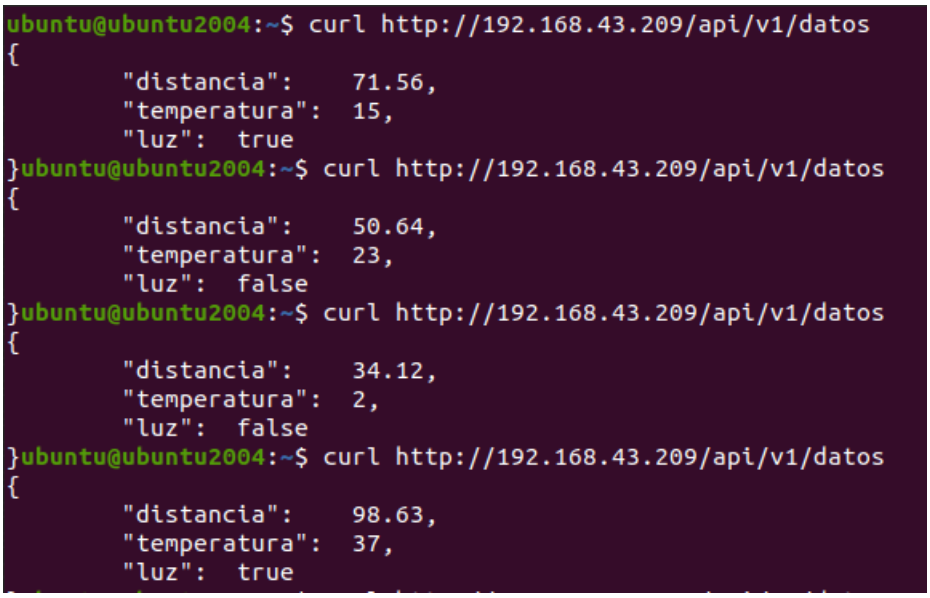

Figura 5: Datos obtenidos con curl

Se pide en la tarea también incluir una captura de Wireshark de la comunicación entre cliente y servidor:

No.	Time	Source	Destination	Protocol	Length	Info
3	3.74…	192.168.43.196	192.168.43.209	TCP	74	52878 → 80 [SYN] Seq=0 Win=64240 …
4	3.89…	192.168.43.209	192.168.43.196	TCP	60	80 → 52878 [SYN, ACK] Seq=0 Ack=1…
5	3.89…	192.168.43.196	192.168.43.209	TCP	54	52878 → 80 [ACK] Seq=1 Ack=1 Win=…
6	3.89…	192.168.43.196	192.168.43.209	HTTP	144	GET /api/v1/datos HTTP/1.1
7	3.93…	192.168.43.209	192.168.43.196	TCP	123	80 → 52878 [PSH, ACK] Seq=1 Ack=9…
8	3.93…	192.168.43.196	192.168.43.209	TCP	54	52878 → 80 [ACK] Seq=91 Ack=70 Wi…
9	3.94…	192.168.43.209	192.168.43.196	HTTP	112	HTTP/1.1 200 OK (application/jso…
10	3.94…	192.168.43.196	192.168.43.209	TCP	54	52878 → 80 [ACK] Seq=91 Ack=128 W…
11	3.94…	192.168.43.196	192.168.43.209	TCP	54	52878 → 80 [FIN, ACK] Seq=91 Ack=…
12	3.95…	192.168.43.209	192.168.43.196	TCP	60	80 → 52878 [ACK] Seq=128 Ack=92 W…
13	3.95…	192.168.43.209	192.168.43.196	TCP	60	80 → 52878 [FIN, ACK] Seq=128 Ack…
14	3.95…	192.168.43.196	192.168.43.209	TCP	54	52878 → 80 [ACK] Seq=92 Ack=129 W…

Figura 6: Captura Wireshark petición JSON "datos"

Cabe destacar como los mensajes HTTP y PUSH son los más pesados con diferencia y estos datos se utilizarán más adelante en la práctica al comparar con CBOR.

Como última tarea se pide añadir un nuevo endpoint con la diferencia de usar el formato CBOR en lugar de JSON. Tras realizar los cambios necesarios para adecuar el mensaje anterior (datos) al nuevo endpoint (datos_cbor) se recoge lo recibido usando curl en un archivo "output.cbor". Utilizando el programa en python proporcionado en el boletín se puede ver el contenido del mensaje:

```
ubuntu@ubuntu2004:~/RPI2/Prac5$ curl http://192.168.43.209/api/v1/datos_cbor > output.cbor
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    36  100    36    0     0    352      0 --:--:-- --:--:-- --:--:--   352
```

Figura 7: Petición de datos_cbor para guardarlo en el archivo output.cbor

```
ubuntu@ubuntu2004:~/RPI2/Prac5$ python3 cbor_reader.py
[{'distancia': 66.0199966430664, 'temperatura': 33, 'luz': True}]
```

Figura 8: Contenido del archivo CBOR

Como puede verse en la figura 8 el contenido de datos_cbor es igual al de datos. Se ha estructurado en un mapa para acompañar cada valor con un string. Para comparar la diferencia de peso entre transmitir un JSON o un CBOR se ha realizado una captura del tráfico generado por CBOR que se muestra a continuación:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00…	192.168.43.196	192.168.43.209	TCP	74	53404 → 80 [SYN] Seq=0 Win=64240 …
2	0.07…	192.168.43.209	192.168.43.196	TCP	60	80 → 53404 [SYN, ACK] Seq=0 Ack=1…
3	0.07…	192.168.43.196	192.168.43.209	TCP	54	53404 → 80 [ACK] Seq=1 Ack=1 Win=…
4	0.07…	192.168.43.196	192.168.43.209	HTTP	149	GET /api/v1/datos_cbor HTTP/1.1
5	0.09…	192.168.43.209	192.168.43.196	TCP	123	80 → 53404 [PSH, ACK] Seq=1 Ack=9…
6	0.09…	192.168.43.196	192.168.43.209	TCP	54	53404 → 80 [ACK] Seq=96 Ack=70 Wi…
7	0.10…	192.168.43.209	192.168.43.196	HTTP	92	HTTP/1.1 200 OK
8	0.10…	192.168.43.196	192.168.43.209	TCP	54	53404 → 80 [ACK] Seq=96 Ack=108 W…
9	0.10…	192.168.43.196	192.168.43.209	TCP	54	53404 → 80 [FIN, ACK] Seq=96 Ack=…
10	0.10…	192.168.43.209	192.168.43.196	TCP	60	80 → 53404 [ACK] Seq=108 Ack=97 W…
11	0.11…	192.168.43.209	192.168.43.196	TCP	60	80 → 53404 [FIN, ACK] Seq=108 Ack…
12	0.11…	192.168.43.196	192.168.43.209	TCP	54	53404 → 80 [ACK] Seq=97 Ack=109 W…

Figura 9: Captura Wireshark petición a datos_cbor

Puede verse al comparar la figura 6 y 9 como la mayor parte de los mensaje al compartir protocolo son idénticos salvo el mensaje GET (que cambia porque la dirección de CBOR es un poco más larga) y el mensaje OK donde se encuentra el mensaje principal de la comunicación.

```
    TCP payload (58 bytes)
    TCP segment data (58 bytes)
▸ [2 Reassembled TCP Segments (127 bytes): #7(69), #9(58)]
▸ Hypertext Transfer Protocol
▸ JavaScript Object Notation: application/json

0000  08 00 27 95 9b 83 24 0a  c4 ea 3c c8 08 00 45 00   ··'···$· ··<···E·
0010  00 62 00 62 00 00 ff 06  e2 4d c0 a8 2b d1 c0 a8   ·b·b···· ·M··+···
0020  2b c4 00 50 ce 8e 00 00  1a 53 09 cf 0b 59 50 18   +··P···· ·S···YP·
0030  16 16 90 80 00 00 0d 0a  7b 0a 09 22 64 69 73 74   ········ {··"dist
0040  61 6e 63 69 61 22 3a 09  34 32 2e 30 35 2c 0a 09   ancia":· 42.05,··
0050  22 74 65 6d 70 65 72 61  74 75 72 61 22 3a 09 31   "tempera tura":·1
0060  2c 0a 09 22 6c 75 7a 22  3a 09 74 72 75 65 0a 7d   ,··"luz" :·true·}
```

Figura 10: Tamaño mensaje JSON

Para poder hacer mayor incapie en la diferencia de peso del paquete correspondiente al envío (paquete número 7) se han realizado dos capturas extra:

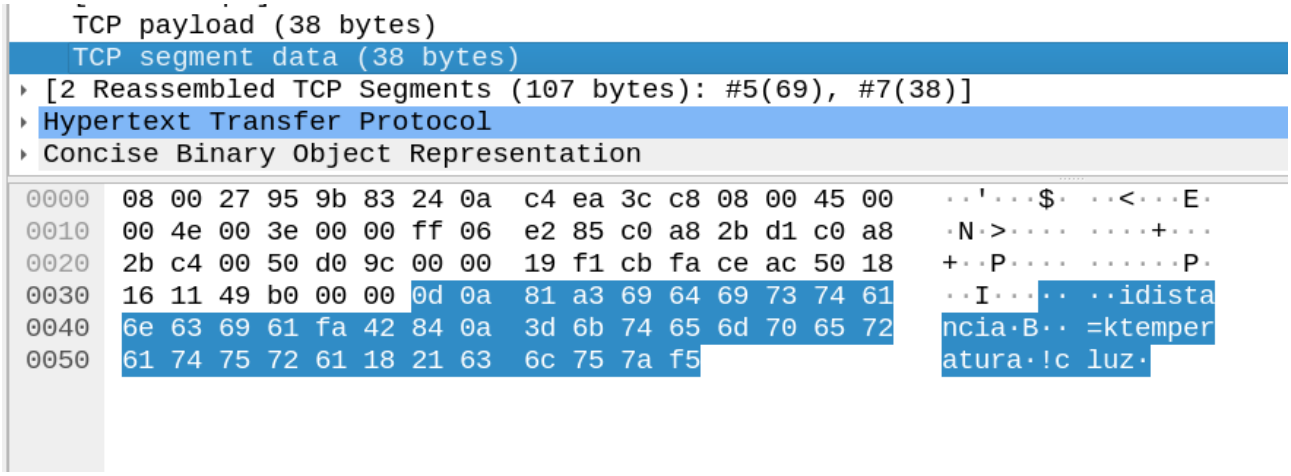

Figura 11: Tamaño mensaje en CBOR

En estas dos últimas capturas puede verse la diferencia de tamaño para el mismo tipo de mensaje. En el caso de estos datos hay una diferencia de 20 bytes entre usar un formato u otro.