Lab activity no. 12
A Model for the RESTful Integration of IoT Components

One of the initial approaches to encapsulating an IoT component as a RESTful WEB service (while maintaining the references from the previous lab session [1, 2, 3]) involves attaching this type of interface to the application managing the respective component. The steps involved are as follows:

1. modules are developed to enable *read*/*write* operations;
   - the method corresponding to the *read* function may, for instance, retrieve the next value measured by a sensor or provide information regarding the state of the encapsulated equipment;
   - the method corresponding to the *write* function may, for example, modify certain data acquisition settings or transmit a new command to a potential actuator-type device;
2. at the WEB service level, public *endpoints* are configured through which HTTP messages will be received, and the *interaction* component with the previously described methods is set up;
3. upon receiving an HTTP request, the following steps are carried out:
   (a) the methods corresponding to the desired operation are invoked (i.e., *read* or *write*, with the distinction made based on the HTTP *verb* present in the request);
   (b) the corresponding response is formulated and returned to the client.

Lab application

Building on the steps described above, adapt the REST service developed during the previous lab session to implement the following functionalities:

GET method call on a sensor ID a value provided by a sensor will be read; optionally, you may simulate multiple sensors;

POST method call on a sensor ID a configuration file will be created, which will allow, for example, modification of the scale used to represent the measured value:

- if the file does not exist, it will be created with a default name;
- if the file already exists, its recreation via the POST method will not be permitted, and a response of type `400 Bad Request`, `406 Not Acceptable`, or `409 Conflict` will be returned (for all three status codes, it is recommended that the response include a message body detailing the error encountered);

PUT method call on a sensor ID/configuration file name the configuration file will be replaced (if the file does not exist, its creation will not be permitted and one of the aforementioned status codes will be returned);

Remarks

1. Actual sensors may be simulated for demonstration purposes using separate processes – see [4, 5].
2. The actual structure of the REST service URLs is part of the laboratory assignment.
3. The HTTP server can be instantiated directly on the ESP32 development board – see [6].

References

[1] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000.

[2] Berners-Lee T. and Fielding R. and Masinter L. RFC 3986: Uniform Resource Identifier(URI): Generic Syntax. https://tools.ietf.org/html/rfc3986.

[3] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP Semantics. https://www.rfc-editor.org/rfc/rfc9110.html, June 2022.

[4] ESPRESSIF. Miscellaneous System APIs. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/system.html.

[5] ESPRESSIF. FreeRTOS – Task API. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html.

[6] ESPRESSIF. HTTP Server. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp_http_server.html.