Homework 3 - Team 13

Fabrizio Battiloro (s317786) - Lorenzo Mossotto (s318971) - Arash Daneshvar (s314415)

EXERCISE1: DATA COLLECTION, COMMUNICATION, AND STORAGE

Explaining the superiority of "MQTT" over "REST"

There is some reasons that show MQTT is a better choice than REST for monitoring and transmitting battery status data:

Efficient Real-Time Updates: The application involves continuously monitoring the battery level of laptops, which requires frequent and real-time updates. MQTT is highly efficient for this task as it allows for real-time communication through its lightweight publish-subscribe model. It can transmit small messages (like battery level updates) with minimal latency, making it ideal for applications that require immediate data delivery.

Low Bandwidth Utilization: Given that battery level data comprises small packets of information, MQTT's minimalistic protocol design is a perfect fit. It minimizes data overheads (small header size), which is especially beneficial when transmitting data over constrained networks. REST, in comparison, involves a heavier data transfer load due to the HTTP protocol's header requirements, making it less efficient for frequent, small data transmissions.

Persistent Sessions and Less Network Strain: MQTT supports persistent sessions, which is advantageous for battery monitoring applications. When a device connects to an MQTT broker, it can subscribe to a topic and remain connected for ongoing updates. This approach reduces the need to establish and tear down connections frequently (as is the case with REST), leading to less network strain and better battery conservation on the monitored devices.

Reduced Energy Consumption on Client Devices: : The battery monitoring application is inherently energy-sensitive since it deals with battery-powered devices. MQTT, being a lightweight protocol, consumes less power for data transmission compared to the more resource-intensive REST protocol. This is crucial for preserving the battery life of the devices being monitored.

Scalability for Multiple Devices: In a scenario where you need to monitor the battery levels of numerous laptops, MQTT's scalability becomes a significant advantage. It can handle a large number of concurrent connections efficiently, making it ideal for large-scale deployments without a proportional increase in overhead.

In summary, MQTT's real-time update capability, low bandwidth requirement, persistent sessions, reliable message delivery, reduced energy consumption, scalability, ease of integration with IoT platforms, and bi-directional communication capabilities make it particularly suitable for a battery level monitoring application.

TABLE I
HTTP METHODS FOR BATTERY MONITORING APPLICATION

Method	Endpoint	Motivation
GET	/devices	The GET method is chosen for its intended use to request and retrieve data from a resource without causing any side effects, ensuring the operation is safe and idempotent. It is ideal for listing resources, such as device MAC addresses, without changing the server's state.
GET	/device/{mac_address}	GET is appropriate here as it is used to read or retrieve data. It fits the requirement for obtaining specific battery status information for a device based on its MAC address within a date range, adhering to REST principles of data retrieval.
Delete	/device/{mac_address}	The DELETE method is correctly used to remove a resource from the server. It indicates the action to delete the timeseries data for a device, aligning with the RESTful approach where the method type signifies the operation on the resource.

EXERCISE2: DATA MANAGEMENT & VISUALIZATION

Suitable HTTP method and the motivation

As shows in the table I, the **GET** method is chosen for the endpoints /devices and /device/{mac_address} due to its standard use in retrieving data without side effects, aligning with the operations' read-only nature to list device MAC addresses and fetch battery status information respectively. The **DELETE** method for the /device/{mac_address} endpoint is appropriately used to remove the time series data from the server, as it clearly communicates the intent to delete a resource and is idempotent, ensuring that the operation has no further effect once the resource is deleted. These methods are consistent with REST principles, providing an intuitive and standardized interface for the API's clients.