**Comm A (sensors to Arduino):**

* **Analog Communication**
* The Grove - Temperature Sensor outputs a voltage signal that is proportional to the temperature. This sensor is connected to the A0 analog pin of the Grove-Base Shield. The sensor outputs a continuous voltage signal based on the temperature.
* The voltage signal is transformed into an analog reading ranging from 0 to 1024.
* Here, the communication is direct electrical signaling (analog voltage), which does not involve data packets or protocol layers.

**Comm B (Arduino to laptop):**

* **Serial Communication (UART - Universal Asynchronous Receiver-Transmitter)**
* UART is one of the most used device-to-device (serial) communication protocols. It’s the protocol used by Arduino boards to communicate with the computer.
* UART communication is sent over USB via the built-in USB-to-Serial converter on the Arduino.
* UART operates by transmitting data as a series of bits, including a start bit, data bits, an optional parity bit, and stop bit(s). Unlike parallel communication, where multiple bits are transmitted simultaneously, UART sends data serially, one bit at a time. As the name reveals, the protocol operates asynchronous which means that it doesn't rely on a shared clock signal. Instead, it uses predefined baud rates to determine the timing of data bits.
* The Arduino Uno and the laptop exchange data via a **serial interface** over a **USB connection**.
* The data is transmitted as a series of bits in a sequential manner (bit by bit).
* The Python code uses the **PySerial library** to read the data from the Arduino, which sends temperature values over its serial port.

Why UART for Comm B?

* Ease of Use: Arduino's default communication method with a PC is via its UART interface, accessible through the Serial library.
* Plug-and-Play: The USB cable not only powers the Arduino but also enables serial data exchange with your laptop.
* Cross-Platform: UART is universally supported across operating systems and requires no additional hardware for basic communication.

**Comm B’ (laptop to MongoDB database):**

* **TCP/IP (Transmission Control Protocol/Internet Protocol)**
* Clients communicate with the MongoDB database server through a regular TCP/IP socket. This is the standard for MongoDB.
* Sensor data often includes sensitive or proprietary information. MongoDB Atlas ensures secure transmission and storage of data, even if TCP/IP itself can have some privacy concerns.

### Comm C (Facilities Data in MongoDB)

**TCP/IP (Transmission Control Protocol/Internet Protocol)**  
Storage: The facilities database (facilities\_db) stores information about room specifications (e.g., seating capacity, projectors, computers, training robots).

Why TCP/IP for Facilities Data?

* Scalability: MongoDB allows seamless expansion as the number of rooms and facilities increases.
* Security: MongoDB Atlas encrypts data at rest and in transit, ensuring secure access.
* Flexibility: The NoSQL schema supports changes in facilities data without disrupting existing structure.

### Comm D (Agenda Data in MongoDB)

**TCP/IP (Transmission Control Protocol/Internet Protocol)**  
Storage: The agenda database (agenda\_db) stores time slots for room bookings to prevent conflicts in room reservation.

Why TCP/IP for Agenda Data?

* Real-Time Updates: The recommendation system frequently pulls agenda data to ensure accurate availability.
* Database Integration: Since both agenda and sensor data reside in MongoDB, TCP/IP ensures unified access.
* Security & Access Control: Fine-grained permissions restrict unauthorized modifications to room bookings.

**Comm E (Client to REST API)**

**HTTP/HTTPS**

The API follows the RESTful principles and communicates with the frontend via HTTP requests. GET and POST methods are used in our API implementation to reach and interact with the desired endpoints. All data exchanges between the API and the clients occur in JSON format. JSON provides a lightweight and human-readable way to transmit structured data.

*If we would have had an Arduino with Internet connection and a Raspberry Pi, then we would have proposed the following communication protocols for A and B:*

***Comm A: Modbus***

* ***Scalability:*** *Modbus RTU over a serial connection allows multiple sensors (e.g., temperature, sound, humidity, CO₂) to communicate efficiently with the Arduino by assigning each sensor a unique address.*
* ***Reliability:*** *It is an industry-standard protocol for connecting industrial devices and sensors. It ensures robust communication over short distances with minimal error.*

***Comm B: MQTT***

* *MQTT is lightweight, optimized for real-time communication with low bandwidth, and works very well for IoT applications like Arduino. It uses a publish-subscribe model that allows the Raspberry Pi (or any broker) to receive data from multiple Arduino sensors and push that data to MongoDB in near real-time.*
* *MQTT would allow you to push data directly from the Arduino to a broker (via the Raspberry Pi or another broker), which could then be stored in MongoDB or passed to any other service.*

**Comm C: HTTP (REST API)**

* **Interoperability:** Most databases or external systems (e.g., room facility control systems) already support HTTP-based REST APIs for interaction. This makes HTTP a natural choice.
* **Ease of Integration:** HTTP APIs can be easily implemented with libraries in most programming languages. This simplifies sending structured data (e.g., JSON or XML) between the Raspberry Pi and external systems.
* **Scalability:** REST APIs are stateless, allowing multiple concurrent requests. This ensures scalability if the number of room facilities grows.

**Comm D: HTTP (REST API)**

* Shared university agendas (like Google Calendar or similar systems) typically offer APIs that use HTTP and follow the RESTful architecture for integration.
* Many modern programming languages and libraries (e.g., Python requests or http.client) support HTTP REST API calls, simplifying development. For example, we can use Google's Calendar API over HTTP to fetch availability data and synchronize it with the DSS.