* The Pico.py code is to collect the temperature data from the Pico using the TCP socket communication every 30 seconds. It is sending as a dataframe as key-value pair. When the device is not sending the data, it is in deep sleep mode.
* The Fipy code collects the light sensor information using the UDP socket when the button is pressed. The data is sent as dataframe with key-value pair.
* In the main program app.py sets up the Flask application that acts as the IoT web server, this communicates with the Raspberry Pi gateway through the TCP and UDP connections. The sensor data collected are stored in a local SQLite database. The web server can send MQTT requests to the gateway to retrieve the latest sensor readings, and the gateway responds with the requested data. When there are 50 values of each of the temperature or light sensor it sends an email to me.

The code sets up the MQTT connection and subscribes to two MQTT topics: topic/temp\_request\_20648424 and topic/light\_request\_20648424. It also defines an event handler for when the MQTT client connects to the broker.

TCP Client Handling: The code defines a function **handle\_tcp\_client** that runs in a separate thread to handle TCP client connections. It receives data from the client, parses it as JSON, and checks if it contains temperature data. Checks if thee data is normal, if abnormal discard the value else insert into the databse.

The **handle\_udp\_client** function handles the UDP client’s communication. It runs in a separate thread to handle UDP client connections. It receives messages from the client, parses them as JSON, and checks if they contain light intensity data . Checks if thee data is normal, if abnormal discard the value else insert into the database.

The **handle\_publish** function is decorated with @socketio.on('publish') when the client publishes a message via the socket.io connection this function is called. It handles publishing MQTT messages based on the received JSON data. It queries the latest 10 data from each client and publishes it. The **handle\_mqtt\_message** function is decorated with @mqtt.on\_message() and is called when an MQTT message is received. It handles MQTT messages, check the data count of each sensor value and if there are 50 values of each sensor send email notification.

Threading is used which would allow multiple functions to run simultaneously, enabling efficient handling of different operations such as receiving data from TCP and UDP connections while running the Flask application. So, TCP thread and UDP thread is created which would start the execution of the function in new thread. **start\_background\_threads** function to start the TCP and UDP server threads. The main function starts the background threads and runs the Flask application using SocketIO to handle real-time communication between the server and clients.