1. **Device (ESP32) to MQTT Broker**
   * ESP32 reads sensor data (temperature, humidity, light)
   * Publishes to topics:
     + esp32/sensors for sensor readings
     + esp32/lights/log for device state changes
     + esp32/lights/control for receiving control commands
2. **Backend (Node.js) MQTT Flow**
   * Subscribes to esp32/sensors and esp32/lights/log
   * When messages arrive:
     + Parses JSON data
     + Stores in SQL Server database (sensors and devices tables)
   * Can publish control commands to esp32/lights/control
3. **Frontend (React) MQTT Flow**
   * Connects to same MQTT broker
   * Subscribes to same topics for real-time updates
   * Updates UI components:
     + SensorChart for visualizing sensor data
     + DeviceControl for device states
   * Sends control commands through:
     + MQTT: Direct device control
     + REST API: Database logging
4. **Database Flow**
   * Two main tables:
     + sensors: Stores temperature, humidity, light readings
     + devices: Stores device state changes
   * Backend provides REST API for:
     + Querying historical data
     + Filtering and pagination
     + Device control logging
5. **User Interaction Flow**
   * User toggles device → Frontend sends:
     + MQTT message for real-time control
     + API request to log change
   * User views data:
     + Real-time updates via MQTT
     + Historical data via REST API

This creates a full-circle flow where:

* ESP32 generates data
* MQTT enables real-time communication
* Backend handles persistence
* Frontend provides visualization and control

Here's a breakdown of how the LED state is maintained and synchronized across the system:

1. **ESP32 (.ino file):**
   * **Initial State:** The ledStates array in the .ino file stores the current state of each LED (1, 2, and 3, corresponding to "light", "fan", and "aircon"). Initially, all LEDs are set to LOW (off) in the setup() function.
   * **MQTT Control:** The callback() function is triggered when the ESP32 receives an MQTT message on the esp32/lights/control topic. This function parses the message to determine which LED to control and its desired state (on or off).
   * **State Update:** Inside the callback() function, the digitalWrite() function sets the physical LED pin to HIGH (on) or LOW (off) based on the received state. The ledStates array is then updated to reflect this new state.
   * **Logging:** After changing the LED state, the ESP32 publishes a log message to the esp32/lights/log topic with the new state and timestamp.
   * **Initial Log:** When the ESP32 connects to the MQTT broker, it publishes the initial state of each LED to the esp32/lights/log topic. This ensures that the backend and frontend are aware of the current LED states upon connection.
2. **Backend (be folder):**
   * **MQTT Handling:** The mqttHandler in be/config/mqtt.js connects to the MQTT broker and subscribes to the esp32/lights/log topic.
   * **State Logging:** When a message is received on the esp32/lights/log topic, the on("message") function parses the message, extracts the device name, state, and timestamp, and inserts this data into the devices table in the SQL Server database.
   * **Device Control:** The publishDeviceControl function in be/config/mqtt.js is used to send control commands to the ESP32 by publishing messages to the esp32/lights/control topic.
   * **API Endpoints:** The device.controller.js file defines API endpoints for retrieving device history (getAll), getting the current device status (getStatus), and toggling device states (toggle).
   * **Database Interaction:** The device.service.js file interacts with the database to fetch and update device information. The getRecentStatus function retrieves the most recent state of each device from the devices table. The toggle function publishes the device control message to MQTT.
3. **Frontend (fe folder):**
   * **Data Context:** The DataContext in fe/src/context/DataContext.tsx manages the application's state, including the state of the devices (devices object).
   * **Initial Data Fetch:** When the application loads, DataContext fetches the initial device status from the backend API endpoint /device/status. This endpoint returns the most recent state of each device from the database.
   * **MQTT Subscription:** The mqttService in fe/src/services/mqttService.ts connects to the MQTT broker and subscribes to the esp32/lights/log topic.
   * **State Updates:** When a message is received on the esp32/lights/log topic, the subscribeToDeviceData function parses the message, extracts the device name and state, and updates the devices state in the DataContext.
   * **Device Control:** The toggleDevice function in DataContext is called when the user toggles a device in the UI. This function updates the local state, publishes a control message to the backend API endpoint /device/toggle, and publishes the device control message to MQTT.
   * **Device Control:** The DeviceControl component in fe/src/components/devices/DeviceControl.tsx displays the current state of each device and allows the user to toggle the state.

**Initial Light State Dataflow Order:**

1. **ESP32 Initialization:** The ESP32 boots up and sets the initial state of the LEDs based on the ledStates array.
2. **MQTT Connection:** The ESP32 connects to the MQTT broker and publishes the initial LED states to the esp32/lights/log topic.
3. **Backend Update:** The backend receives the MQTT messages on the esp32/lights/log topic and updates the devices table in the database with the initial LED states.
4. **Frontend Fetch:** The frontend application loads and fetches the initial device status from the backend API endpoint /device/status.
5. **State Initialization:** The frontend updates its local state (devices in DataContext) with the initial device states received from the backend.

Therefore, the initial light state is received by the frontend **after** the ESP32 publishes it to MQTT and the backend updates the database. The frontend relies on the backend API to get the initial state.

**1. ESP32 (Microcontroller) Data Flow**

* **Sensor Data Acquisition:** The ESP32 reads data from DHT11 (temperature and humidity) and a light sensor.
* **Time Synchronization:** The ESP32 synchronizes its time with an NTP server to ensure accurate timestamps.
* **MQTT Publishing (Sensor Data):** The ESP32 publishes sensor data (temperature, humidity, light, timestamp) to the esp32/sensors MQTT topic.
* **MQTT Subscribing (Device Control):** The ESP32 subscribes to the esp32/lights/control MQTT topic to receive control commands for the LEDs (light, fan, aircon).
* **LED Control:** Based on the commands received via MQTT, the ESP32 turns the corresponding LEDs on or off. It also publishes log messages to the esp32/lights/log topic.
* **Alert System:** The ESP32 monitors sensor values and triggers alert LEDs if temperature, humidity, or light levels exceed predefined thresholds.

**2. Backend (Node.js) Data Flow**

* **MQTT Handling:** The backend connects to the MQTT broker and subscribes to the esp32/sensors and esp32/lights/log topics.
* **Data Processing:** When a message is received on these topics, the backend parses the JSON data.
* **Database Storage:** The backend stores the sensor data and device state changes in a SQL Server database. The sensors table stores temperature, humidity, light, and timestamp. The devices table stores device name, state ("on" or "off"), and timestamp.
* **API Endpoints:** The backend provides REST API endpoints for the frontend to retrieve data:
  + /device: Returns paginated device history data. Supports filtering by date, sorting, and pagination.
  + /device/status: Returns the current status (on/off) of the light, fan, and aircon.
  + /sensor: Returns paginated sensor data. Supports filtering by temperature, humidity, light, time, sorting, and pagination.
* **Device Control API:** The /device/toggle endpoint receives requests from the frontend to toggle the state of devices. It publishes the requested state to the esp32/lights/control MQTT topic.

**3. Frontend (React) Data Flow**

* **Initial Data Fetch:** When the application loads, the DataContext fetches the initial device status from the /device/status endpoint and the latest sensor data from the /sensor endpoint.
* **MQTT Connection:** The frontend connects to the MQTT broker to receive real-time updates.
* **Real-time Updates:** When the frontend receives a message on the esp32/sensors or esp32/lights/log topics, it updates the sensorData or deviceData state variables, respectively.
* **Device Control:** When the user toggles a device (light, fan, aircon), the toggleDevice function in DataContext is called. This function:
  + Updates the local devices state.
  + Publishes the new state to the /device/toggle backend endpoint.
  + Publishes the new state to the esp32/lights/control MQTT topic via the mqttService.
* **Data Display:** The frontend displays the sensor data in charts and tables, and the device status is shown using toggle switches.
* **Data Fetching and Display:** The DeviceHistoryPage and SensorDataPage components fetch data from the corresponding backend API endpoints and display it in tables using the DataTable component. They also handle pagination, sorting, and filtering.

**Data Flow Order and Initial Light State**

1. **Initial State:** The ESP32 code initializes the LEDs to a LOW state (off) at startup.
2. **Backend Initialization:** When the backend starts, it does *not* have any initial state for the lights.
3. **Frontend Initialization:** When the frontend starts, it fetches the device status from the /device/status endpoint.
4. **MQTT Publish:** The getStatus function in deviceService.js publishes the current status of the devices to the esp32/lights/control topic. This ensures that the ESP32 receives the correct state of the devices.
5. **ESP32 Receives State:** The ESP32 receives the device status from the MQTT broker and updates the LEDs accordingly.

Therefore, the initial light state data flow order is:

1. ESP32 initializes LEDs to OFF.
2. Frontend fetches device status from the backend /device/status endpoint.
3. Backend publishes the device status to the esp32/lights/control MQTT topic.
4. ESP32 receives the device status from MQTT and updates the LEDs.

