Factory Monitoring System: An Architectural Investigation using ESP32, MQTT, MongoDB, and Nuxt.js

1. Introduction

This report details the design and architectural considerations for a factory monitoring system leveraging ESP32 microcontroller nodes, MQTT messaging, MongoDB data storage, and a Nuxt.js web dashboard. The primary objective is to create a reliable and scalable system capable of collecting sensor data (temperature, voltage) from various points within a factory environment, ensuring data integrity even during network disruptions through local buffering on the ESP32 nodes, and providing real-time and historical data visualization via a web-based interface.

Key requirements include robust network connectivity using both WiFi and Ethernet (W5500), efficient local data storage on ESP32 flash memory (LittleFS) with mechanisms for managing storage limits and reliable synchronization upon reconnection, including confirmation of data upload before clearing local buffers. The system also explores the potential use of ESP-NOW for peer-to-peer communication and defines a data pipeline for ingesting MQTT messages into a MongoDB Time Series collection. The Nuxt.js frontend will feature an interactive factory layout map derived from an AutoCAD PDF export, real-time data updates via MQTT over WebSockets, and access to historical data through backend APIs. Critical architectural aspects such as reliability, scalability, and security are analyzed throughout the design.

2. ESP32 Node Design and Implementation

The ESP32 nodes form the core data acquisition layer of the system. Each node is responsible for reading sensor data, managing network connectivity, publishing data via MQTT, and buffering data locally when necessary.

2.1. Sensor Data Acquisition

Each ESP32 node will interface with sensors to measure critical factory parameters, specifically temperature and voltage. Standard libraries compatible with the chosen sensors (e.g., libraries for analog-to-digital converters for voltage, or specific libraries for temperature sensors like DHT, BME280 [1], or DS18B20 [3]) will be used within the Arduino IDE or ESP-IDF environment. The code will periodically sample these sensors at a configurable rate. The sampling rate is a critical parameter influencing network bandwidth usage and local storage requirements during offline periods.

2.2. Network Connectivity: WiFi and W5500 Ethernet

To enhance reliability in a potentially challenging factory RF environment, nodes will support dual connectivity: built-in WiFi and external W5500 Ethernet modules.

- **Libraries:**
  - **WiFi:** The standard WiFi.h library included with the ESP32 Arduino core will be used for WiFi connectivity.[1]
  - **Ethernet (W5500):** The standard Arduino Ethernet.h library has been updated to support the W5500 chip [7], offering functions like hardwareStatus() and linkStatus() for connection verification. Alternatively, older W5500-specific libraries like Ethernet2.h [8] or potentially more performant platform-specific drivers like ESP-IDF's esp_eth [9] could be considered if standard library performance proves insufficient. The W5500 communicates via SPI, requiring careful pin assignment to avoid conflicts, especially if other SPI peripherals are used.[10] Default SPI pins might need reconfiguration for the ESP32.[10] Performance with W5500 over SPI is typically in the range of 12-20 Mbps [9], which should be adequate for sensor data transmission but might be a bottleneck for higher bandwidth applications.
- **MAC Address:** For Ethernet connectivity, a unique MAC address is required for each W5500 module. To avoid manual configuration and potential conflicts on the LAN, the ESP32's unique WiFi MAC address can be programmatically retrieved using WiFi.macAddress() and assigned to the W5500 module, potentially modifying the last few bytes to ensure uniqueness if both interfaces need to be active simultaneously.[8]
- **Connection Management Logic:**
  1. **Initialization:** Attempt to initialize and connect using the preferred interface (e.g., Ethernet first).
  2. **Status Check:** Regularly check the status of the active connection (e.g., WiFi.status(), Ethernet.linkStatus()).
  3. **Failover:** If the active connection is lost, attempt to disconnect/de-initialize it and establish a connection using the secondary interface.
  4. **Failback (Optional):** Implement logic to periodically check if the preferred interface becomes available again and switch back if desired.
  5. **Network Status Indication:** Maintain an internal state variable indicating the currently active network interface (None, WiFi, Ethernet) for use by the MQTT publishing logic.

2.3. MQTT Publishing

Sensor data will be published to a central Mosquitto MQTT broker over the currently active network connection (WiFi or Ethernet).

- **Libraries:**
  - **Arduino IDE:** The PubSubClient library by Nick O'Leary is a common choice.[1] While widely used, it's noted to have potential compatibility nuances with ESP32 and primarily blocking operations.[1] Alternatives like AsyncMqttClient [2] or Espressif's native esp-mqtt library (via ESP-IDF or Arduino component [15]) offer asynchronous, non-blocking operations and potentially better TLS support, which are advantageous for responsiveness and handling concurrent tasks.
  - **MicroPython:** The umqtt.simple [17] or umqtt.robust libraries are standard choices.
- **Implementation (using PubSubClient):**
  1. **Include Libraries:** Include WiFi.h (or Ethernet library) and PubSubClient.h.[1]
  2. **Client Initialization:** Create WiFiClient (or EthernetClient) and PubSubClient instances.[1]
  3. **Set Server:** Configure the MQTT broker IP address and port (e.g., 1883 for TCP, 8883 for TLS) using client.setServer().[1]
  4. **Connect:** In a reconnect() function, attempt connection using client.connect(clientId, mqttUser, mqttPassword). Check client.connected() status in the main loop.[1] Handle connection failures and retries.[1]
  5. **Publish:** Periodically read sensor data, format it (e.g., as a JSON string or concise binary format), and publish using client.publish(topic, payload, [retained], [qos]).[1] Use specific topics (e.g., factory/floor1/sensorA/temperature).
  6. **Maintain Connection:** Call client.loop() frequently in the main loop to process incoming messages and maintain the connection.[1]

2.4. Local Data Buffering and Synchronization with Clearing

To prevent data loss during network or MQTT broker unavailability, sensor readings will be buffered locally on the ESP32's internal flash memory. LittleFS is chosen over SPIFFS due to its superior robustness against power failures and better performance characteristics, although it has a slightly higher overhead.[24]

- **2.4.1. Filesystem Choice: LittleFS:**
  - **Robustness:** LittleFS is designed to be power-loss resilient, minimizing the risk of filesystem corruption during writes, a critical feature for embedded systems.[24] SPIFFS is known to be more prone to corruption.[25]
  - **Performance:** LittleFS generally offers faster read/write performance compared to SPIFFS, especially for read operations.[24] SPIFFS performance can degrade significantly as the partition fills up.[29]

- **Features:** LittleFS supports directories and stores file metadata (like timestamps), unlike SPIFFS.[24]
- **Overhead:** LittleFS has a higher per-file overhead (minimum 4KB allocation unit) compared to SPIFFS (256 bytes).[26] Metadata pairs also add overhead.[27] This needs consideration during partition sizing.
- **Recommendation:** LittleFS is the preferred choice for this application due to its enhanced reliability and performance, despite the slightly higher overhead.[24]

- **2.4.2. Implementation Details:**
  - **Initialization:** Include FS.h and LittleFS.h. Initialize in setup() using LittleFS.begin(true), where true formats the partition if mounting fails.[32] Check the return value for success.[32]
  - **File Operations:** Use standard file operations provided by the File object obtained via LittleFS.open():
    - file.print(), file.println(), file.write(): To write data.[32]
    - file.read(), file.readBytes(), file.available(): To read data.[32]
    - LittleFS.remove(path): To delete files.[32]
    - LittleFS.open(path, FILE_APPEND): To append data.[32]
    - LittleFS.openNextFile(), file.isDirectory(), file.name(), file.size(): For listing directory contents.[32]
  - **Connection Loss Detection:** Monitor the MQTT client connection status using client.connected().[1] Additionally, monitor the underlying network status (WiFi.status(), Ethernet.linkStatus()). A lost MQTT connection while the network is active indicates a broker issue; a lost network connection implies both are unavailable.[15] The PubSubClient::loop() function helps maintain the connection but does not automatically re-establish it after a network drop; explicit reconnection logic is required.[22]
  - **Data Storage Format:**
    - **CSV:** Human-readable, easy to parse on the server. Example: timestamp,temperature,voltage\n. Overhead: ~20 bytes per reading (example).
    - **Binary:** More compact, reduces flash wear and storage space. Requires careful packing/unpacking on both ends. Example: Pack timestamp (uint32_t), temperature (float), voltage (float) directly. Overhead: 12 bytes per reading (example).
    - **Recommendation:** Use a **binary format** for efficiency. Define a consistent struct for sensor readings (including timestamp) and write the raw bytes of the struct to the buffer file.
  - **Flash Partition Size Calculation (36 Hours):**

1. **Data per Reading (Binary):** Timestamp (e.g., uint32_t = 4 bytes) + Temperature (float = 4 bytes) + Voltage (float = 4 bytes) = **12 bytes**.
2. **Sampling Rate:** Assume **1 reading per minute** (configurable). This needs to be defined based on factory requirements.
3. **Total Readings:** 36 hours * 60 minutes/hour = **2160 readings**.
4. **Raw Data Size:** 2160 readings * 12 bytes/reading = **25,920 bytes** (~25.3 KB).
5. **Filesystem Overhead:** LittleFS has overhead for metadata, block allocation (minimum 4KB unit [26]), and wear leveling.[27] A fixed overhead plus per-file/per-block overhead applies. Storing data in fewer, larger files is generally more space-efficient than many small files due to the 4KB minimum allocation.
6. **Required Partition Size:** A significantly larger partition is needed than the raw data size. Considering the overhead and the need for space for wear leveling operations, a partition size of **256 KB** or **512 KB** would be a reasonable starting point, providing ample buffer space and accommodating filesystem structures. Standard ESP32 modules typically have 4MB or more flash [36], allowing allocation of such partition sizes via the partition table (partitions.csv).

- **Storage Management (Circular Buffer/FIFO):**
  - **Strategy:** Implement a circular buffer *within one or a few files* rather than creating thousands of small files (due to LittleFS overhead).
  - **Approach:**
    1. Maintain metadata (e.g., in NVS using Preferences.h [39] or a separate small file) storing the current write position (byte offset) and the oldest data position within the buffer file(s).
    2. When writing new data: Append to the current write position. Increment the write position.
    3. Check if the write position exceeds the allocated buffer file size. If so, wrap around to the beginning of the file (write_position = 0).
    4. Before overwriting, check if the data being overwritten is the oldest data (compare write position with oldest data position). If overwriting un-uploaded data, this indicates the buffer is full.
    5. When data is successfully uploaded and acknowledged (see 2.4.5), update the oldest data position pointer to reflect the cleared space.
  - **Alternative (Multiple Files):** Use a fixed number of files (e.g., buffer_0.bin, buffer_1.bin,... buffer_N.bin). Write sequentially to files. When the last file is full, wrap around and start overwriting the first file (buffer_0.bin). Keep track of the oldest and newest file indices. This

simplifies pointer management but might incur slightly more overhead due to multiple file metadata.

- **2.4.5. Data Synchronization and Clearing Logic:**
  1. **Detect Reconnection:** When client.connected() becomes true after a period of being false (and WiFi.status() or Ethernet.linkStatus() is connected), initiate the synchronization process.
  2. **Read Buffered Data:** Open the buffer file(s) starting from the oldest known data position. Read data in manageable chunks (e.g., 10-50 readings per MQTT message) to avoid exceeding MQTT payload limits and overwhelming the network or broker.
  3. **Format for MQTT:** Format each chunk into the MQTT payload format (e.g., JSON array of readings or a custom binary format). Include timestamps and necessary metadata (sensorId, deviceMac). Add a flag or use a specific MQTT topic (e.g., factory/device_mac/buffered_data) to indicate this is backfilled data.
  4. **Publish with QoS 1:** Publish each chunk to the MQTT broker using **QoS 1.**[3] QoS 1 ensures the message is delivered *at least once* to the broker. The PubSubClient library handles the underlying PUBACK mechanism for QoS 1 internally when client.loop() is called.[14] While PubSubClient itself might not expose a direct callback for successful QoS 1 *publication* acknowledgment from the broker [45], relying on QoS 1 provides a reasonable guarantee the broker received the message.
  5. **Implement Application-Level Acknowledgement (Crucial for Clearing):** Since QoS 1 only confirms delivery to the broker, not processing by the backend, an application-level ACK is needed before safely clearing local data.
     - **Backend:** The MQTT-to-MongoDB pipeline (Section 4) subscribes to the buffered data topic. After successfully inserting a batch of data into MongoDB, it publishes an ACK message back to a device-specific topic (e.g., factory/device_mac/ack/buffered_data). The ACK payload should contain identifiers (e.g., a batch ID, or the timestamp range) of the data successfully stored.
     - **ESP32:** The ESP32 subscribes to its specific ACK topic (factory/device_mac/ack/buffered_data). Implement an MQTT message callback (client.setCallback()).[1]
  6. **Process ACKs and Clear Data:**
     - In the ESP32's MQTT callback, when an ACK message is received:
       - Parse the ACK payload to identify the successfully stored data batch/range.
       - Update the "oldest data position" pointer in the local buffer metadata

to reflect that this data range is now safe to overwrite.

  - **Physical Deletion (Optional but Recommended for Space Recovery):** While the circular buffer handles overwriting, if strict space management is needed or if the buffer doesn't fill completely between syncs, you could implement logic to periodically delete the oldest *acknowledged* file(s) (if using the multiple file approach) or potentially truncate/recreate the single buffer file after a full successful sync (more complex). For simplicity, relying on the circular buffer's overwrite mechanism combined with updating the oldest data pointer is often sufficient.

7. **Rate Limiting:** Implement delays between publishing buffered data chunks (e.g., delay(100); between publishes) to avoid flooding the network and broker, especially after a long disconnection.
8. **Continue Upload:** Repeat steps 2-7 until all buffered data (up to the current write position) has been published and acknowledged.
9. **Resume Live Publishing:** Once the buffer is empty (oldest position catches up to write position), resume publishing live sensor readings.

This robust buffering and synchronization mechanism, utilizing LittleFS for reliability and a QoS 1 + Application ACK handshake, ensures data integrity and prevents premature deletion of locally stored data.

3. ESP-NOW Peer-to-Peer Communication

ESP-NOW is a connectionless, low-power, 2.4 GHz protocol from Espressif enabling direct communication between ESP devices without needing a traditional Wi-Fi router or AP.[5]

3.1. Protocol Basics

- **Communication:** Fast, short packet transmission (up to 250 bytes).[6]
- **MAC Address:** Devices identify each other using their unique MAC addresses. The sender must know the receiver's MAC address.[5]
- **Pairing:** Before sending unicast data, devices must be paired using esp_now_add_peer() providing the peer's MAC address and channel.[5] A maximum of 20 peers can be registered for unencrypted communication, fewer for encrypted.[47]
- **Modes:** Supports one-way and two-way communication.[5] Can coexist with WiFi Station mode (using WiFi.mode(WIFI_AP_STA)) provided they operate on the same WiFi channel.[50]
- **Callbacks:** Uses callback functions (esp_now_register_send_cb,

esp_now_register_recv_cb) to handle send status notifications and received data.[5]

- **Range:** Practical range is typically less than 150m for stable communication, though up to 300-400m is possible in open environments under ideal conditions.[46] This is significantly less than Zigbee but potentially better than standard WiFi for simple packet exchange.[48]

3.2. Integration Strategy and Use Cases

Given the primary architecture relies on MQTT for scalability and centralized data collection, ESP-NOW's role will be supplementary, focusing on localized interactions where low latency or operation independent of the main network is beneficial.

- **Potential Use Cases:**
  - **Local Coordination:** Triggering an action on a nearby node (e.g., flash a warning light) based on a local sensor reading, without going through the MQTT broker.
  - **Data Relaying/Gateway:** Designating specific ESP32 nodes as ESP-NOW gateways. These nodes collect data from nearby, potentially battery-powered ESP-NOW-only sensors and then forward this aggregated data to the MQTT broker via their own WiFi/Ethernet connection.[50] This can extend network reach to areas with poor WiFi/Ethernet coverage or simplify the deployment of numerous simple sensors.
- **Gateway Implementation:**
  - An ESP32 gateway node needs to operate in WIFI_AP_STA mode to handle both ESP-NOW and WiFi/Ethernet concurrently.[51]
  - It must register ESP-NOW receive callbacks to capture data from sensor nodes.[5]
  - The received ESP-NOW data (within the callback or queued) needs to be formatted into MQTT messages (topic, payload).
  - The gateway maintains its connection to the MQTT broker (using WiFi/Ethernet) and publishes the formatted messages.[51]
  - The gateway needs to know the MAC addresses of the sensor nodes it serves, or sensors need to know the gateway's MAC address for sending.
- **Chosen Strategy:** Utilize ESP-NOW primarily for **data relaying via designated gateway nodes**. This leverages ESP-NOW's low-power potential for simple sensor nodes while bridging them to the main MQTT infrastructure. Direct peer-to-peer coordination will be secondary unless specific low-latency local control loops are identified.

4. MQTT-to-MongoDB Data Pipeline

This section details the process of subscribing to MQTT topics, processing the sensor data (including buffered data), and storing it persistently in MongoDB.

4.1. Technology Options

Several approaches exist for bridging MQTT data to MongoDB:

1. **Python Script (Paho-MQTT + Pymongo):** A common approach using mature libraries. Requires running a persistent Python process. paho-mqtt handles MQTT connection and subscription [55], while pymongo interacts with MongoDB.[57] Suitable for custom logic and flexibility.
2. **Node.js Script (mqtt.js + mongodb/mongoose):** Leverages JavaScript ecosystem. mqtt.js connects to MQTT [59], and mongodb [61] or mongoose [63] handles database interaction. Well-suited given the Nuxt.js frontend (shared language). Requires running a persistent Node.js process.
3. **Node-RED:** A visual flow-based programming tool. Provides nodes for MQTT input (mqtt in) [67] and MongoDB output (node-red-contrib-mongodb4 or similar).[68] Faster development for simple pipelines, potentially less flexible for complex transformations or error handling logic.[73]
4. **Broker Bridge/Plugin:** Some MQTT brokers offer built-in connectors or plugins to bridge data directly to databases like MongoDB. Examples include Pro Mosquitto's MongoDB Bridge plugin [60] or EMQX Data Integration features.[74] Simplifies architecture by eliminating a separate pipeline service but might require specific broker versions (often commercial) and offer less transformation flexibility than custom code.

Table 4.1: MQTT-to-MongoDB Pipeline Technology Comparison

Criteria	Python (Paho/Pymongo)	Node.js (mqtt.js/mongoose)	Node-RED	Broker Bridge (e.g., Pro Mosquitto/EMQX)
Ease of Setup	Medium	Medium	High	Medium (depends on broker/plugin)
Development Speed	Medium	Medium	High (for simple flows)	High (if direct mapping suffices)

Performance/Scalability	High	High	Medium (can bottleneck)	Potentially Very High (integrated)
Flexibility/Custom Logic	High	High	Medium (custom nodes)	Low to Medium (limited by plugin)
Maintenance Overhead	Medium (process mgmt)	Medium (process mgmt)	Low (visual interface)	Low (broker config)
Ecosystem/Libraries	Very Strong	Very Strong	Strong (Node-RED nodes)	Broker-Specific
Team Skillset Alignment	(Depends)	High (matches Nuxt.js)	(Visual)	(Broker config)
Handling Buffered Data ACK	High (custom logic)	High (custom logic)	Medium (requires logic)	Medium (depends on plugin features)

Recommendation: Given the Nuxt.js frontend, using **Node.js with mqtt.js and mongoose** offers a good balance of performance, flexibility for custom logic (like handling ACKs for buffered data), and aligns with the project's JavaScript ecosystem.

4.2. Implementation (Node.js + Mongoose)

A persistent Node.js script will perform the following:

1. **Dependencies:** Install mqtt and mongoose (npm install mqtt mongoose).
2. **MongoDB Connection:** Establish a persistent connection to MongoDB using Mongoose, preferably reusing the connection pool.[75] Use environment variables for the connection string.

```
JavaScript
// Simplified connection logic
import mongoose from 'mongoose';
import Reading from './models/Reading'; // Import schema from 4.3

const MONGO_URI = process.env.MONGO_URI |
| 'mongodb://localhost:27017/factory_monitoring';
const MQTT_BROKER = process.env.MQTT_BROKER |
| 'mqtt://localhost:1883';
const MQTT_TOPIC = process.env.MQTT_TOPIC |
| 'factory/#'; // Subscribe to all factory topics
```

```
mongoose.connect(MONGO_URI)
.then(() => console.log('MongoDB Connected'))
.catch(err => console.error('MongoDB Connection Error:', err));
```

3. **MQTT Client:**
   - **Connect:** Use mqtt.connect(MQTT_BROKER, options) to connect to the Mosquitto broker. Include credentials if required. Handle connection events (connect, error, close).
   - **Subscribe:** On successful connection (client.on('connect',...)), subscribe to the relevant factory topics using client.subscribe(MQTT_TOPIC, { qos: 1 }). QoS 1 ensures the pipeline receives messages at least once from the broker.
   - **Message Handling (client.on('message', async (topic, payload) => {... })):**
     - Receive message (topic, payload Buffer).
     - Parse payload: const messageString = payload.toString(); let data; try { data = JSON.parse(messageString); } catch (e) { /* Handle non-JSON or binary data */ }. Adapt parsing for the chosen binary format if used.
     - Identify Data Type: Check topic (e.g., /buffered_data suffix) or a field within data to determine if it's live or buffered.
     - Map to Schema: Transform the received data into the Mongoose Reading schema structure defined in 4.3. Ensure timestamps are correctly handled (convert from device format if needed). Set meta.dataType.
     - Insert into MongoDB: try { await Reading.create(mappedData); // Or insertMany for batches } catch (dbError) { console.error('DB Insert Error:', dbError); }.
     - **Buffered Data ACK:** If the data was identified as buffered and successfully inserted, publish an ACK message back to the specific device's ACK topic (e.g., factory/${deviceMac}/ack/buffered_data). The ACK payload should contain identifiers for the processed batch.

       ```
       JavaScript
       // Inside 'message' handler, after successful DB insert of buffered data
       ```

```
const ackTopic = `factory/${mappedData.meta.deviceMac}/ack/buffered_data`;
const ackPayload = JSON.stringify({ batchId: data.batchId }); // Example ACK payload
client.publish(ackTopic, ackPayload, { qos: 0 }); // QoS 0 for ACK is usually sufficient
```

   ○ **Error Handling:** Implement robust error handling for MQTT connection issues, message parsing failures, and database write errors. Consider retry mechanisms or dead-letter queues for persistent failures.

4.3. MongoDB Schema Design

The schema is designed to leverage MongoDB's Time Series collections for optimal storage and querying of sensor data.

- **Collection Strategy:** Use a single **Time Series Collection** for all sensor readings.[77] This approach benefits from optimized storage, indexing, and query performance tailored for time-stamped data.[78]
- **Schema Definition (Mongoose):**

```
JavaScript
import mongoose from 'mongoose';

const readingSchema = new mongoose.Schema({
  timestamp: { // The timeField (Required for Time Series)
    type: Date,
    required: true
  },
  meta: { // The metaField (Required for Time Series)
    sensorId: { type: String, required: true }, // e.g., 'TEMP_SENSOR_01'
    deviceMac: { type: String, required: true }, // Unique MAC of the ESP32 node
    location: { type: String }, // e.g., 'Floor1_MachineA_Inlet'
    dataType: { type: String, enum: ['live', 'buffered'], default: 'live' } // Identify backfilled data
  },
  // Measurement fields
  temperature: { type: Number },
  voltage: { type: Number }
  // Add other sensor fields as needed
}, {
  // Time Series Collection Options
  timeseries: {
    timeField: 'timestamp', // Specifies the date field
    metaField: 'meta',       // Specifies the metadata field
    granularity: 'minutes'  // Adjust based on data frequency ('seconds', 'minutes', 'hours') [79, 81]
  },
  // Optional: Automatically delete documents older than 1 year
  // expireAfterSeconds: 365 * 24 * 60 * 60 // [79]
});

// Indexing (MongoDB 6.3+ automatically creates a compound index on timeField & metaField [81])
// Explicitly create indexes on specific meta sub-fields if frequently used in queries
readingSchema.index({ 'meta.sensorId': 1, timestamp: -1 });
readingSchema.index({ 'meta.location': 1, timestamp: -1 });
readingSchema.index({ 'meta.dataType': 1, timestamp: -1 }); // Index for filtering live/buffered

const Reading = mongoose.model('Reading', readingSchema); // Mongoose creates
collection 'readings'
export default Reading;
```

- **Key Fields Explanation:**
  - timeField ('timestamp'): **Crucial.** Stores the exact time the sensor reading was taken on the device.[78] Must be a BSON Date.
  - metaField ('meta'): **Crucial.** Contains metadata identifying the data source.[78] Fields here should have relatively low cardinality (few unique values) and change infrequently. sensorId, deviceMac, and location fit this well. Including dataType allows distinguishing live vs. buffered data efficiently during queries. Poor metaField design (high cardinality) can negatively impact performance.[81] Queries should filter on sub-fields (meta.sensorId) rather than the entire meta object.[81]
  - granularity: Set based on the typical time between readings from a *single sensor* (e.g., 'seconds' if sampling every few seconds, 'minutes' if sampling every minute or few minutes).[79] This helps MongoDB optimize bucket creation and storage.[79]
  - Measurement Fields (temperature, voltage): Store the actual numeric sensor values. Rounding data to fewer decimal places can improve compression.[81]
- **Embedding vs. Referencing:** Time Series collections handle the "one-to-squillions" relationship (one device, many readings) efficiently.[83] The metaField acts like embedded metadata within the optimized time series structure. Separate device configuration data (if any) could be stored in a devices collection and referenced via deviceMac or a dedicated deviceId in the metaField, but avoid embedding large, frequently changing data in the metaField itself.[83]

This schema provides an efficient and query-optimized structure for storing the high-volume, time-stamped sensor data typical of IoT applications.

5. Nuxt.js Dashboard Development

The Nuxt.js dashboard serves as the user interface for visualizing real-time data, interacting with the factory layout, and accessing historical sensor information.

5.1. Factory Layout Visualization

Displaying the AutoCAD-exported PDF factory layout requires choosing between client-side rendering or server-side pre-conversion.

- **Option 1: Client-Side PDF Rendering:**
  - **Libraries:** Use pdf.js directly or via Vue wrappers like @tato30/vue-pdf [84] or vue-pdf-embed.[86] These libraries render the PDF within the browser.
  - **Pros:** No server-side conversion step needed. Potential for built-in PDF features like text selection (if rendered as text, not image).[86]
  - **Cons:** Performance issues with large/complex PDFs.[87] Requires client-side resources. Must be wrapped in <ClientOnly> or similar in Nuxt 3 to avoid SSR errors.[84] May require configuration for CMaps (character rendering) and Web Workers.[84] Rendering quality can vary (potential blurriness).[87] Interactivity (like overlaying a grid) needs to be built on top of the rendered output (likely a canvas).
- **Option 2: Server-Side Conversion to Web Format (SVG/Image):**
  - **Process:** Convert the PDF on the server (e.g., using a Nitro utility or a separate service) to either SVG or a high-resolution raster image (PNG/JPEG) before sending it to the client.
  - **SVG:** Scalable Vector Graphics. Ideal for line drawings like factory layouts.
    - *Pros:* Scales without quality loss [88], allows CSS styling and JavaScript interaction with individual elements [88], text can remain searchable/selectable [87], potentially smaller file size for vector content.[89] Excellent for interactive overlays.[90]
    - *Cons:* Conversion fidelity depends heavily on the tool used. Can become large/complex for very detailed source PDFs.[88] Potential minor browser rendering differences.[88]
  - **Image (PNG/JPEG):** Raster format.
    - *Pros:* Consistent rendering, simpler format.
    - *Cons:* Loses quality when zoomed, not inherently interactive, larger file size than SVG for diagrams.
  - **Conversion Tools (Node.js):** Finding robust, open-source, server-side

PDF-to-SVG libraries for Node.js can be challenging. Options include:

- Command-line wrappers: inkscape [92], pdf2svg.[92] Require installing these tools on the server.
- NPM Packages: Many are commercial cloud services (Aspose [94], ConvertAPI [97]), use client-side rendering (pdf-extractor wraps pdf.js [99]), or convert SVG *to* PDF (svg2pdf.js [100], svg-to-pdfkit [101]). BuildVu offers a server/cloud option but is commercial.[102]

- **Recommendation: Server-Side Conversion to SVG.** Despite the potential challenge in finding the ideal conversion tool, SVG offers superior advantages for this use case, primarily its scalability and inherent support for interactivity needed for the grid overlay.[88] Pre-conversion avoids client-side PDF rendering bottlenecks.[87] The resulting SVG can be displayed efficiently using standard HTML/Vue techniques. Evaluate Inkscape CLI or pdf2svg for conversion.

Table 5.1: PDF/SVG Display Technique Comparison

Criteria	Client-Side PDF Rendering (vue-pdf-embed/@tato30/vue-pdf)	Server-Side PDF-to-SVG	Server-Side PDF-to-Image
Performance (Initial Load)	Medium-Low (depends on PDF size/complexity) [87]	High (loads SVG/image)	High (loads image)
Performance (Interaction)	Medium (library dependent)	High (native SVG/DOM)	High (native image)
Interactivity Potential (Grid)	Low-Medium (overlay on canvas/DOM)	High (SVG elements)	Low (overlay on image)
Visual Fidelity (Scaling)	Medium (can be blurry) [87]	High (vector) [88]	Low (raster)
Implementation Complexity (Nuxt)	Medium (SSR handling, config) [84]	Medium (conversion setup)	Medium (conversion setup)
Server Load	Low (serves PDF)	Medium (one-time convert)	Medium (one-time convert)

Client Load	High (rendering)	Low (displaying SVG)	Low (displaying image)
Text Selectability/Searchability	Yes (if text layer enabled) [86]	Yes (if converted well)	No

5.2. Interactive Map Component Implementation (SVG Approach)

Assuming the factory layout is available as an SVG (factory_layout.svg):

1. **Component (InteractiveMap.vue):** Create a dedicated Vue component.
2. **Display SVG:**
   - Option A: Use <NuxtImg src="/path/to/factory_layout.svg" />.[104] Simpler, leverages Nuxt Image optimization if applicable.
   - Option B: Embed the SVG markup directly in the <template>. Allows easier binding of Vue directives (@click, :class) to specific SVG elements (e.g., <rect>, <path>).[105] **Recommended for interactivity.**
3. **Grid Overlay & Interaction:**
   - **Identify Grid Cells:** Ensure the SVG conversion process assigns unique IDs or classes to elements representing grid cells or clickable factory areas. If not, manually overlay a grid using SVG elements (<rect>) or absolutely positioned HTML <div> elements on top of the base SVG.
   - **Attach Click Listener:** Add a @click handler to the main SVG container or delegate events.[91]
   - **Get Click Coordinates:** Inside the click handler (handleClick(event)):
     - Get the SVG element reference.
     - Get the click coordinates relative to the viewport (event.clientX, event.clientY).
     - Create an SVG point: const pt = svgElement.createSVGPoint(); pt.x = event.clientX; pt.y = event.clientY;.[90]
     - Get the screen-to-SVG transformation matrix: const matrix = svgElement.getScreenCTM().inverse();.[90]
     - Transform the point: const svgPoint = pt.matrixTransform(matrix);.[90] svgPoint.x and svgPoint.y are now coordinates within the SVG's coordinate system.
   - **Map Coordinates to Grid Cell:** Calculate the grid cell row and column based on svgPoint.x, svgPoint.y, the SVG's viewBox dimensions, and the defined grid cell size.

     JavaScript
     ```
     // Example calculation (assuming grid starts at SVG 0,0)
     ```

```
const cellWidth = svgViewBoxWidth / numberOfColumns;
const cellHeight = svgViewBoxHeight / numberOfRows;
const col = Math.floor(svgPoint.x / cellWidth);
const row = Math.floor(svgPoint.y / cellHeight);
const cellId = `cell_${row}_${col}`;
```

   - **Highlight Cell (Optional):** Use Vue's reactive data to dynamically add a CSS class to the clicked SVG grid element for visual feedback.
   - **Trigger Modal:** Call a method to open the modal, passing the cellId or related information.

4. **Modal Component (SensorModal.vue):**
   - Use a pre-built component like UModal from Nuxt UI [109] or create a custom one.[110]
   - Accept the cellId or other data as a prop.
   - Inside the modal, display relevant information (e.g., cell identifier, current sensor readings for that location fetched via MQTT or API call, buttons for actions).

5.3. Real-Time Data Display

Real-time sensor readings will be displayed by connecting the Nuxt.js frontend directly to the Mosquitto broker using MQTT over WebSockets.

1. **Technology:** MQTT over WebSockets (WSS for security).[112]
2. **Library:** mqtt.js.[62]
3. **Implementation:**
   - **Client-Side Connection:** Establish the MQTT connection only on the client-side. Use a Nuxt plugin with client mode or manage within a component's onMounted hook.
   - **Connection Details:** Use mqtt.connect('wss://your_broker_domain:secure_port/mqtt', options).[62] The secure WebSocket port is often 8084 or 443.[4] Provide username, password, and clientId in the options if required by the broker.
   - **Subscription:** In the connect event handler, subscribe to relevant topics (e.g., factory/floor1/+/temperature, factory/floor1/+/voltage).
   - **Reactivity:** Use Vue's ref or reactive to create state variables for holding the latest sensor readings (e.g., const latestTemperatures = reactive({})).
   - **Message Handling:** In the message event handler (client.on('message', (topic, message) => {... })):
     - Parse the topic to identify the sensor/location.

        - Parse the payload (message.toString()) to get the value.
        - Update the corresponding reactive state variable (e.g., latestTemperatures[sensorId] = parsedValue).
    - **UI Binding:** Bind Vue components directly to these reactive variables. Changes will automatically update the UI.
      Code snippet
      ```
      <template>
        <div>Sensor {{ sensorId }}: {{ latestTemperatures[sensorId] |
      ```

| 'N/A' }} °C</div>
</template>
```

4. Visualization: Use libraries like ECharts 112 or Chart.js 112 wrapped in Vue components. Pass the reactive data variables as props to the chart components. Configure the charts to update dynamically when the props change. Be mindful of SSR compatibility with charting libraries; lazy loading or client-only rendering might be necessary.119

### 5.4. Historical Data Access

Historical data stored in MongoDB will be accessed via backend API endpoints created using Nuxt 3's Nitro server engine.

1. **Backend API Endpoints (server/api/):**
    - Create API routes, e.g., server/api/history.get.ts.
    - Use defineEventHandler to handle incoming GET requests.[64]
    - Access query parameters (e.g., sensorId, startTime, endTime, location) from the event object.
    - Import the Mongoose Reading model (from section 4.3).
    - Construct a MongoDB query using Mongoose methods based on the query parameters:
      TypeScript
      ```
      // server/api/history.get.ts
      import Reading from '~/server/models/Reading'; // Adjust path if needed
      import { getQuery } from 'h3';

      export default defineEventHandler(async (event) => {
        const queryParams = getQuery(event);
        const { sensorId, startTime, endTime, location } = queryParams;

        const filter: any = {};
        if (sensorId) filter['meta.sensorId'] = sensorId;
        if (location) filter['meta.location'] = location;
      ```
      ```

```
    if (startTime |


| endTime) {
filter.timestamp = {};
if (startTime) filter.timestamp.$gte = new Date(startTime);
if (endTime) filter.timestamp.$lte = new Date(endTime);
}




    try {
      const data = await Reading.find(filter)
                           .sort({ timestamp: 1 }) // Sort chronologically
                           .limit(1000); // Add limit for safety
      return data;
    } catch (error) {
      console.error("API Error fetching history:", error);
      throw createError({ statusCode: 500, statusMessage: 'Failed to fetch historical
data' });
    }
  });
  ```
```
*   Return the fetched data as JSON. Handle database errors appropriately.

2. **Frontend Fetching:**
   - In Vue components (e.g., HistoricalChart.vue or the page containing it), use Nuxt's data fetching composables.
   - Use useFetch or $fetch to call the Nitro API endpoint (/api/history).[122] Pass user-selected parameters (from date pickers, dropdowns) as query parameters.
   - Trigger the fetch when parameters change or on component mount (onMounted).

     Code snippet
     ```
     <script setup>
     import { ref, watch, computed } from 'vue';

     const selectedSensor = ref('TEMP_SENSOR_01');
     const selectedRange = ref({ start: new Date(), end: new Date() }); // From date
     ```
     ```

```
pickers

const queryParams = computed(() => ({
  sensorId: selectedSensor.value,
  startTime: selectedRange.value.start.toISOString(),
  endTime: selectedRange.value.end.toISOString(),
}));

// Use useFetch for automatic refetching when params change
const { data: historicalData, pending, error, refresh } = useFetch('/api/history',
{
  query: queryParams,
  watch: [queryParams] // Refetch when queryParams change
});

// Or use $fetch inside a function/watcher
// async function fetchData() {
//   historicalData.value = await $fetch('/api/history', { params:
queryParams.value });
// }
// watch(queryParams, fetchData, { immediate: true });
</script>
```

3. **Visualization:** Pass the fetched historicalData to chart components (ECharts, Chart.js) for display.

5.5. Frontend Structure and Components

Organize the Nuxt.js application for maintainability:

- **Pages (pages/):**
  - index.vue: Landing/entry page.
  - dashboard.vue: Main view with the interactive map and real-time widgets.
  - history.vue: View for exploring historical data with selectors and charts.
- **Components (components/):**
  - InteractiveMap.vue: SVG map rendering and interaction logic.
  - SensorModal.vue: Modal dialog triggered by map clicks.[109]
  - RealtimeDataWidget.vue: Displays a single real-time sensor value.
  - RealtimeChart.vue: ECharts/Chart.js component for live data streams.
  - HistoricalChart.vue: ECharts/Chart.js component for displaying fetched

    historical data.
  - DataSelector.vue: Component for selecting sensors, locations, date ranges.
- **Layouts (layouts/):**
  - default.vue: Common layout with navigation, header, footer.
- **Composables (composables/):**
  - useMqtt.js: Encapsulate MQTT connection and data handling logic (client-side).
  - useSensorData.js: Manage reactive sensor state.
- **Server Utilities (server/utils/):** Helper functions for API routes (e.g., database query helpers).
- **State Management:** Use useState for simple cross-component state (e.g., selected grid cell ID) if needed.[124] For more complex state, consider Pinia.

6. System Architecture Considerations

Designing a robust factory monitoring system requires careful consideration of reliability, scalability, and security across all components.

6.1. Reliability

Ensuring continuous operation and data integrity is paramount.

- **Node Level:**
  - *Network Redundancy:* Implementing both WiFi and W5500 Ethernet with automatic failover logic (Section 2.2) mitigates network connectivity issues specific to one medium.
  - *Data Buffering:* Using LittleFS for local data buffering (Section 2.4) prevents data loss when both network interfaces or the MQTT broker are unavailable.[24]
  - *Guaranteed Upload & Clearing:* The combination of MQTT QoS 1 publishing and an application-level acknowledgement (ACK) from the backend pipeline before clearing local data (Section 2.4.5) ensures that buffered data is confirmed as stored in MongoDB before being removed from the node's flash memory.[22] This prevents data loss if the backend fails after the broker receives the message but before it's stored.
- **Broker Level (Mosquitto):** For higher reliability, consider running Mosquitto in a high-availability configuration (e.g., using clustering features if available in specific distributions or manual setup with load balancers and shared state). Configure MQTT persistence (persistence true in mosquitto.conf) to store QoS 1/2 messages for offline clients.[61]
- **Data Pipeline (Node.js):** The pipeline script must handle errors gracefully (MQTT disconnects, database errors, message parsing errors). Running the script within

a process manager (like PM2) or container orchestrator (like Docker Swarm/Kubernetes) can ensure automatic restarts upon failure. Implement dead-letter queue logic for messages that repeatedly fail processing.
- **Database Level (MongoDB):** Deploy MongoDB as a replica set.[125] This provides automatic failover and data redundancy. Implement a robust backup strategy (e.g., mongodump, filesystem snapshots, MongoDB Atlas backups) with regular testing.[126]
- **Data Recovery Process:** If a node loses connection, it buffers data. Upon reconnection, it initiates the upload process (Section 2.4.5). Data is sent with QoS 1. The backend pipeline receives, stores in MongoDB (marking as 'buffered'), and sends an ACK per batch. The node receives the ACK, verifies it, and updates its internal pointer for the oldest acknowledged data, effectively clearing that space in the circular buffer for reuse.

6.2. Scalability

The system must accommodate potential growth in the number of sensors and data volume.

- **Nodes (ESP32):** Scaling is achieved by adding more ESP32 nodes. The primary limitation is the capacity of the downstream components (MQTT broker, pipeline, database). ESP-NOW, if used for gateways, has practical limits (<100 nodes per gateway) [47], reinforcing MQTT's role as the main data conduit.
- **MQTT Broker (Mosquitto):** Mosquitto scales well vertically (more CPU/RAM on the server). However, for very large deployments (hundreds of thousands or millions of devices), consider horizontally scalable, clustered brokers like EMQX [74], HiveMQ [125], or cloud-based IoT platforms (e.g., AWS IoT Core [127], Azure IoT Hub, Google Cloud IoT Core). Monitor broker connection limits and message throughput.
- **Data Pipeline (Node.js):** The Node.js script can be scaled vertically (more resources) or horizontally by running multiple instances. MQTT's publish/subscribe model naturally distributes messages if multiple pipeline instances subscribe to the same topics (though care must be taken with shared subscriptions or specific topic partitioning if needed).
- **Database (MongoDB):** MongoDB scales horizontally using sharding.[125] Time Series collections are specifically designed for high-volume, high-frequency IoT data ingestion and efficient querying.[77] Optimizing the schema (metaField cardinality, granularity) [79] and batching writes (insertMany) [81] are crucial for ingestion performance.
- **Dashboard (Nuxt.js):** The frontend application scales like a typical web

application using standard techniques (load balancing, CDNs). The Nuxt 3 Nitro backend API can be deployed as serverless functions or containerized services, allowing horizontal scaling independent of the frontend.

6.3. Security

Protecting the system from unauthorized access and data breaches is critical in an industrial setting. A multi-layered approach (defense-in-depth) is required.

- **Device Level (ESP32):**
  - *Firmware Protection:* Utilize Secure Boot and Flash Encryption if supported by the ESP32 hardware revision (e.g., ESP32 ECO V3 [128]) to prevent tampering and unauthorized firmware flashing.
  - *Credential Security:* Avoid hardcoding credentials in the firmware. Store WiFi and MQTT credentials securely, for example, in the Non-Volatile Storage (NVS) using the Preferences.h library [39] or by encrypting them within the LittleFS filesystem (requires an additional encryption library).
- **Network Level:**
  - *Wireless:* Enforce WPA2/WPA3 encryption for WiFi connections.
  - *Wired:* Implement physical security measures for Ethernet network access points.
  - *Segmentation:* Isolate IoT devices onto a dedicated VLAN or subnet. Use firewalls (e.g., iptables on Linux servers hosting the broker/DB [129]) to strictly control traffic between the IoT network and other corporate networks, allowing only necessary ports (e.g., MQTT TLS port 8883).[129]
- **Communication (MQTT):**
  - *Transport Encryption:* Mandate the use of TLS/SSL for all MQTT communication. Use port 8883 for MQTT over TLS, and WSS (WebSocket Secure, typically port 8084 or 443) for the dashboard connection.[4] ESP32 libraries support TLS, often via WiFiClientSecure.[4] This requires managing certificates (CA, potentially client certificates).
  - *Authentication:* Each ESP32 node must authenticate with the MQTT broker using strong, unique credentials (username/password).[4] Avoid shared credentials. Consider using client certificate authentication (mutual TLS - mTLS) for stronger device identity verification.[128]
  - *Authorization (ACLs):* Configure MQTT broker Access Control Lists (ACLs) to enforce the principle of least privilege.[130] Each device should only be permitted to publish to its designated topics (e.g., factory/floor1/sensorA/data, factory/floor1/sensorA/buffered_data) and subscribe only to necessary topics (e.g., its ACK topic factory/floor1/sensorA/ack/#). The backend pipeline needs

subscribe access to data topics and publish access to ACK topics. The dashboard needs subscribe access to live data topics.
    - *Topic Design:* Use specific, hierarchical topics and embed unique device identifiers (e.g., MAC address or assigned ID) to aid authorization and prevent accidental cross-talk.[138]
- **Backend Security:**
    - *Pipeline:* Securely store MongoDB credentials (use environment variables, secrets management). Sanitize and validate all data received via MQTT *before* inserting it into the database to prevent injection attacks.[139]
    - *MongoDB:* Follow MongoDB security best practices [126]:
        - Enable Authentication (security.authorization: enabled in config).[126]
        - Implement Role-Based Access Control (RBAC): Create specific roles for the pipeline service (write access to readings collection) and the Nuxt API backend (read access).[126]
        - Encrypt Data at Rest: Use MongoDB Enterprise/Atlas features or filesystem-level encryption (e.g., dm-crypt).[129]
        - Limit Network Exposure: Configure net.bindIp to bind only to trusted network interfaces; use firewalls.[129] Avoid default ports if possible.[129]
        - Enable Auditing: Track access and administrative actions.[126]
        - Regular Updates: Keep MongoDB patched and updated.[132]
        - Run as Dedicated User: Run mongod process under a non-privileged dedicated user account.[131]
- **Web Dashboard (Nuxt.js):**
    - *Transport Security:* Serve the dashboard exclusively over HTTPS.
    - *Authentication/Authorization:* Implement robust user login and role-based access control to restrict access to the dashboard and its features.
    - *Input/Output Validation:* Validate all user inputs and sanitize all data displayed from the API to prevent XSS attacks.[139]
    - *HTTP Security Headers:* Implement security headers like Content Security Policy (CSP), HTTP Strict Transport Security (HSTS), X-Frame-Options, etc. The nuxt-security module can automate much of this.[124]
    - *CSRF Protection:* If the dashboard includes forms that trigger state changes via POST requests, implement CSRF protection (e.g., using tokens provided by nuxt-security).[142]
    - *Dependency Management:* Regularly scan dependencies for vulnerabilities using tools like npm audit or Snyk [124] and keep them updated.
    - *OWASP Practices:* Adhere to OWASP Top 10 recommendations and checklists.[124]

Implementing security across all these layers is essential for protecting the integrity and confidentiality of the factory monitoring data and preventing unauthorized control of systems.

Table 6.1: Security Measures Checklist

Security Area	**Recommended Measure**	**Implementation Detail/Tool**	**Relevant Snippets**
Device Firmware	Secure Boot & Flash Encryption	ESP32 Hardware Feature (if available)	128
Device Credentials	Secure Storage (No Hardcoding)	NVS (Preferences.h) or Encrypted LittleFS	39
Network (WiFi)	Use WPA2/WPA3 Encryption	Standard WiFi Configuration	
Network (Ethernet)	Physical Network Security	Access Control, Secure Cabling	
Network (Segment)	Isolate IoT Network	VLANs, Firewalls (iptables, Security Groups)	129
MQTT Transport	Encrypt All Traffic (TLS/SSL/WSS)	Use Port 8883 (TLS) / 8084 (WSS), Configure Certificates	4
MQTT Authentication	Use Strong, Unique Credentials per Device	Username/Password or Client Certificates (mTLS)	4
MQTT Authorization	Enforce Least Privilege via ACLs	Configure Broker ACLs per device/topic	130
Data Pipeline	Secure DB Credentials, Validate/Sanitize	Environment Variables, Input	64

	Input	Validation Logic	
MongoDB Access	Enable Authentication & RBAC	security.authorization : enabled, Create specific roles	126
MongoDB Data	Encrypt Data at Rest	MongoDB Enterprise/Atlas Feature or Filesystem Encryption	129
Web Dashboard Auth	User Authentication & Authorization	Login System, Role Management (e.g., Nuxt Auth utilities)	
Web Dashboard I/O	Input Validation & Output Sanitization	Nuxt Server Validation, Frontend Sanitization	139
Web Headers	Implement HTTP Security Headers (CSP, HSTS, etc.)	nuxt-security Module or Manual Configuration	124
Dependencies	Regularly Scan & Update	npm audit, Snyk	124

7. Conclusion and Recommendations

This report outlines a comprehensive architecture for a factory monitoring system utilizing ESP32 nodes, MQTT, MongoDB, and a Nuxt.js dashboard. The design prioritizes reliability through dual network connectivity (WiFi/Ethernet) and robust local data buffering on the ESP32 using LittleFS, coupled with a reliable MQTT QoS 1 and application-level acknowledgement mechanism to ensure data is safely stored before being cleared from the device. ESP-NOW is proposed for localized data relaying via gateway nodes. A Node.js-based pipeline using mqtt.js and mongoose is recommended for transferring data from the Mosquitto MQTT broker to a MongoDB Time Series collection, leveraging its optimized storage and query capabilities for IoT data. The Nuxt.js dashboard employs server-side SVG conversion for displaying the factory layout interactively and utilizes MQTT over WebSockets for real-time data visualization, complemented by Nitro backend APIs for historical data access.

Key Findings:

- **Reliability:** The combination of LittleFS, MQTT QoS 1, and application-level ACKs provides a strong mechanism for data integrity during network outages. Careful implementation of the ACK handshake and buffer management is crucial.
- **Connectivity:** Dual WiFi/Ethernet offers valuable redundancy, but requires careful library selection and failover logic. W5500 performance over SPI is likely sufficient but should be verified.
- **Storage:** LittleFS offers better reliability than SPIFFS for local buffering, but its 4KB allocation overhead necessitates careful partition sizing. MongoDB Time Series collections are highly suitable for efficient long-term storage and querying of sensor data.
- **Interactivity:** Converting the PDF layout to SVG server-side provides the best foundation for the interactive grid overlay in the Nuxt.js dashboard due to SVG's scalability and element-level interactivity.
- **Security:** A multi-layered security approach encompassing device hardening, network segmentation, transport encryption (TLS/WSS), strong authentication (MQTT/DB/Web), and authorization (ACLs/RBAC) is essential for an industrial environment.

Actionable Recommendations:

1. **ESP32 Development:**
   - Prioritize implementation of the LittleFS buffering mechanism with the circular buffer logic.
   - Implement and thoroughly test the QoS 1 + Application ACK data synchronization and clearing protocol.
   - Select and benchmark Ethernet libraries (Ethernet.h vs. platform-specific) for performance and reliability with the W5500 module.
   - Develop and test the WiFi/Ethernet failover logic.
2. **Data Pipeline:**
   - Implement the Node.js pipeline using mqtt.js and mongoose.
   - Define the MongoDB Time Series schema carefully, selecting appropriate granularity based on the final sensor sampling rate.
   - Implement the ACK publishing logic within the pipeline.
3. **Nuxt.js Dashboard:**
   - Evaluate and select a server-side PDF-to-SVG conversion tool (Inkscape CLI or pdf2svg recommended for open-source). Test conversion fidelity with sample factory layouts.
   - Develop the InteractiveMap.vue component, focusing on robust SVG coordinate mapping for grid interaction.
   - Implement the client-side MQTT (WSS) connection using mqtt.js for real-time

updates.
   - Build Nitro API endpoints for historical data retrieval.
4. **ESP-NOW:** Prototype the ESP-NOW gateway functionality if required for specific sensor locations, ensuring stable coexistence with WiFi/Ethernet on the gateway node.
5. **Security Implementation:** Integrate security measures from the start, including TLS/WSS configuration for MQTT, broker ACLs, MongoDB authentication/RBAC, and web security headers (consider using nuxt-security).

Future Considerations:

- **Advanced Analytics:** Leverage MongoDB's aggregation framework and potentially integrate machine learning tools for predictive maintenance or anomaly detection based on historical data.
- **System Integration:** Explore integration with existing factory systems like Manufacturing Execution Systems (MES) or Enterprise Resource Planning (ERP) via APIs.
- **Scalability Testing:** Conduct load testing on the MQTT broker, data pipeline, and MongoDB instance as the number of nodes increases to identify potential bottlenecks.
- **Device Management:** Implement Over-The-Air (OTA) firmware update capabilities for the ESP32 nodes. Consider a device management platform for provisioning and monitoring nodes at scale.

Works cited

1. ESP32 MQTT Publish Subscribe with Arduino IDE - Random Nerd Tutorials, accessed April 14, 2025, https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/
2. ESP32 MQTT - Publish DHT11/DHT22 Temperature and Humidity Readings (Arduino IDE), accessed April 14, 2025, https://randomnerdtutorials.com/esp32-mqtt-publish-dht11-dht22-arduino/
3. ESP32 MQTT - Publish DS18B20 Temperature Readings (Arduino IDE), accessed April 14, 2025, https://randomnerdtutorials.com/esp32-mqtt-publish-ds18b20-temperature-arduino/
4. MQTT on ESP32: A Beginner's Guide - EMQX, accessed April 14, 2025, https://www.emqx.com/en/blog/esp32-connects-to-the-free-public-mqtt-broker
5. Getting Started with ESP-NOW (ESP32 with Arduino IDE) | Random ..., accessed April 14, 2025, https://randomnerdtutorials.com/esp-now-esp32-arduino-ide/
6. ESP-NOW Two-Way Communication Between ESP32 Boards - Random Nerd Tutorials, accessed April 14, 2025, https://randomnerdtutorials.com/esp-now-two-way-communication-esp32/

7. PSA: Arduino Ethernet library has been upgraded to support W5500 - Reddit, accessed April 14, 2025, https://www.reddit.com/r/arduino/comments/a8b92k/psa_arduino_ethernet_library_has_been_upgraded_to/
8. ESP32 W5500 Ethernet problem, accessed April 14, 2025, https://esp32.com/viewtopic.php?t=10827
9. ESP32 and W5500 ethernet performance, accessed April 14, 2025, https://esp32.com/viewtopic.php?t=37250
10. ESP32-DEV, W5500 Wired Ethernet Module, and BME-280 Web Server with Favicon!, accessed April 14, 2025, https://www.youtube.com/watch?v=L6G6XnFdoiw
11. Ethernet W5500 Module and ESP32Cam · jomjol AI-on-the-edge-device · Discussion #607, accessed April 14, 2025, https://github.com/jomjol/AI-on-the-edge-device/discussions/607
12. Esp32 with w5500 - slow speed - 3rd Party Boards - Arduino Forum, accessed April 14, 2025, https://forum.arduino.cc/t/esp32-with-w5500-slow-speed/1219476
13. ESP32 with W5500 MAC Address - Networking, Protocols, and Devices - Arduino Forum, accessed April 14, 2025, https://forum.arduino.cc/t/esp32-with-w5500-mac-address/1189574
14. Publish and Subscribe Data with Wemos D1 Mini Using MQTT Protocol, accessed April 14, 2025, https://gariskode.com/publish-and-subscribe-data-with-wemos-d1-mini-using-mqtt-protocol
15. ESP32 connection failed (rc = -2) after 20 minutes aprox and can´t reconnect. · Issue #947 · knolleary/pubsubclient - GitHub, accessed April 14, 2025, https://github.com/knolleary/pubsubclient/issues/947
16. MQTT Library for ESP32 with TLS, non-blocking - Programming - Arduino Forum, accessed April 14, 2025, https://forum.arduino.cc/t/mqtt-library-for-esp32-with-tls-non-blocking/1225427
17. MicroPython - Getting Started with MQTT on ESP32/ESP8266 ..., accessed April 14, 2025, https://randomnerdtutorials.com/micropython-mqtt-esp32-esp8266/
18. Intro to MicroPython on the ESP32: MQTT, accessed April 14, 2025, https://boneskull.com/micropython-on-esp32-part-2/
19. MicroPython: MQTT Publish DHT11/DHT22 with ESP32/ESP8266 | Random Nerd Tutorials, accessed April 14, 2025, https://randomnerdtutorials.com/micropython-mqtt-publish-dht11-dht22-esp32-esp8266/
20. MicroPython - MQTT Publish/Subscribe using ESP32/ESP8266 - YouTube, accessed April 14, 2025, https://www.youtube.com/watch?v=VmfnRMLFvE0
21. MicroPython - MQTT tutorial on ESP32 - YouTube, accessed April 14, 2025, https://www.youtube.com/watch?v=BkXWInr-KWM
22. Device does not maintain MQTT connection : r/esp8266 - Reddit, accessed April 14, 2025, https://www.reddit.com/r/esp8266/comments/1djtvom/device_does_not_maintain_mqtt_connection/

23. PubSubClient: Subscription working, but callback never called - Arduino Stack Exchange, accessed April 14, 2025, https://arduino.stackexchange.com/questions/80780/pubsubclient-subscription-working-but-callback-never-called
24. File management on ESP32: SPIFFS and LittleFS compared - Techrm, accessed April 14, 2025, https://www.techrm.com/file-management-on-esp32-spiffs-and-littlefs-compared/
25. Difference between spiffs and littleFs? : r/esp32 - Reddit, accessed April 14, 2025, https://www.reddit.com/r/esp32/comments/170l9po/difference_between_spiffs_and_littlefs/
26. RalphBacon/203-SPIFFS-vs-LITTLEFS: A simple file system for your ESP32 & ESP8266 - GitHub, accessed April 14, 2025, https://github.com/RalphBacon/203-SPIFFS-vs-LITTLEFS
27. littlefs-esp32/littlefs/DESIGN.md at master - GitHub, accessed April 14, 2025, https://github.com/LaurentLouf/littlefs-esp32/blob/master/littlefs/DESIGN.md
28. No more SPIFFS! LittleFS vs FatFs - YouTube, accessed April 14, 2025, https://m.youtube.com/watch?v=4pcUewJgPeM
29. File System Considerations - ESP32 - — ESP-IDF Programming Guide v5.4.1 documentation, accessed April 14, 2025, https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/file-system-considerations.html
30. Filesystem - ESP8266 Arduino Core documentation - Read the Docs, accessed April 14, 2025, https://arduino-esp8266.readthedocs.io/en/latest/filesystem.html
31. The design of the little filesystem - MCUXpresso SDK API Reference Manual, accessed April 14, 2025, https://mcuxpresso.nxp.com/api_doc/dev/1620/a00014.html
32. ESP32: Write Data to a File (LittleFS) - Arduino | Random Nerd ..., accessed April 14, 2025, https://randomnerdtutorials.com/esp32-write-data-littlefs-arduino/
33. ESP32: Upload Files to LittleFS using Arduino IDE - Random Nerd Tutorials, accessed April 14, 2025, https://randomnerdtutorials.com/esp32-littlefs-arduino-ide/
34. MQTT losing server connection - ESP32 Forum, accessed April 14, 2025, https://esp32.com/viewtopic.php?t=36555
35. MQTT Connection Lost : Connection issues when combining HTTP and MQTT calls in a loop, accessed April 14, 2025, https://stackoverflow.com/questions/77734870/mqtt-connection-lost-connection-issues-when-combining-http-and-mqtt-calls-in-a
36. How much flash memory does my ESP32 really have, accessed April 14, 2025, https://www.esp32.com/viewtopic.php?t=4086
37. ESP32-S3-WROOM-2 - Espressif Systems, accessed April 14, 2025, https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-2_datasheet_en.pdf
38. ESP32-WROOM-32E - FCC Report, accessed April 14, 2025, https://fcc.report/FCC-ID/2AC7Z-ESP32WROOM32E/4684794.pdf

39. ESP32 Save Data Permanently using Preferences Library - Random Nerd Tutorials, accessed April 14, 2025, https://randomnerdtutorials.com/esp32-save-data-permanently-preferences/
40. What is MQTT Quality of Service (QoS) 0,1, & 2? – MQTT Essentials: Part 6 - HiveMQ, accessed April 14, 2025, https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/
41. MQTT QoS 0, 1, 2 Explained: A Quickstart Guide | EMQ - EMQX, accessed April 14, 2025, https://www.emqx.com/en/blog/introduction-to-mqtt-qos
42. Build an effective mesh network with ESP32 and visualize data on Node-RED - Techrm, accessed April 14, 2025, https://www.techrm.com/build-an-effective-mesh-network-with-esp32-and-visualize-data-on-node-red/
43. MQTT QoS Guide - Quality of Service 0, 1, 2 Explained | Cedalo, accessed April 14, 2025, https://cedalo.com/blog/understanding-mqtt-qos/
44. Rapid Messages will Crash the Client? · Issue #55 · knolleary/pubsubclient - GitHub, accessed April 14, 2025, https://github.com/knolleary/pubsubclient/issues/55
45. PubSubClient mqtt how to ensure message is received/transmitted - Arduino Forum, accessed April 14, 2025, https://forum.arduino.cc/t/pubsubclient-mqtt-how-to-ensure-message-is-received-transmitted/671500
46. ESP NOW - Peer to Peer ESP32 Communications - DroneBot Workshop, accessed April 14, 2025, https://dronebotworkshop.com/esp-now/
47. ESP-NOW - - — ESP-FAQ latest documentation - Technical Documents, accessed April 14, 2025, https://docs.espressif.com/projects/esp-faq/en/latest/application-solution/esp-now.html
48. ESP-NOW and HaLow: good networking for IoT at last? - PU5EPX, accessed April 14, 2025, https://epxx.co/artigos/espnow_en.html
49. ESP-Now with ESP32C3 - Programming - Arduino Forum, accessed April 14, 2025, https://forum.arduino.cc/t/esp-now-with-esp32c3/1226946
50. ESP32: ESP-NOW and Wi-Fi Web Server Dashboard (Arduino) - Random Nerd Tutorials, accessed April 14, 2025, https://randomnerdtutorials.com/esp32-esp-now-wi-fi-web-server/
51. esp 32 with esp now and mqtt connection - arduino - Stack Overflow, accessed April 14, 2025, https://stackoverflow.com/questions/74957548/esp-32-with-esp-now-and-mqtt-connection
52. Comparison of Different Mesh Solutions - - — ESP-Techpedia latest ..., accessed April 14, 2025, https://docs.espressif.com/projects/esp-techpedia/en/latest/esp-friends/solution-introduction/mesh/mesh-comparison.html
53. Gateway ESPNOW to MQTT over WiFi - eMariete, accessed April 14, 2025, https://emariete.com/en/gateway-espnow-mqtt/

54. ESP-NOW with ESP32: Send Data to Multiple Boards (one-to-many), accessed April 14, 2025, https://randomnerdtutorials.com/esp-now-one-to-many-esp32-esp8266/
55. Paho MQTT Python client: a tutorial with examples | Cedalo blog, accessed April 14, 2025, https://cedalo.com/blog/configuring-paho-mqtt-python-client-with-examples/
56. How to Subscribe on Multiple topic using PAHO-MQTT on python - Stack Overflow, accessed April 14, 2025, https://stackoverflow.com/questions/48942538/how-to-subscribe-on-multiple-topic-using-paho-mqtt-on-python
57. Build A Python Database With MongoDB, accessed April 14, 2025, https://www.mongodb.com/resources/languages/python
58. PySide6 QTableView integration into GUI - Qt Forum, accessed April 14, 2025, https://forum.qt.io/topic/136266/pyside6-qtableview-integration-into-gui
59. Keeping MQTT Data History with Node.js - ReductStore, accessed April 14, 2025, https://www.reduct.store/blog/tutorials/iot/how-to-keep-mqtt-data-node
60. Integrating MQTT Data to MongoDB | Cedalo, accessed April 14, 2025, https://cedalo.com/blog/mqtt-to-mongodb-integration/
61. javascript - How to subscribe client and save the data to MongoDB ..., accessed April 14, 2025, https://stackoverflow.com/questions/42809581/how-to-subscribe-client-and-save-the-data-to-mongodb-in-node-js-mosca-and-mqtt
62. JavaScript MQTT Client: A Beginner's Guide to MQTT.js | EMQ - EMQX, accessed April 14, 2025, https://www.emqx.com/en/blog/mqtt-js-tutorial
63. How to use Nodejs as Backend in your Nuxtjs App: A Beginner Tutorial - Bukoye Olaniyi's Blog, accessed April 14, 2025, https://radiantcodes.hashnode.dev/how-to-use-nodejs-as-backend-in-your-nuxtjs-app-a-beginner-tutorial
64. Nuxt mongoose integration guide | Restackio, accessed April 14, 2025, https://www.restack.io/p/nuxt-mongoose-answer-integration-guide
65. Connect to 2 different mongodb databases in Nuxt 3 / Vue 3 Application - Stack Overflow, accessed April 14, 2025, https://stackoverflow.com/questions/75882757/connect-to-2-different-mongodb-databases-in-nuxt-3-vue-3-application
66. Using Vue.js & Nuxt.js with MongoDB & Docker | Metadrop, accessed April 14, 2025, https://metadrop.net/en/articles/using-vuejs-nuxtjs-mongodb-docker
67. Connect to an MQTT Broker - Node-RED Cookbook, accessed April 14, 2025, https://cookbook.nodered.org/mqtt/connect-to-broker
68. Using MongoDB With Node-RED • FlowFuse, accessed April 14, 2025, https://flowfuse.com/node-red/database/mongodb/
69. Node-Red + Sensors – Store data in MongoDB Database and exploring Thingspeak & MQTTool – Internet of Things - Leelu Chowdary Ravi, accessed April 14, 2025, https://leeluchowdaryravi.wordpress.com/2018/02/26/node-red-sensors-store-data-in-mongodb-database-and-exploring-thingspeak-mqttool/

70. node-red-node-mongodb, accessed April 14, 2025, https://flows.nodered.org/node/node-red-node-mongodb
71. NODE RED | GET DATA FROM MONGO DB AND DISPLAY ON HTML TABLE - YouTube, accessed April 14, 2025, https://www.youtube.com/watch?v=PHFuUUF2fuw
72. Node-RED Database Integration Guides - FlowFuse, accessed April 14, 2025, https://flowfuse.com/node-red/database/
73. (PDF) From Data to Decisions: A Smart IoT and Cloud Approach to ..., accessed April 14, 2025, https://www.researchgate.net/publication/388075621_From_Data_to_Decisions_A_Smart_IoT_and_Cloud_Approach_to_Environmental_Monitoring
74. MQTT to MongoDB: A Beginner's Guide for IoT Data Integration | EMQ - EMQX, accessed April 14, 2025, https://www.emqx.com/en/blog/mqtt-and-mongodb-crafting-seamless-synergy-for-iot-data-mangement
75. How to use collection in MQTT onMessage event - Node.js Frameworks - MongoDB, accessed April 14, 2025, https://www.mongodb.com/community/forums/t/how-to-use-collection-in-mqtt-onmessage-event/305457
76. Part 134 Getting data from MQTT storing into MongoDB in Node JS - YouTube, accessed April 14, 2025, https://www.youtube.com/watch?v=myMmEByT1gU
77. Optimizing Data Ingestion for High-Frequency IoT Sensors in MongoDB Time Series Database, accessed April 14, 2025, https://www.mongodb.com/community/forums/t/optimizing-data-ingestion-for-high-frequency-iot-sensors-in-mongodb-time-series-database/254983
78. IoT and MongoDB: Powering Time Series Analysis of Household ..., accessed April 14, 2025, https://www.mongodb.com/developer/products/atlas/iot-mongodb-powering-time-series-analysis-household-power-consumption/
79. Creating Time Series Collections in MongoDB - Simple Talk, accessed April 14, 2025, https://www.red-gate.com/simple-talk/featured/creating-time-series-collections-in-mongodb/
80. Mastering Time Series Data Management with MongoDB Time Series Collections, accessed April 14, 2025, https://embarkingonvoyage.com/blog/technologies/mongodb-time-series-collections/
81. Best Practices for Time Series Collections - Database Manual v7.0 ..., accessed April 14, 2025, https://www.mongodb.com/docs/v7.0/core/timeseries/timeseries-best-practices/
82. A production ready database schema for my iot data - MongoDB, accessed April 14, 2025, https://www.mongodb.com/community/forums/t/a-production-ready-database-schema-for-my-iot-data/230492
83. Data Modeling - Database Manual v8.0 - MongoDB Docs, accessed April 14,

2025,
https://www.mongodb.com/developer/products/mongodb/mongodb-schema-design-best-practices/
84. TaTo30/vue-pdf: PDF component for Vue 3 - GitHub, accessed April 14, 2025, https://github.com/TaTo30/vue-pdf
85. VuePDF - PDF Viewer Component - Made With Vue.js, accessed April 14, 2025, https://madewithvuejs.com/vuepdf
86. vue-pdf-embed - npm, accessed April 14, 2025, https://www.npmjs.com/package/vue-pdf-embed
87. Top JavaScript libraries for converting HTML to PDF - Nutrient, accessed April 14, 2025, https://www.nutrient.io/blog/html-to-pdf-in-javascript/
88. SVG Format: Features, Common Uses, and Pros/Cons You Should ..., accessed April 14, 2025, https://cloudinary.com/guides/image-formats/svg-format-features-common-uses-and-pros-cons-you-should-know
89. SVG vs PDF: what are advantages of SVG? - Everything Else - Glowforge Owners Forum, accessed April 14, 2025, https://community.glowforge.com/t/svg-vs-pdf-what-are-advantages-of-svg/33991
90. How to Translate from DOM to SVG Coordinates and Back Again - SitePoint, accessed April 14, 2025, https://www.sitepoint.com/how-to-translate-from-dom-to-svg-coordinates-and-back-again/
91. add mouse event to svg
92. About PDF to SVG converters - GitHub Gist, accessed April 14, 2025, https://gist.github.com/douglasmiranda/9c19f23c4570a7b7e02137791880ab43
93. designlook/node_inkscape_pdf2svg: Convert PDF to SVG files using Inkscape - GitHub, accessed April 14, 2025, https://github.com/designlook/node_inkscape_pdf2svg
94. Convert PDF to SVG in Node.js | products.aspose.com, accessed April 14, 2025, https://products.aspose.com/slides/nodejs-java/conversion/pdf-to-svg/
95. Javascript Convert PDF To SVG REST API - Aspose Cloud, accessed April 14, 2025, https://products.aspose.cloud/words/nodejs/conversion/pdf-to-svg/
96. Free online PDF to SVG conversion App via nodejs - Aspose Cloud, accessed April 14, 2025, https://products.aspose.cloud/total/nodejs/conversion/pdf-to-svg/
97. Convert PDF to SVG using Node.js - ConvertAPI, accessed April 14, 2025, https://www.convertapi.com/pdf-to-svg/nodejs
98. Convert PDF to SVG using JavaScript - ConvertAPI, accessed April 14, 2025, https://www.convertapi.com/pdf-to-svg/javascript
99. pdf-extractor - npm, accessed April 14, 2025, https://www.npmjs.com/package/pdf-extractor
100. yWorks/svg2pdf.js: A javascript-only SVG to PDF conversion utility that runs in the browser. Brought to you by yWorks - the diagramming experts - GitHub, accessed April 14, 2025, https://github.com/yWorks/svg2pdf.js/
101. svg-to-pdfkit - NPM, accessed April 14, 2025,

https://www.npmjs.com/package/svg-to-pdfkit
102. Convert PDF to SVG or HTML in Node.js with BuildVu - IDRsolutions, accessed April 14, 2025, https://www.idrsolutions.com/buildvu/convert-pdf-in-nodejs
103. @idrsolutions/buildvu - npm, accessed April 14, 2025, https://www.npmjs.com/package/@idrsolutions/buildvu?activeTab=readme
104.
105. Using SVG and Vue.js: A complete guide - LogRocket Blog, accessed April 14, 2025, https://blog.logrocket.com/using-svg-and-vue-js-a-complete-guide/
106. Using inline SVGs in Vue components - Caleb Porzio, accessed April 14, 2025, https://calebporzio.com/using-inline-svgs-in-vue-compoments/
107. How to get the click coordinates relative to SVG element holding the onclick listener?, accessed April 14, 2025, https://stackoverflow.com/questions/29261304/how-to-get-the-click-coordinates-relative-to-svg-element-holding-the-onclick-lis
108. Draggable SVG elements - Peter Collingridge, accessed April 14, 2025, https://www.petercollingridge.co.uk/tutorials/svg/interactive/dragging/
109. Modal Vue Component - Nuxt UI, accessed April 14, 2025, https://ui.nuxt.com/components/modal
110. timb-103/nuxt-modal: Custom modal component for Nuxt 3. - GitHub, accessed April 14, 2025, https://github.com/timb-103/nuxt-modal
111. Modal | Nuxt Black Dashboard @ Creative Tim, accessed April 14, 2025, https://www.creative-tim.com/learning-lab/nuxt/modal/black-dashboard
112. Adding real time using MQTT + Websockets | Ubidots Help Center, accessed April 14, 2025, https://help.ubidots.com/en/articles/9572884-adding-real-time-using-mqtt-websockets
113. What is MQTT over WebSockets and when is it used? - Engineers Garage, accessed April 14, 2025, https://www.engineersgarage.com/mqtt-over-websockets-arduino-iot/
114. A Quickstart Guide to Using MQTT over WebSocket | EMQ - EMQX, accessed April 14, 2025, https://www.emqx.com/en/blog/connect-to-mqtt-broker-with-websocket
115. mop/docs/using-mqtt-over-websocket.md at master - GitHub, accessed April 14, 2025, https://github.com/streamnative/mop/blob/master/docs/using-mqtt-over-websocket.md
116. Real-Time with Nuxt 3: A Guide to WebSocket Integration - Krutie Patel, accessed April 14, 2025, https://krutiepatel.com/blog/30-real-time-with-nuxt-3-a-guide-to-websocket-integration
117. Connect to MQTT Broker with Websocket - DEV Community, accessed April 14, 2025, https://dev.to/emqx/connect-to-mqtt-broker-with-websocket-14do
118. mqttjs/MQTT.js: The MQTT client for Node.js and the browser - GitHub, accessed April 14, 2025, https://github.com/mqttjs/mqtt.js/
119. Server-Side Rendering - Nuxt ECharts - Nuxt Module for Apache ECharts™,

accessed April 14, 2025, https://echarts.nuxt.dev/guides/ssr/
120. How To Build Realtime Charts With JavaScript and WebSocket - PieHost, accessed April 14, 2025, https://piehost.com/blog/building-realtime-charts-in-javascript
121. Lazy Hydration and Server Components in Nuxt - Vue.js 3 Performance, accessed April 14, 2025, https://vueschool.io/articles/vuejs-tutorials/lazy-hydration-and-server-components-in-nuxt-vue-js-3-performance/
122. NUXT 3 fetching data on the client side - vue.js - Stack Overflow, accessed April 14, 2025, https://stackoverflow.com/questions/75698949/nuxt-3-fetching-data-on-the-client-side
123. Data Fetching · Get Started with Nuxt, accessed April 14, 2025, https://nuxt.com/docs/getting-started/data-fetching
124. Good Practices - Nuxt Security, accessed April 14, 2025, https://nuxt-security.vercel.app/advanced/good-practices
125. Building a Reliable and Scalable IoT Platform - HiveMQ, accessed April 14, 2025, https://www.hivemq.com/blog/building-a-reliable-and-scalable-iot-platform/
126. Top 5 MongoDB Security Best Practices (+Checklist) - CloudDefense.AI, accessed April 14, 2025, https://www.clouddefense.ai/mongodb-security-best-practices/
127. MQTT over WebSocket - ThingsBoard, accessed April 14, 2025, https://thingsboard.io/docs/mqtt-broker/user-guide/mqtt-over-ws/
128. How do you secure your MQTT based ESP32? - Reddit, accessed April 14, 2025, https://www.reddit.com/r/esp32/comments/lia89w/how_do_you_secure_your_mqtt_based_esp32/
129. Best Practices for MongoDB Security | Severalnines, accessed April 14, 2025, https://severalnines.com/blog/best-practices-mongodb-security/
130. MQTT Security Best Practices - Cirrus Link Solutions, accessed April 14, 2025, https://cirrus-link.com/mqtt-security-best-practices/
131. Security Checklist - MongoDB Manual v7.2, accessed April 14, 2025, https://www.mongodb.com/zh-cn/docs/v6.2/administration/security-checklist/
132. Security Checklist for Self-Managed Deployments - Database Manual v8.0 - MongoDB Docs, accessed April 14, 2025, https://www.mongodb.com/docs/manual/administration/security-checklist/
133. Security Checklist - MongoDB Manual v4.4, accessed April 14, 2025, https://www.mongodb.com/docs/v4.4/administration/security-checklist/
134. Security Checklist — MongoDB Manual, accessed April 14, 2025, https://www.xuchao.org/docs/mongodb/administration/security-checklist.html
135. Ensuring MQTT Data Security - ESP32-C3 Wireless Adventure: A Comprehensive Guide to IoT, accessed April 14, 2025, https://espressif.github.io/esp32-c3-book-en/chapter_9/9.3/index.html
136. Enabling ESP32 MQTT: Ultimate Guide | Cedalo, accessed April 14, 2025,

https://cedalo.com/blog/enabling-esp32-mqtt/
137. MQTT safety in home network : r/esp32 - Reddit, accessed April 14, 2025, https://www.reddit.com/r/esp32/comments/ljlw9d/mqtt_safety_in_home_network/
138. MQTT Topics, Wildcards, & Best Practices – MQTT Essentials: Part 5 - HiveMQ, accessed April 14, 2025, https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/
139. Nodejs Security - OWASP Cheat Sheet Series, accessed April 14, 2025, https://cheatsheetseries.owasp.org/cheatsheets/Nodejs_Security_Cheat_Sheet.html
140. 11 MongoDB Security Features and Best Practices - Satori, accessed April 14, 2025, https://satoricyber.com/mongodb-security/11-mongodb-security-features-and-best-practices/
141. Adding security to Nuxt 3 - Snyk, accessed April 14, 2025, https://snyk.io/blog/adding-security-to-nuxt-3/
142. nuxt-security · Nuxt Modules, accessed April 14, 2025, https://nuxt.com/modules/security
143. Nuxt Security, accessed April 14, 2025, https://nuxt-security.vercel.app/
144. More secure Vue & Nuxt apps -> by default! 🛡️ - DEV Community, accessed April 14, 2025, https://dev.to/jacobandrewsky/more-secure-vue-nuxt-apps-by-default-3nhi
145. OWASP Web Application Security Testing Checklist - GitHub, accessed April 14, 2025, https://github.com/0xRadi/OWASP-Web-Checklist