ESP-CAM Monitoring Device

ARCHITECHTURE DOCUMENT

DEVELOPED BY - BRAINLYTREE

BrainlyTree.com

Contents

IoT Hospital Room Monitoring Architecture

1. Introduction

This document describes the architecture for a hospital room monitoring system. The system consists of ESP32-S3 based IoT devices equipped with BME680 environmental sensors and ESP CAM modules to capture images. Each device stores data locally on an SD card and communicates with a central Python server over Wi-Fi using MQTT protocol.

2. Requirements

Key requirements are:

• Capture environmental data (temperature, humidity, pressure, gas) via BME680. • Capture images via ESP32-CAM (OV2640 sensor).
• Data storage on local SD card (images and sensor data).
• Server requests images twice daily per device, with scheduling to avoid overload. • Device sends data in chunks with retry mechanism for missing payloads. • Device operates as a sleepy node to save power, waking only during server-defined windows.

• Communication via MQTT over Wi-Fi.
• Commissioning and onboarding via BLE (for Wi-Fi credentials and device registration).

3. Device Architecture

Each device is built on ESP32-S3 with camera and SD card module. It remains in deep sleep most of the time and wakes only when scheduled. The wake window is dynamically assigned by the server. The device workflow is as follows:

1. Wake from deep sleep (RTC timer).
2. Connect to Wi-Fi and MQTT broker.
3. Send HELLO message.
4. Listen for server commands (e.g., capture image, send sensor data, next wake time).
5. Capture sensor data and image, store on SD card.
6. Send data to server in chunks.
7. Resend missing chunks based on server ACK/NACK.
8. Receive next wake schedule, set RTC timer, and return to deep sleep.

4. Server Architecture

The server is implemented in Python, hosting both the MQTT broker and device management scripts. It maintains device states, schedules wake windows, and requests data from devices. It ensures no overload by staggering requests across 100+ devices.

Server components:

• Device Registry: maintains list of devices, credentials, last-seen, and next wake time. • Scheduler: calculates wake windows and distributes requests evenly. • MQTT Broker: handles publish/subscribe messaging between server and devices. • Data Collector: receives sensor/image payloads in chunks and validates CRC. • Retry Manager: requests retransmission of missing payload chunks.
• Storage Service: stores images and sensor data in central database or file system.

5. Communication Protocol & JSON Payload format

MQTT is used for messaging between devices and server. Example

topics: • device/{id}/status → Device HELLO messages (alive).

• MQTT JSON format for status message:

```
{
    "device_id": "esp32-cam-01",
    “status”: “alive”
```

“pendingImg”: 1

}

• device/{id}/cmd → Server commands (capture_image, set_next_wake).

• MQTT JSON format for capture image message:

{

"device_id": "esp32-cam-01",

“capture_image”

}

• MQTT JSON format for getting image message:

{

"device_id": "esp32-cam-01",

“send_image”: “image_001.jpg”

}

• MQTT JSON format for next wake up message:

{

"device_id": "esp32-cam-01",

“next_wake”: “wake_up_time”

}

• device/{id}/data → Device payload chunks (image + sensor).

• MQTT JSON format example for metadata:

{

"device_id": "esp32-cam-01",

"capture_timestamp": "2025-08-29T14:30:00Z",

“image_name”: “image_001.jpg”

“image_size”: 4153

“max_chunk_size”: 128

“total_chunks_count”: 15

“location”: <dev_location>

“error”: 0

"temperature": 25.5,

"humidity": 45.2,

"pressure": 1010.5,

"gas_resistance": 15.3

}

• MQTT JSON format example for chunk payload:

{

"device_id": "esp32-cam-01",
“image_name”: “image_001.jpg”
“chunk_id”: 1
“max_chunk_size:” 30
“payload”: {0xFF, 0xD8, 0xFF, 0xE0, ...}
}

• device/{id}/ack → Device ACKs for received commands or get missing chunks.

• MQTT JSON format example for asking retry of missing chunk:

{
"device_id": "esp32-cam-01",
“image_name”: “image_001.jpg”
“missing_chunks”: {5,10,23}
}

• MQTT JSON format example for asking retry of missing chunk:

{
"device_id": "esp32-cam-01",
“image_name”: “image_001.jpg”
“ACK_OK”:
{“next_wake_time”: “5:30PM”}
}

6. Reliability & Retry Mechanism

To ensure reliable transfer of large data (e.g., images) over potentially unstable Wi-Fi/MQTT networks, the system implements a chunk-based transmission protocol with verification and retry support.

• Chunked Data Transmission
  - Images are split into fixed-size chunks (chunk_id from 1…N).
  - Each chunk includes metadata (device_id, image_name, chunk_id, payload).
  - Device transmits chunks sequentially without waiting for individual acknowledgments.

• Server Verification
  - The server maintains a list of received chunk_ids for each image.
  - After all chunks are received (or a timeout expires), the server checks for missing chunks.

- Two possible outcomes:
    - Missing chunks found → server sends missing_chunks message with an array of missing chunk_ids.

    - All chunks received → server sends ACK_OK message confirming completion.

- Retry Mechanism
    - On receiving missing_chunks, the device retransmits only the requested chunks.
    - The server re-verifies until all chunks are received.
    - Maximum retry attempts per chunk shall be configurable (default = 3). If retries fail, server logs error and notifies operator.

7. Commissioning & Onboarding

Devices are commissioned via BLE. A mobile app provides Wi-Fi credentials and device ID. Once connected, the device registers with the server, receives its initial wake schedule, and transitions into sleepy operation.

8. Data Handling & Recovery Mechanism

8.1 Data Storage Architecture

Two files are maintained on the SD card:

- **metadata.txt**:
    - Stores full capture metadata including image name, capture timestamp, environmental data (temperature, humidity, pressure, gas resistance), and sync status (PendingToSend = true/false).
    - Acts as a historical log of all captured data.
- **pendingImage.txt**:
    - Contains only entries still awaiting upload.
    - Works as a transmission queue for recovery after offline periods.
    - Entries are removed once successfully acknowledged by the server.

8.2 Wi-Fi Retry Logic

- On wake-up, the device attempts to connect to Wi-Fi.
- Each attempt waits up to 8 seconds for connection.
- Retries up to 3 times (≈ 24s total).
- If unsuccessful → device proceeds in offline mode.

8.3 Online Operation (Wi-Fi Available)

• Device receives capture_image request from server.
• Captures image and sensor data.
• Stores them on SD card.
• Appends entry to both metadata.txt and pendingImage.txt (PendingToSend=true).

• Send metadata of current request.
• Device received send_image request from server.
• Transmits data to server in chunks.
• On ACK_OK from server:
    - Removes entry from pendingImage.txt.
    - Updates PendingToSend=false in metadata.txt.
    - next wake-up schedule (next_wake UTC) received in ACK packet.
• On MISSING_CHUNK from server:
    - Send remaining chunks to server.
• Enters deep sleep until scheduled wake-up.

8.4 Offline Operation (Wi-Fi Unavailable)

• At scheduled wake-up, the device still performs capture of image and sensor data. • Saves image + metadata to SD card.
• Appends entries to both metadata.txt and pendingImage.txt (PendingToSend=true). • Since Wi-Fi is unavailable, no transmission occurs.
• For timestamp:
    - Uses last known NTP time from last successful Wi-Fi sync.
    - Find Sleep duration based on Current time + on next_wake from last ACK.
    - Ensures continuity of timestamps without external RTC.
• Device enters deep sleep until the next scheduled wake-up.

8.5 Recovery & Synchronization

• When Wi-Fi is restored, device reads pendingfiles.txt and counts unsynced entries. • Includes this pending_count in the next alive message, e.g.:
{ "device_id": "dev_001", "alive": true, "pending_count": 5 }
• Server responds with sequential capture_image requests:
    - 1 request for the current capture.
    - N requests for the N pending entries.
• Requests are issued in looped sync fashion:
    - Device transmits → Server validates → Sends ACK_OK.
    - On ACK_OK, entry removed from pendingImage.txt and updated in metadata.txt.
    - Next request issued only after previous file is confirmed.
• This continues until all pending files are retransmitted.

8.6 Example Offline Scenario

- Device is offline for 5 days.
    - Each day, one capture is stored → 5 entries in both metadata.txt and pendingfiles.txt.
    - Upon Wi-Fi reconnection:
        - Device reports pending_count=5.

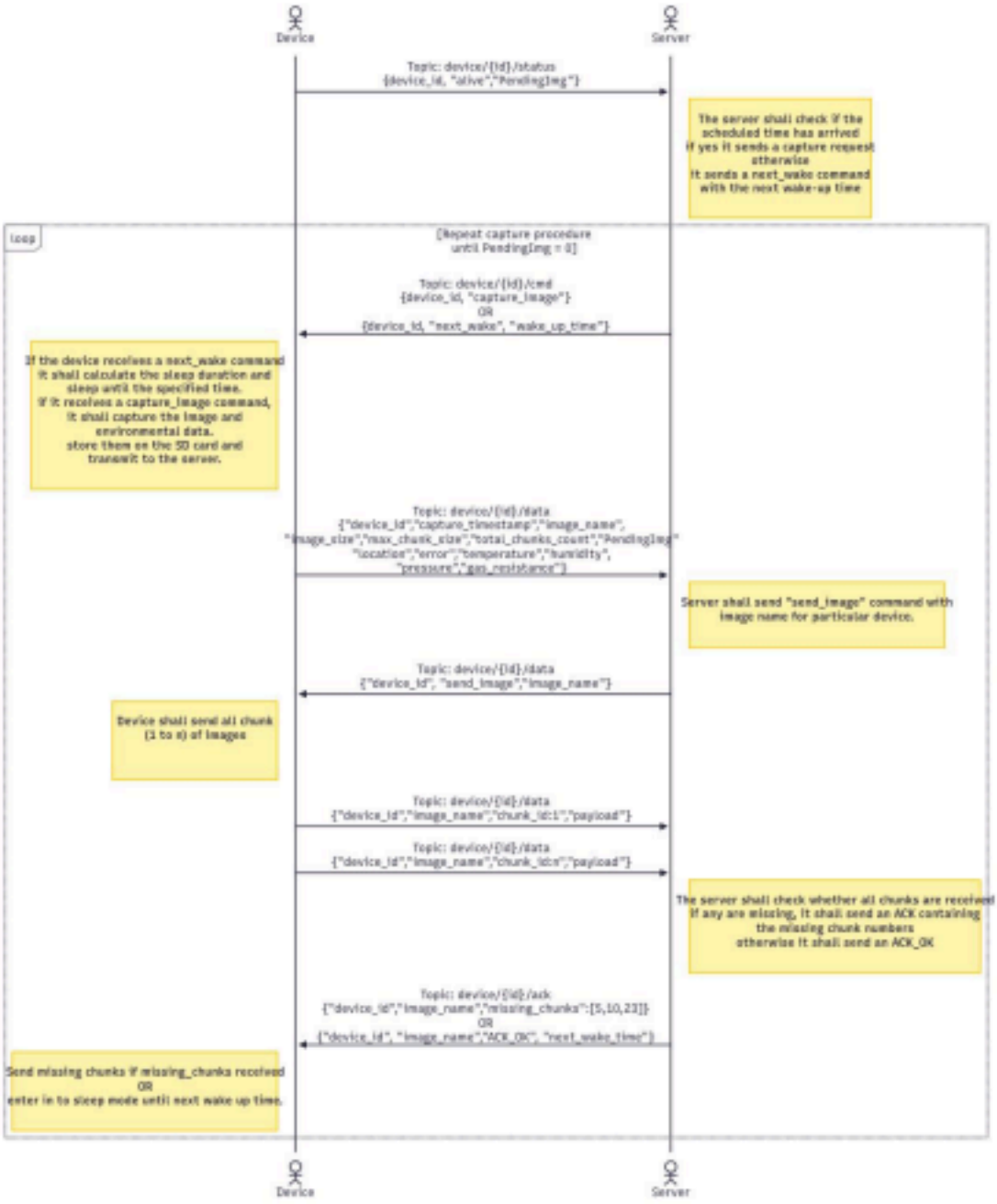
        - Server requests 6 files total (5 pending + 1 current).
        - After each file is acknowledged, device removes entry from pendingfiles.txt and updates metadata.txt.
        - Once complete, all captures show PendingToSend=false.

9. System Diagrams

- **State Diagram**

• **Flow Diagram**

Device-Server interaction (HELLO ⟶ CMD ⟶ Data Chunks ⟶ ACK/NACK ⟶ Next Wake).



10. Test Plan Strategy

10.1 Scop & Goal

Goal: verify offline data handling & recovery described in Section 8, plus chunk/retry and Wi-Fi retry/timestamp fallback behaviors. Tests will verify correctness of:

- SD metadata management: metadata.txt and pendingfiles.txt
- Data chunking / assembly
- Server missing-chunks detection & sending ACK_MISSING
- Device behavior on ACK_MISSING (resend only listed chunks)
- Device offline capture and pending queue behavior
- Device reporting pending_count in alive message
- Wi-Fi retry attempts (3 × 8s) logical flow
- Timestamp fallback when offline (last NTP + 24h per offline wake for image timestamp)

Tested components required:

- Device-side logic - file IO, queuing, chunk sending, retry behavior, timestamp calculation.
- Server-side chunk verification & missing-chunk logic (Python).
- Integration: device-server handshake and full retransmission cycle.

10.2 Test Strategy

- **Unit tests** - small modules in isolation using mocks:
    - sd_manager - file read/write/atomic update of metadata + pending queue
    - chunker - split/assemble of data
    - uploader (device side) - chunk send simulation and local retry queue logic
    - scheduler (device side) - next_wake handling
    - wifi_manager - retry attempt logic

- **Integration tests** - run device interacting with a mocked server:
    - Simulate offline period, create pending entries, simulate reconnection, ensure server requests and device retransmits and metadata/pending files updated.

- **End-to-end tests (HIL)** - run on device:
    - Flash firmware, disconnect Wi-Fi, capture images for N cycles, reconnect and verify server receives and verifies images and device metadata updated.

10.3 Test Cases

Each test case includes: ID, objective, steps, expected result.

Unit test cases (examples)

UT-01: **Chunk split & assemble**

• Objective: chunker splits binary into N chunks and reassembles to exact original • Steps: create bytearray file, chunk with size=16KB, compute base64 CRC for each chunk, reassemble
• Expect: reassembled bytes == original

UT-02: **Server detects missing chunks (Server Side)**

• Obj: Given chunk indices received set, server returns missing indices
• Steps: Provide total=10, received [1,2,3,5,6,7,8,9,10]
• Expect: missing [4]

UT-03: **pendingfiles.txt append / remove**

• Obj: Write metadata entry to metadata.txt and pendingfiles; then simulate ack -> removed/updated
• Expect: pendingfiles.txt entry removed; metadata entry updated with PendingToSend=false

UT-04: **Device retry on ACK_MISSING**

• Obj: Device resends only missing chunk ids
• Steps: simulate server responds with missing_chunks=[3,7] after initial send; device resends 3 & 7
• Expect: server receives chunk 3 and 7; image completes

UT-05: **Wi-Fi connection retries**

• Obj: wifi_manager attempts 3 × 8s, then reports offline
• Steps: simulate failing connect (return false), track attempts and timeouts • Expect: attempts==3; final status offline

Integration tests

IT-01: **Offline capture queue & recovery**

• Steps:
  - Simulate device capturing 3 images during offline mode → entries in both files
  - Reconnect → device sends alive with pending_count=3
  - Server issues 4 capture requests (1 current + 3 pending)

  - Verify for each: chunked upload, server validate & ACK_OKAY, pendingfiles entries removed, metadata updated
• Expect: server has 4 verified images, pendingfiles empty, metadata shows PendingToSend=false

IT-02: **Partial chunk loss & retransmit**

- Steps:
    - Device uploads 10 chunk image; server intentionally drops chunks [4,9]
    - Server issues ACK_MISSING for [4,9]
    - Device resends only [4,9]
    - Server reassembles successfully
- Expect: reassembled image exact

E2E / HIL tests (manual/hardware)

E2E-01: **5-day offline mode**

• Disconnect Wi-Fi; allow device to run 5 wake cycles storing images • Reconnect Wi-Fi; verify device reports pending_count=5; server requests retransmissions; all acknowledged and metadata updated

E2E-02: **Online mode- No retires**

- Connect Wi-Fi, MQTT.
- Send Alive message, receive capture_image from server.
- Send metadata, receive send_image from server.
- Send all chunks, receive ACK_OK with next_wake UTC time.
- Calculate time for sleep and configured timer.
- Enter in Sleep mode.

E2E-03: **Online mode – retries chunk**

- Connect Wi-Fi, MQTT.
- Send Alive message, receive capture_image from server.
- Send metadata, receive send_image from server.
- Send all chunks, receive ack with MISSING_CHUNKS.
- Send missing chunks, receive ACK_OK with next_wake UTC time. • Calculate time for sleep and configured timer.
- Enter in Sleep mode.

10.4 Pass / Fail Criteria

- Unit tests: pass if assertions succeed

• Integration tests: pass if server receives expected images & metadata updated

• E2E/HIL: pass if 0 data loss in offline scenario & timestamps preserved (or flagged if beyond tolerance)

• All test cases should be pass before signing off specially in E2E.

11. Connection Diagram