

DẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH



Đồ án liên ngành (CO4041 - CO4029)

BÁO CÁO DỰ ÁN:

Hệ thống quản lý chất lượng và độ tin cậy của dòng dữ liệu cảm biến trong nông nghiệp thông minh

GVHD:	PGS.TS. Thoại Nam
	ThS. Hoàng Lê Hải Thanh
	CN. Trần Nguyễn Minh Duy
Sinh viên:	Lê Hoàng Việt 2252903
	Ché Minh Đức 2210783
	Nguyễn Phú Quý 2212871

Mục lục

Danh sách hình vẽ	4
Danh sách bảng	4
Lời cảm ơn	5
Danh sách phân chia công việc	7
1 Giới thiệu	8
1.1 Bối cảnh và bài toán	8
1.2 Bài toán cốt lõi và tính liên ngành CS + CE	8
1.2.1 Định nghĩa bài toán cốt lõi	8
1.2.2 Phân tích: Tại sao bài toán đòi hỏi sự kết hợp CE và CS?	8
1.2.3 Sự kết hợp liên ngành CS + CE trong bài toán	11
1.3 Tính cấp thiết của đề tài	12
1.4 Tình hình nghiên cứu và các giải pháp hiện có	12
1.5 Mục tiêu và câu hỏi nghiên cứu	13
1.6 Phạm vi và giới hạn của đề tài	13
2 Cơ sở lý thuyết	14
2.1 Kiến trúc hệ thống và Công nghệ nền tảng	14
2.1.1 Kiến trúc tổng thể	14
2.1.2 Nền tảng ứng dụng	15
2.1.3 Hạ tầng Backend và Quy trình DevOps	15
2.2 Giao thức công nghiệp	16
2.2.1 Chuẩn vật lý RS485 (Physical Layer)	16
2.2.2 Giao thức Modbus RTU (Data Link Layer)	16
2.2.3 Lý do lựa chọn RS485/Modbus thay vì Analog	16
2.3 Bài toán phát hiện lỗi cảm biến trong hệ thống IoT	17
2.3.1 Lý thuyết về chất lượng dữ liệu cảm biến	17
2.3.1.a Lỗi trôi (Drift Fault)	17
2.3.1.b Lỗi Kẹt giá trị (Stuck-at Fault)	17
2.3.1.c Lỗi Gai nhiễu (Spike Fault)	18
2.3.2 Các phương pháp tiếp cận hiện có và hạn chế	18
2.3.3 Cơ sở lý thuyết của phương pháp Robust Feature Extractor (RFE)	18
2.3.4 Sơ đồ Luồng xử lý tổng thể	19
2.4 Edge Computing	21
2.4.1 Định nghĩa	21
2.4.2 Vai trò giảm tải cho Server	21
2.4.3 Mô hình xử lý dữ liệu tại biên trong đề tài	21
3 Khảo sát các giải pháp đã có	22
3.1 Khảo sát các ứng dụng, sản phẩm hoặc hệ thống liên quan	22
3.2 Khảo sát các mô hình, thuật toán, phương pháp	22
3.3 Bảng tổng hợp và phân tích khoảng trống	22
4 Phân tích yêu cầu	23
4.1 Mục đích	23
4.2 Phương pháp thu thập yêu cầu	23
4.3 Kết quả thu thập yêu cầu	23
4.4 Yêu cầu chức năng	23
4.5 Yêu cầu phi chức năng	24
4.6 Các mô hình phân tích UML	25
4.6.1 Use case: Giám sát Nông trại	25
4.6.2 Use case: Giám sát Thiết bị	29
4.6.3 Use case: Giám sát và Phân tích Hệ thống	32



4.6.4 Use case: Quản lý Báo cáo	34
5 Giải pháp đề xuất	36
5.1 Mục tiêu của giải pháp	36
5.1.1 Mục tiêu 1: Tự động hóa quá trình thu thập và xử lý dữ liệu môi trường	36
5.1.2 Mục tiêu 2: Phát hiện và cảnh báo sớm các sự cố hệ thống	36
5.1.3 Mục tiêu 3: Cung cấp giao diện quản lý trực quan và dễ sử dụng	36
5.1.4 Mục tiêu 4: Đảm bảo khả năng mở rộng và hiệu năng cao	36
5.1.5 Mục tiêu 5: Tối ưu hóa chi phí vận hành và bảo trì	36
5.1.6 Mục tiêu 6: Đảm bảo tính bảo mật và độ tin cậy	36
5.2 Thiết kế kiến trúc (Mức khái niệm)	37
5.2.1 Phân hệ Quản lý Nông trại:	37
5.2.2 Phân hệ IoT và Thu thập dữ liệu:	38
5.2.3 Phân hệ Người dùng và Phân quyền:	39
5.2.4 Phân hệ Kinh doanh	40
5.3 Thiết kế Kiến trúc Hệ thống	41
5.3.1 Kiến trúc Phân lớp (Layered Architecture)	41
5.3.2 Kiến trúc Triển khai (Deployment Architecture)	44
5.4 Phân tích đánh đổi kiến trúc và Quản lý rủi ro	45
5.5 Tại sao không chọn kiến trúc Monolithic đơn giản?	45
5.6 Dánh đổi kỹ thuật	46
5.7 Rủi ro và Chiến lược giảm thiểu	46
5.8 Đề xuất hiện thực và lựa chọn công nghệ	46
5.8.1 Quản lý lược đồ dữ liệu cảm biến từ xa	46
5.8.2 Đề xuất cải tiến cho thuật toán RFE	48
5.8.2.a Vấn đề và hạn chế của RFE nguyên bản	48
5.8.2.b Giải pháp đề xuất:	48
5.8.2.c Cơ chế hoạt động	49
5.8.2.d Kết luận	49
5.9 Thiết kế các thành phần chính	49
5.9.1 Thiết kế thành phần phần mềm: Quản lý cảm biến hướng dữ liệu	49
5.9.1.a Quản lý cảm biến hướng dữ liệu (Data-centric Sensor Management) . .	50
5.9.1.b Pipeline phát hiện và phân loại lỗi cảm biến (Sensor Fault Detection Pipeline)	51
5.9.1.c Quản lý vòng đời thiết bị dựa trên dữ liệu (Data-driven Lifecycle Management)	52
5.9.1.d Kiến trúc hệ thống và Giao diện quản trị (System Architecture & Administration Interface)	52
5.9.1.e Quy trình quản lý Farm:	54
5.9.1.f Quy trình quản lý thiết bị:	56
5.9.2 Firmware	58
5.9.2.a Activity Diagram	58
5.9.2.b Sequence Diagram	59
5.9.2.c Timing Diagram	61
5.9.2.d State Machine Diagram	64
5.10 Cách đánh giá giải pháp	65
5.10.1 Dánh giá Hiệu năng Hệ thống (Performance Evaluation)	65
5.10.2 Dánh giá Hiệu quả Thuật toán Phát hiện lỗi (Algorithmic Evaluation)	65
5.10.3 Dánh giá Trải nghiệm và Chức năng (Functional & UX Evaluation)	65
6 Kế hoạch thực hiện	66
6.1 Mục tiêu theo từng giai đoạn	66
6.1.0.a Firmware	66
6.1.0.b Software	66
6.2 Lịch trình và mốc thời gian	67
6.2.0.a Firmware	67
6.2.0.b Software	68



6.3	Rủi ro và phương án giảm thiểu	69
6.3.0.a	Firmware	69
6.3.0.b	Software	69
6.4	Gantt chart	70
7	Kết luận	71
7.1	Tóm tắt vấn đề và hướng tiếp cận	71
7.2	Công việc đã hoàn thành trong giai đoạn Đồ án Chuyên ngành	71
7.3	Định hướng thực hiện Đồ án Tốt nghiệp	71
7.4	Bài học kinh nghiệm và Góc nhìn phản tư	72
A	Sơ đồ Cơ sở dữ liệu vật lý (Physical ERD)	73
B	Danh sách các API chính (API Specifications)	73
C	Danh sách linh kiện phần cứng (Bill of Materials)	74
D	Bảng so sánh các giải pháp tham khảo	75
	Tài liệu tham khảo	76

Danh sách hình vẽ

1	Sơ đồ luồng xử lý sơ cấp ở Edge Node	19
2	Sơ đồ luồng xử lý phát hiện lỗi với RFE ở Backend Server	20
3	Sơ đồ quản lý farm	25
5	Sơ đồ luồng quản lý và tạo báo cáo	32
6	Sơ đồ luồng quản lý và tạo báo cáo	34
7	Sơ đồ thực thể quan hệ của phân hệ Quản lý Nông trại	38
8	Sơ đồ thực thể quan hệ của phân hệ IoT và Thu thập dữ liệu	39
9	Sơ đồ thực thể quan hệ của phân hệ Người dùng và Phân quyền	40
10	Sơ đồ thực thể quan hệ của phân hệ Kinh doanh	41
11	Sơ đồ Kiến trúc Phân lớp của Hệ thống	42
12	Deployment Diagram	44
13	Activity Diagram mô tả luồng quản lý farm	54
14	Activity Diagram mô tả luồng quản lý thiết bị	56
15	Activity Diagram	58
16	Sequence Diagram	59
17	Timing Diagram for sensor processing and control tasks	61
18	Timing Diagram for camera task	62
19	State Machine Diagram	64
20	Biểu đồ Gantt thể hiện tiến độ thực hiện đề tài	70
21	Sơ đồ thực thể chi tiết (Physical Data Model)	73
22	Dặc tả API Module Quản lý Nông trại (Farms Endpoint) [Nguồn: Swagger UI Hệ thống]	74
23	Dặc tả API Module Quản lý Thiết bị (Devices Endpoint) [Nguồn: Swagger UI Hệ thống] .	74

Danh sách bảng

1	Bảng phân chia công việc	7
2	So sánh cảm biến Analog và Modbus RS485 trong môi trường nông nghiệp	17
3	So sánh Cloud Computing và Edge Computing trong hệ thống IoT	21
4	Yêu cầu chức năng - Nhóm 1: Tổng quan Dashboard	23
5	Yêu cầu chức năng - Nhóm 2: Quản lý Nông trại và Người dùng	23
6	Yêu cầu chức năng - Nhóm 3: Quản lý và Giám sát Thiết bị	24
7	Tóm tắt phân loại ưu tiên các yêu cầu chức năng	24
8	Bảng phân tích đánh đổi khi lựa chọn công nghệ	46
9	Các kịch bản đánh giá thuật toán phát hiện lỗi	65
10	Bảng phân bố lịch trình thực hiện đồ án 15 tuần	67
12	Bảng phân tích rủi ro và phương án giảm thiểu	69
13	Phân tích rủi ro phần mềm và phương án xử lý	69
14	Bill of Materials (BOM)	75



Lời cảm ơn

Lời đầu tiên, nhóm chúng em xin gửi lời cảm ơn chân thành đến Ban Giám hiệu trường Đại học Bách Khoa - DHQG TP.HCM và quý Thầy Cô khoa Khoa học và Kỹ thuật Máy tính đã tạo điều kiện thuận lợi, cung cấp môi trường học tập và nghiên cứu hiện đại để chúng em có thể thực hiện đề tài này.

Đặc biệt, chúng em xin bày tỏ lòng biết ơn sâu sắc đến **PGS. TS. Thoại Nam, ThS. Hoàng Lê Hải Thanh** và **CN. Trần Nguyễn Minh Duy**. Các Thầy và Anh đã tận tình hướng dẫn, định hướng giải pháp và đưa ra những lời khuyên quý báu về mặt chuyên môn cũng như kỹ năng nghiên cứu trong suốt quá trình thực hiện Đề án chuyên ngành. Những kiến thức nền tảng và kinh nghiệm thực tế mà các Thầy chia sẻ chính là kim chỉ nam giúp nhóm vượt qua những khó khăn kỹ thuật để hoàn thiện hệ thống.

Chúng em cũng xin cảm ơn các bạn bè và anh chị đi trước đã nhiệt tình hỗ trợ, đóng góp ý kiến và chia sẻ tài liệu tham khảo hữu ích.

Mặc dù đã nỗ lực hết mình, nhưng do giới hạn về thời gian và kiến thức, đề án khó tránh khỏi những thiếu sót. Chúng em rất mong nhận được sự đóng góp ý kiến từ quý Thầy Cô và Hội đồng đánh giá để đề tài được hoàn thiện và phát triển tốt hơn trong tương lai.

Chúng em xin chân thành cảm ơn!

TP. Hồ Chí Minh, tháng 12 năm 2025
Nhóm sinh viên thực hiện



TÓM TẮT ĐỒ ÁN

Tên đề tài: Hệ thống quản lý nông trại thông minh

Tóm tắt:

Trong bối cảnh chuyển đổi số mạnh mẽ của ngành nông nghiệp Việt Nam, mô hình "Nông trại thông minh" đang trở thành xu hướng tất yếu để nâng cao năng suất và chất lượng nông sản. Tuy nhiên, việc triển khai thực tế đang đối mặt với thách thức lớn về sự phân mảnh của các thiết bị IoT đa nguồn gốc, độ tin cậy của dữ liệu cảm biến trong môi trường khắc nghiệt và rào cản kỹ thuật đối với những nhà quản lý không chuyên. Xuất phát từ thực tiễn đó, đề tài tập trung nghiên cứu và phát triển một hệ thống quản lý nông trại tập trung, nhằm cung cấp giải pháp vận hành thống nhất, đơn giản hóa trải nghiệm người dùng và đảm bảo tính toàn vẹn của dữ liệu giám sát.

Để giải quyết bài toán trên, nhóm nghiên cứu đề xuất xây dựng hệ thống dựa trên kiến trúc phân lớp (Layered Architecture) kết hợp với hướng dịch vụ, đảm bảo khả năng mở rộng linh hoạt. Về mặt công nghệ, hệ thống tích hợp các giải pháp tiên tiến như NestJS cho Backend, RabbitMQ để xử lý hàng đợi thông điệp hiệu năng cao và TimescaleDB tối ưu cho lưu trữ dữ liệu chuỗi thời gian lớn. Tại lớp thiết bị biên, vi điều khiển ESP32 được vận hành trên hệ điều hành thời gian thực FreeRTOS với cơ chế đa luồng, giúp tối ưu hóa tác vụ đọc cảm biến và truyền tải dữ liệu. Điểm nổi bật trong hướng tiếp cận của đề tài là việc tích hợp thuật toán trích xuất đặc trưng mạnh (Robust Feature Extractor - RFE) và các quy tắc thống kê, cho phép hệ thống tự động phát hiện sớm các lỗi cảm biến phổ biến như trôi số liệu (drift), mất kết nối hoặc giá trị bất thường.

Kết quả đạt được trong giai đoạn Đồ án chuyên ngành bao gồm việc hoàn thiện thiết kế kiến trúc tổng thể, xây dựng cơ sở dữ liệu vật lý chi tiết và đặc tả kỹ thuật cho các giao thức truyền thông lai (Hybrid Protocol MQTT/HTTP). Các phân tích và thiết kế này tạo nền tảng vững chắc cho giai đoạn hiện thực hóa sản phẩm, hướng tới mục tiêu triển khai một hệ thống thực tế có khả năng chịu tải trên 50 thiết bị đồng thời với độ trễ thấp và độ ổn định cao. Đề tài không chỉ mang ý nghĩa khoa học trong việc ứng dụng thuật toán phát hiện lỗi tại biên, mà còn có giá trị thực tiễn cao, góp phần tháo gỡ điểm nghẽn về công nghệ cho các mô hình kinh doanh nông nghiệp vừa và nhỏ.

Từ khóa: Nông nghiệp thông minh, Quản lý tập trung, IoT, Phát hiện lỗi cảm biến.



Danh sách phân chia công việc

STT	Họ tên	MSSV	Vai trò	Distribution
1	Lê Hoàng Việt	2252903	Backend	100%
2	Ché Minh Đức	2210783	Frontend	100%
3	Nguyễn Phú Quý	2212871	Firmware	100%

Bảng 1: Bảng phân chia công việc



1 Giới thiệu

1.1 Bối cảnh và bài toán

Trong xu thế nông nghiệp hiện đại, mô hình "Nông trại thông minh" (Smart Farm) đang trở thành tiêu chuẩn để đảm bảo năng suất và chất lượng nông sản thông qua việc ứng dụng công nghệ IoT và AI [3].

Tuy nhiên, thị trường hiện nay đang hình thành một nhóm đối tượng quản lý mới: các thương lái hoặc nhà đầu tư sở hữu nhiều nông trại (hoặc thuê lại để kinh doanh) nhưng không có chuyên môn sâu về công nghệ thông tin.

Mô hình kinh doanh của họ thường bao gồm việc quản lý chuỗi các nông trại phân tán hoặc cho khách hàng thuê ngắn hạn để trải nghiệm và canh tác. Từ thực tế này, bài toán đặt ra bao gồm:

- **Sự phân mảnh thiết bị:** Các nông trại này thường tích hợp thiết bị IoT từ nhiều nhà cung cấp khác nhau, dẫn đến việc thiếu đồng bộ về giao thức và khó khăn trong quản lý tập trung [12].
- **Rào cản kỹ thuật:** Người dùng (thương lái) gặp khó khăn khi phải thao tác trên nhiều ứng dụng rời rạc hoặc tự mình xử lý các sự cố kỹ thuật phức tạp.
- **Yêu cầu về minh bạch chất lượng:** Khi kinh doanh dịch vụ, họ cần một công cụ tin cậy để chứng minh chất lượng môi trường canh tác (nhiệt độ, độ ẩm ổn định) cho khách hàng.

1.2 Bài toán cốt lõi và tính liên ngành CS + CE

1.2.1 Định nghĩa bài toán cốt lõi

Bài toán cốt lõi của đồ án này được định nghĩa là: "**Thách thức về đảm bảo độ tin cậy dữ liệu trong môi trường phân tán với tài nguyên hạn chế.**"

Cụ thể, bài toán không chỉ đơn giản là thu thập dữ liệu (CE) hay hiển thị dữ liệu (CS), mà là bài toán **đảm bảo tính toàn vẹn của luồng dữ liệu (Data Integrity Pipeline)** từ vật lý đến đám mây trong điều kiện mạng và môi trường không ổn định. Trong môi trường nông nghiệp thực tế, các cảm biến IoT phải hoạt động trong điều kiện khắc nghiệt (nhiệt độ cao, độ ẩm cao, bụi bẩn, dao động nguồn điện), dẫn đến các thách thức:

- **Dữ liệu nhiễu và không ổn định:** Cảm biến có thể gặp nhiễu do môi trường, dẫn đến các giá trị bất thường (outliers) không phản ánh đúng trạng thái thực tế.
- **Mất kết nối mạng:** Thiết bị IoT thường hoạt động ở các vùng nông thôn với kết nối mạng không ổn định, dễ bị gián đoạn.
- **Trôi số liệu (Sensor Drift):** Cảm biến có thể bị trôi theo thời gian do lão hóa hoặc điều kiện môi trường, dẫn đến dữ liệu sai lệch dần.
- **Hạn chế tài nguyên:** Thiết bị IoT (ESP32) có RAM/CPU hạn chế, không thể xử lý các thuật toán phức tạp hoặc lưu trữ lịch sử dài hạn.
- **Quy mô phân tán:** Hệ thống cần quản lý hàng trăm node IoT phân bố trên nhiều nông trại địa lý khác nhau.

Giải pháp cho bài toán này không thể chỉ dừng lại ở việc "thu thập và hiển thị", mà phải xây dựng một **pipeline xử lý dữ liệu đa tầng** đảm bảo chất lượng dữ liệu ở mọi giai đoạn: từ thu thập tại edge (CE), truyền tải qua mạng, đến lưu trữ và phân tích tại cloud (CS).

1.2.2 Phân tích: Tại sao bài toán đòi hỏi sự kết hợp CE và CS?

Dựa trên phân tích các thách thức nêu trên, bài toán này **đòi hỏi sự kết hợp liên ngành giữa Khoa học Máy tính (Computer Science - CS) và Kỹ thuật Máy tính (Computer Engineering - CE)** để giải quyết hiệu quả. Việc chỉ dựa vào một trong hai lĩnh vực sẽ gặp những hạn chế đáng kể. Phân tích cụ thể như sau:



Tại sao CS làm một mình không được? Giả sử chỉ xây dựng một hệ thống phần mềm mạnh mẽ trên máy chủ (Backend NestJS với kiến trúc microservices, thuật toán Machine Learning phức tạp như Random Forest, Isolation Forest, cơ sở dữ liệu TimescaleDB tối ưu) mà không có phần cứng và firmware tương ứng được thiết kế cẩn thận:

- **Chất lượng dữ liệu đầu vào (Data Quality):** Theo nguyên lý "Garbage In, Garbage Out" trong khoa học dữ liệu, hiệu quả của các thuật toán phân tích phụ thuộc trực tiếp vào chất lượng dữ liệu đầu vào. Trong môi trường nông nghiệp khắc nghiệt, cảm biến có thể gặp các vấn đề sau:
 - Nhiêu điện từ (EMI) do máy móc nông nghiệp, dẫn đến các giá trị bất thường không có ý nghĩa.
 - Mất kết nối tạm thời, khiến dữ liệu bị mất hoặc đến server không đồng bộ.
 - Trôi số liệu (sensor drift) do cảm biến bị lão hóa hoặc ảnh hưởng bởi môi trường (nhiệt độ, độ ẩm cao), dẫn đến dữ liệu sai lệch dần theo thời gian mà không dễ nhận biết.
 - Lỗi phần cứng (hardware failure) như cảm biến bị chết hoặc kết nối lỏng lẻo, tạo ra dữ liệu cố định (frozen values) hoặc giá trị null liên tục.
- **CS cần CE để đảm bảo chất lượng nguồn tin ngay tại biên (Edge):** Để giải quyết vấn đề trên, hệ thống cần xử lý dữ liệu ngay tại edge (trên thiết bị IoT) trước khi gửi về server:
 - **Firmware phải có cơ chế lọc nhiễu cơ bản:** Xử lý tín hiệu analog, áp dụng bộ lọc số (digital filter), loại bỏ giá trị ngoại lai rõ ràng (obvious outliers) trước khi truyền tải.
 - **Firmware phải có cơ chế phát hiện lỗi hardware-level:** Phát hiện cảm biến không phản hồi (I2C/SPI timeout), kiểm tra tính hợp lệ của dữ liệu (range checking, sanity check) ngay trên thiết bị.
 - **Firmware phải đảm bảo tính nhất quán dữ liệu:** Thêm timestamp chính xác, sequence number, checksum để đảm bảo dữ liệu không bị mất hoặc đảo thứ tự khi truyền qua mạng không ổn định.
 - **Firmware phải có cơ chế buffering và retry:** Khi mất kết nối, firmware phải lưu dữ liệu tạm thời và gửi lại khi kết nối khôi phục, đảm bảo không mất dữ liệu quan trọng.
- Nếu thiếu các cơ chế trên ở tầng CE, hệ thống CS sẽ phải xử lý một lượng lớn dữ liệu chất lượng thấp, dẫn đến các hệ quả:
 - Tăng tải tính toán không cần thiết trên server để xử lý và làm sạch dữ liệu không hợp lệ.
 - Tăng độ trễ trong việc phát hiện lỗi do phải thu thập đủ dữ liệu để phân tích pattern bất thường.
 - Giảm độ chính xác của thuật toán phát hiện lỗi do tỷ lệ false positive/false negative cao, làm giảm độ tin cậy của hệ thống.
- **CS không thể tích hợp thiết bị đa nguồn hiệu quả:** Vấn đề phân mảnh thiết bị IoT từ nhiều nhà cung cấp khác nhau (như thực tế tại Tomochan Farm với các cảm biến SHTC3, ES-ALS-02, ES-RAIN-02, và camera Hikvision) đòi hỏi phải có một lớp chuẩn hóa ở mức firmware và phần cứng. Mỗi thiết bị có thể sử dụng giao thức khác nhau (WiFi, RS485 Modbus RTU, IP camera protocol), định dạng dữ liệu khác nhau, và cơ chế bảo lối khác nhau. Nếu không có firmware tùy chỉnh trên ESP-32 và Raspberry Pi để chuẩn hóa giao tiếp, hệ thống CS sẽ phải xử lý quá nhiều edge cases và khó bảo trì.
- **CS không thể điều khiển thiết bị vật lý:** Để thực hiện các hành động khắc phục tự động (như bật quạt khi nhiệt độ cao, bơm nước khi độ ẩm đất thấp), hệ thống cần firmware trên ESP32 để điều khiển relay, bật/tắt thiết bị chấp hành. CS chỉ có thể gửi lệnh, nhưng việc thực thi phải được thực hiện ở tầng CE.

Kết luận: Hiệu quả của hệ thống phần mềm (Backend, thuật toán ML) phụ thuộc đáng kể vào chất lượng dữ liệu đầu vào từ thiết bị vật lý. CS cần CE để đảm bảo chất lượng dữ liệu ngay tại biên (Edge), tạo nền tảng cho việc phân tích ở tầng cloud. Nhiều nghiên cứu đã chỉ ra rằng xử lý dữ liệu tại edge giúp giảm tải mạng và cải thiện độ tin cậy của hệ thống IoT [1].



Tại sao CE làm một mình không được? Giả sử chỉ phát triển phần cứng và firmware tốt cho các node cảm biến (ESP32-S3 với firmware tối ưu, cảm biến chất lượng cao, xử lý tín hiệu tốt) mà không có hệ thống phần mềm phía server được thiết kế để chịu tải và xử lý dữ liệu quy mô lớn:

- **Giới hạn về tài nguyên tính toán và lưu trữ:** Thiết bị ESP32-S3 có các đặc tính kỹ thuật sau [20]:

- RAM: 512KB (hoặc 8MB với PSRAM), đủ cho xử lý real-time nhưng hạn chế cho lưu trữ lịch sử dữ liệu dài hạn (ví dụ: 30 ngày với tần suất 5 giây/lần \approx 518,400 bản ghi, mỗi bản ghi khoảng 50-100 bytes cần khoảng 25-50MB).
- CPU: Dual-core 240MHz, phù hợp cho xử lý real-time và các thuật toán đơn giản, nhưng không đủ để chạy các mô hình Machine Learning phức tạp như Random Forest hoặc Isolation Forest (thường yêu cầu hàng nghìn phép tính và dung lượng bộ nhớ lớn cho cây quyết định).
- Flash: Dung lượng hạn chế (thường 4-16MB), không đủ để lưu trữ mô hình ML lớn hoặc dataset training đầy đủ.

Do các ràng buộc tài nguyên này, firmware trên ESP32 chủ yếu có thể:

- Thực hiện phát hiện lỗi cơ bản ở mức phần cứng (hardware failure detection, threshold checking).
 - Khó khăn trong việc phát hiện các lỗi phức tạp như sensor drift (đòi hỏi so sánh với xu hướng lịch sử dài hạn) hoặc anomaly detection dựa trên pattern (cần phân tích tương quan giữa nhiều cảm biến và dữ liệu lịch sử).
 - Không thể thực hiện learning từ dữ liệu lịch sử do hạn chế về dung lượng lưu trữ và khả năng tính toán.
- **Không thể mở rộng cho hàng trăm node:** Nếu không có kiến trúc phần mềm phía server được thiết kế để chịu tải:
 - **Thiếu Message Queue (RabbitMQ):** Khi có hàng trăm thiết bị gửi dữ liệu đồng thời (mỗi thiết bị 5 giây/lần), hệ thống sẽ bị quá tải. Message queue giúp buffer dữ liệu, decoupling giữa producer (thiết bị) và consumer (xử lý), đảm bảo không mất dữ liệu khi server tạm thời quá tải.
 - **Thiếu Time-series Database (TimescaleDB):** Lưu trữ và truy vấn hàng triệu bản ghi dữ liệu cảm biến đòi hỏi cơ sở dữ liệu chuyên dụng cho time-series data. TimescaleDB, một extension của PostgreSQL được tối ưu hóa cho time-series, có hiệu năng truy vấn tốt hơn đáng kể so với PostgreSQL thông thường khi xử lý dữ liệu chuỗi thời gian [9, 2].
 - **Thiếu kiến trúc Microservices:** Xử lý dữ liệu từ hàng trăm thiết bị đòi hỏi khả năng scale độc lập từng module. Module xử lý dữ liệu (Worker) cần scale theo số lượng thiết bị, trong khi Web API chỉ cần scale theo số lượng người dùng. Kiến trúc monolithic không thể đáp ứng yêu cầu này.

- **Không thể phát hiện các mẫu lỗi phức tạp:** Các lỗi cảm biến thông minh đòi hỏi phân tích ở tầng cao hơn:

- **Sensor Drift Detection:** Cần so sánh dữ liệu hiện tại với xu hướng lịch sử dài hạn (30-90 ngày) để phát hiện cảm biến bị trôi dần. Firmware trên ESP32 không thể lưu trữ đủ dữ liệu lịch sử để thực hiện phân tích này.
- **Anomaly Detection dựa trên Correlation:** Một cảm biến nhiệt độ có thể báo giá trị bình thường, nhưng nếu so sánh với cảm biến độ ẩm và ánh sáng cùng khu vực, có thể phát hiện được sự bất thường (ví dụ: nhiệt độ cao nhưng độ ẩm không giảm như dự kiến \rightarrow có thể cảm biến nhiệt độ bị lỗi). Phân tích correlation này cần dữ liệu từ nhiều cảm biến, không thể thực hiện trên một node IoT đơn lẻ.
- **Learning từ dữ liệu lịch sử:** Thuật toán RFE (Recursive Feature Elimination) hoặc các mô hình ML khác cần được train trên dataset lớn để học pattern bất thường. Quá trình training này đòi hỏi sức mạnh tính toán của server, không thể thực hiện trên ESP32.



- **Không thể quản lý tập trung đa nông trại:** Vấn đề quản lý chuỗi nông trại phân tán (bài toán cốt lõi) không thể giải quyết nếu chỉ có các node IoT độc lập. Cần hệ thống phần mềm với:
 - Cơ sở dữ liệu tập trung để tổng hợp dữ liệu từ hàng trăm thiết bị thuộc nhiều nông trại.
 - Giao diện web để admin quản lý, so sánh hiệu suất giữa các nông trại, và đưa ra quyết định dựa trên dữ liệu tổng hợp.
 - Cơ chế phân quyền (RBAC) để người dùng chỉ xem được dữ liệu nông trại của mình.
- **Không có cơ chế cảnh báo và tương tác với người dùng:** Người dùng (thương lái/chủ vườn) không thể tương tác trực tiếp với firmware của ESP32. Họ cần:
 - Giao diện web trực quan để xem dashboard, nhận cảnh báo real-time.
 - Email/notification khi phát hiện lỗi nghiêm trọng.
 - Khả năng điều khiển thiết bị từ xa qua web interface.

Dây là các chức năng thuộc về CS (phát triển web application, email service, notification system).

Kết luận: Phần cứng IoT (ESP32, cảm biến) có giới hạn về tài nguyên tính toán và lưu trữ do yêu cầu về kích thước, giá thành và tiêu thụ năng lượng. Việc thiếu kiến trúc phần mềm phía server được thiết kế để chịu tải (message queue, time-series database, microservices) và các thuật toán hậu xử lý (như RFE trên server) sẽ hạn chế khả năng mở rộng hệ thống cho quy mô lớn (hàng trăm node) và khả năng phát hiện các mẫu lỗi phức tạp như sensor drift hay anomaly detection dựa trên correlation giữa nhiều cảm biến.

1.2.3 Sơ kết hợp liên ngành CS + CE trong bài toán

Bài toán này yêu cầu một **kiến trúc phân tầng** nối CE và CS bỗng lẩn nhau:

- **Tầng CE (Edge Layer - Phần cứng và Firmware):**
 - **Tích hợp và chuẩn hóa đa giao thức:** Thiết kế và phát triển firmware trên ESP-32 và Raspberry Pi để tích hợp các thiết bị công nghiệp đa dạng tại Tomochan Farm (cảm biến WiFi SHTC3, cảm biến RS485 Modbus RTU như ES-ALS-02, ES-RAIN-02, EC & TDS, và camera IP Hikvision), tạo ra một lớp trừu tượng hóa (abstraction layer) để chuẩn hóa dữ liệu trước khi gửi về server.
 - **Quản lý tài nguyên và điều phối:** Firmware trên ESP-32 quản lý việc đọc dữ liệu song song từ nhiều cảm biến với các giao tiếp khác nhau (I2C cho SHTC3, RS485 Modbus RTU cho các cảm biến công nghiệp), đảm bảo không mất dữ liệu khi xử lý đa nhiệm.
 - **Thiết kế giao thức truyền thông:** Thiết kế cơ chế đóng gói dữ liệu (data packaging) và giao thức truyền tải (MQTT) để đảm bảo tính nhất quán và hiệu quả trong truyền tải, đồng thời hỗ trợ nhiều loại dữ liệu khác nhau (sensor telemetry, camera images, video streams).
 - **Xử lý dữ liệu tại edge:** Thực hiện xử lý sơ bộ (pre-processing) như chuyển đổi định dạng (Modbus RTU → JSON), lọc nhiễu cơ bản, và đảm bảo tính nhất quán dữ liệu ngay tại thiết bị để giảm tải cho server.
 - **Quản lý nguồn và khả năng tự khôi phục:** Quản lý nguồn điện và khả năng tự khôi phục (auto-recovery) khi mất kết nối, đặc biệt quan trọng với các thiết bị hoạt động ở môi trường thực địa.
- **Tầng CS (Cloud/Server Layer - Phần mềm và Dữ liệu):**
 - Xây dựng hệ thống quản lý tập trung với kiến trúc microservices để xử lý dữ liệu từ nhiều node IoT.
 - Phát triển các thuật toán Machine Learning để phát hiện lỗi cảm biến dựa trên phân tích xu hướng và pattern recognition.
 - Thiết kế cơ sở dữ liệu tối ưu (TimescaleDB cho time-series data, PostgreSQL cho metadata) để lưu trữ và truy vấn hiệu quả.



- Phát triển giao diện web thân thiện với người dùng để quản lý, giám sát và cảnh báo.
 - Xây dựng hệ thống tích hợp đa thiết bị (multi-vendor device integration) thông qua API và message queue.
- **Lớp tương tác (Interaction Layer):** Sự kết hợp giữa CE và CS tạo ra một vòng lặp phản hồi (feedback loop):
 - CE cung cấp dữ liệu thô từ môi trường vật lý → CS phân tích và phát hiện bất thường.
 - CS phát hiện lỗi và gửi cảnh báo → CE có thể thực hiện hành động khắc phục tự động (như bật quạt, bơm nước).
 - CS cung cấp giao diện quản lý → Người dùng có thể cấu hình và điều khiển → CE thực thi các lệnh điều khiển từ xa.

Kết luận: Sự cần thiết của sự kết hợp CE + CS Từ phân tích trên, có thể nhận thấy rằng bài toán đảm bảo tính toàn vẹn của luồng dữ liệu từ vật lý đến đám mây đòi hỏi sự kết hợp giữa CE và CS. Hai lĩnh vực này bổ sung lẫn nhau và tạo thành một pipeline xử lý dữ liệu đa tầng:

- **CE đảm bảo chất lượng dữ liệu tại edge:** Firmware và phần cứng được thiết kế để thực hiện xử lý sơ bộ (pre-processing) như lọc nhiễu, phát hiện lỗi hardware-level, và đảm bảo tính nhất quán dữ liệu ngay tại thiết bị. Điều này giúp giảm tải xử lý ở tầng cloud và cải thiện độ tin cậy của hệ thống.
- **CS đảm bảo khả năng mở rộng và phân tích nâng cao:** Kiến trúc phần mềm phân tán (message queue, time-series database, microservices) và các thuật toán Machine Learning (RFE, Isolation Forest) cho phép hệ thống xử lý quy mô lớn và phát hiện các mẫu lỗi phức tạp đòi hỏi phân tích dữ liệu lịch sử và tương quan giữa nhiều cảm biến.
- **Sự kết hợp tạo ra Data Integrity Pipeline:** Dữ liệu được xử lý và kiểm tra chất lượng ở nhiều tầng: tại edge (CE), trong quá trình truyền tải (với cơ chế checksum, retry), và tại cloud (CS với thuật toán phát hiện lỗi thông minh). Việc xử lý đa tầng này giúp đảm bảo tính toàn vẹn của dữ liệu từ đầu đến cuối.

Dồ án này nghiên cứu và phát triển một **hệ thống tích hợp được thiết kế có chủ đích** để giải quyết bài toán thách thức về độ tin cậy dữ liệu trong môi trường phân tán với tài nguyên hạn chế. Bài toán này **đòi hỏi kiến thức chuyên sâu từ cả hai lĩnh vực CE và CS** để có thể giải quyết một cách hiệu quả và toàn diện.

1.3 Tính cấp thiết của đề tài

Đề tài nhằm cấp thiết xuất phát từ nhu cầu thực tế của việc vận hành chuỗi nông trại quy mô thương mại và xu hướng chuyển đổi số mạnh mẽ tại Việt Nam:

- **Xu hướng tất yếu của chuyển đổi số:** Theo thống kê, tốc độ tăng trưởng GDP ngành nông nghiệp trong nửa đầu năm 2024 đạt 3.38%, mức cao nhất trong 5 năm qua, nhờ vào việc đẩy mạnh ứng dụng công nghệ cao [4]. Tuy nhiên, phần lớn các giải pháp hiện tại vẫn còn rò rỉ rác, chưa tạo thành hệ sinh thái thống nhất.
- **Nhu cầu quản lý tập trung (All-in-one):** Một trong những thách thức lớn nhất của nông nghiệp công nghệ cao hiện nay là sự thiếu liên kết chuỗi giá trị và sự manh mún trong quản lý [15]. Đối với các mô hình canh tác phân tán, việc thiếu một nền tảng quản lý hợp nhất dẫn đến lãng phí nguồn lực giám sát. Chủ đầu tư cần một giao diện duy nhất để quản lý hàng loạt nông trại.
- **Độ tin cậy của dữ liệu cảm biến:** Các nghiên cứu gần đây chỉ ra rằng cảm biến IoT trong môi trường nông nghiệp khắc nghiệt thường xuyên gặp lỗi trôi số liệu (drift) hoặc mất kết nối. Nếu không phát hiện sớm bằng các thuật toán thông minh (Data-driven), dữ liệu sai lệch sẽ dẫn đến các quyết định sai lầm [17].

1.4 Tình hình nghiên cứu và các giải pháp hiện có

Hiện nay trên thị trường tồn tại hai nhóm giải pháp chính:



- **Giải pháp trọn gói từ các hãng lớn (Israel, Nhật Bản):** Có độ ổn định cao nhưng chi phí rất đắt đỏ, hệ sinh thái đóng (không cho phép tích hợp thiết bị hãng khác), khó phù hợp với quy mô vừa và nhỏ tại Việt Nam [18].
- **Giải pháp lắp ráp nhỏ lẻ (DIY):** Giá thành rẻ nhưng thiếu tính năng quản lý tập trung, giao diện sơ sài và đặc biệt là thiếu cơ chế cảnh báo lỗi thông minh.

Khoảng trống nghiên cứu: Chưa có nhiều giải pháp tập trung vào việc chuẩn hóa thiết bị đa nguồn kết hợp với thuật toán phát hiện lỗi cảm biến dành riêng cho phân khúc người dùng không chuyên kỹ thuật.

1.5 Mục tiêu và câu hỏi nghiên cứu

Mục tiêu tổng quát: Nghiên cứu và phát triển hệ thống quản lý tập trung và nhận diện lỗi cho các thiết bị IoT, hướng tới việc cung cấp giải pháp vận hành đơn giản, tin cậy cho các mô hình nông trại thông minh đa thiết bị.

Mục tiêu cụ thể:

- Xây dựng kiến trúc quản lý dữ liệu cảm biến (Data-centric Sensor Management) có khả năng tích hợp và quản lý thiết bị từ nhiều nguồn khác nhau, trong đó mỗi cảm biến được mô hình hóa như một thực thể dữ liệu với hồ sơ hành vi (Behavior Profile).
- Phát triển pipeline phát hiện và phân loại lỗi cảm biến (Sensor Fault Detection Pipeline) ứng dụng thuật toán Machine Learning (RFE kết hợp Random Forest/XGBoost) để tự động phát hiện và phân loại các bất thường (mất kết nối, trôi dạt, giá trị bị kẹt) dựa trên phân tích dữ liệu thời gian thực và tương quan không gian.
- Thiết kế hệ thống quản lý vòng đời thiết bị dựa trên dữ liệu (Data-driven Lifecycle Management) để theo dõi xu hướng suy giảm và hỗ trợ ra quyết định bảo trì chủ động (predictive maintenance).
- Xây dựng kiến trúc hệ thống và giao diện quản trị với Dashboard hiển thị chỉ số tin cậy (Confidence Score) cho từng giá trị cảm biến, phục vụ đối tượng người dùng không chuyên kỹ thuật.

Câu hỏi nghiên cứu:

- Làm thế nào để xây dựng một cơ chế định danh và quản lý thống nhất cho các thiết bị IoT đa dạng về giao thức?
- Thuật toán nào là tối ưu để phát hiện lỗi cảm biến trong môi trường dữ liệu thời gian thực với độ trễ thấp?
- Làm thế nào để thiết kế trải nghiệm người dùng tối giản hóa các thao tác kỹ thuật phức tạp?

1.6 Phạm vi và giới hạn của đề tài

Phạm vi:

- **Bối cảnh và Kế thừa:** Đề tài được thực hiện trong bối cảnh kế thừa và tận dụng hạ tầng IoT thực tế đã được triển khai tại **Tomochan Farm** (thuộc dự án nghiên cứu của phòng thí nghiệm TIST Lab - HCMUT). Hệ thống hiện có bao gồm:

1. Hệ thống datalogger với cảm biến công nghiệp:

- Vi điều khiển chính: ESP-32 gửi dữ liệu qua WiFi
- Cảm biến độ ẩm, nhiệt độ không khí: SHTC3 Temperature Humidity
- Cảm biến ánh sáng công nghiệp: ES-ALS-02 (giao tiếp RS485 Modbus RTU)
- Cảm biến mưa: ES-RAIN-02 (giao tiếp RS485)
- Cảm biến đo pH: giao tiếp công nghiệp
- Cảm biến EC & TDS: MODBUS-RTU RS485

2. Hệ thống camera:



- Camera IP: Hikvision DS-2CD1021G0-I
- Raspberry Pi 4 Model B: đọc hình ảnh từ camera, gửi về server và stream live video qua WiFi

Trên cơ sở hệ thống thu thập vật lý đã có (Proof of Concept) với các thiết bị công nghiệp đa dạng về giao thức (WiFi, RS485 Modbus RTU), nhóm tập trung nghiên cứu xây dựng **Kiến trúc quản lý dữ liệu cảm biến (Data-centric Sensor Management)** và phát triển các thuật toán nhằm giám sát chất lượng, trạng thái và độ tin cậy của thiết bị dựa trên dòng dữ liệu thực tế.

Dóng góp của CE trong PoC: Trong giai đoạn Proof of Concept, khôi CE không chỉ đơn giản là phần cứng, mà đóng góp ở các quyết định kỹ thuật quan trọng:

- **Tích hợp đa giao thức:** Thiết kế firmware trên ESP-32 để tích hợp và chuẩn hóa dữ liệu từ nhiều loại cảm biến với giao thức khác nhau (WiFi cho SHTC3, RS485 Modbus RTU cho ES-ALS-02, ES-RAIN-02, EC & TDS, và giao tiếp với Raspberry Pi cho camera).
- **Điều phối tài nguyên:** Quản lý và điều phối việc đọc dữ liệu từ nhiều cảm biến song song, đảm bảo không mất dữ liệu khi xử lý đa nhiệm.
- **Xử lý dữ liệu tại edge:** Thực hiện xử lý sơ bộ và chuẩn hóa dữ liệu từ các định dạng khác nhau (Modbus RTU, analog, digital) trước khi gửi về server qua WiFi.
- **Dối tượng nghiên cứu:** Các thiết bị IoT cảm biến môi trường công nghiệp đã có sẵn tại vườn (như trên) và các thiết bị chấp hành cơ bản trong mô hình nhà kính.
- **Nền tảng công nghệ:** Hệ thống quản lý tập trung nền tảng Web (Web-based platform), sử dụng Cơ sở dữ liệu chuỗi thời gian (Time-series Database) tối ưu cho lưu trữ dữ liệu lớn IoT, và các giao thức truyền thông nhẹ (Lightweight messaging protocols).
- **Triển khai:** Thử nghiệm trên mô hình nông trại mẫu (Pilot testing) với dữ liệu mô phỏng và thực tế, trong đó bao gồm việc tích hợp với hệ thống hiện có tại Tomochan Farm.

Giới hạn:

- Hệ thống tập trung vào việc phát hiện lỗi phần cứng/cảm biến dựa trên phân tích dữ liệu, chưa bao gồm việc chẩn đoán sâu bệnh cây trồng bằng hình ảnh (Computer Vision).
- Các thuật toán phát hiện lỗi sẽ được kiểm nghiệm trên một số loại cảm biến đặc thù, có thể cần chỉnh lại khi áp dụng cho các loại cảm biến công nghiệp mới lạ.

2 Cơ sở lý thuyết

2.1 Kiến trúc hệ thống và Công nghệ nền tảng

2.1.1 Kiến trúc tổng thể

Hệ thống Xanh Market được thiết kế theo mô hình kiến trúc phân lớp (Layered Architecture) kết hợp với hướng dịch vụ (Service-oriented Architecture), đảm bảo tính mô-đun hóa cao và khả năng mở rộng độc lập từng thành phần. Hệ thống được tổ chức thành 5 tầng logic chính:

- **Lớp thiết bị (Device Layer):** Là tầng thấp nhất bao gồm các thiết bị IoT triển khai tại hiện trường nông trại:
 - *Trạm quan trắc (IoT Gateway):* Sử dụng ESP32 làm bộ xử lý trung tâm, tích hợp các cảm biến môi trường (DHT11, DS18B20, Soil Moisture, Light Sensor) và module truyền thông (WiFi, RS485).
 - *Thiết bị chấp hành (Actuators):* Bao gồm hệ thống tưới tiêu tự động với van điện từ, máy bơm và các cơ cấu điều khiển khác.
 - *Giao thức truyền thông:* Sử dụng MQTT qua WiFi để truyền dữ liệu thời gian thực và HTTP cho tải lên hình ảnh/video.
- **Lớp tích hợp (Integration Layer):** Dòng vai trò là cầu nối giữa thiết bị và hệ thống backend:



- *API Gateway*: Điểm vào duy nhất cho tất cả các yêu cầu từ phía client, thực hiện xác thực, định tuyến và cân bằng tải.
- *Message Broker (RabbitMQ)*: Hệ thống hàng đợi thông điệp để xử lý dữ liệu IoT bất đồng bộ, đảm bảo tính reliable và decoupling giữa producer và consumer.
- *Data Ingestion Service*: Tiếp nhận và tiền xử lý dữ liệu từ MQTT broker trước khi lưu trữ.
- **Lớp nghiệp vụ (Business Layer)**: Chứa toàn bộ logic xử lý cốt lõi của hệ thống:
 - *User Management Service*: Quản lý tài khoản người dùng, phân quyền và xác thực.
 - *Farm Management Service*: Xử lý logic quản lý nông trại, thiết bị và cấu hình giám sát.
 - *IoT Data Processor*: Service chuyên biệt để xử lý dữ liệu cảm biến, áp dụng thuật toán phát hiện lỗi (RFE) và tạo cảnh báo.
 - *Analytics Engine*: Phân tích dữ liệu lịch sử để đưa ra insights và khuyến nghị cho người dùng.
- **Lớp dữ liệu (Data Layer)**: Quản lý lưu trữ và truy xuất dữ liệu với chiến lược đa mô hình:
 - *PostgreSQL*: Cơ sở dữ liệu quan hệ lưu trữ thông tin cấu trúc (users, farms, devices, configurations).
 - *TimescaleDB*: Cơ sở dữ liệu chuỗi thời gian chuyên dụng cho dữ liệu cảm biến với hiệu năng truy vấn cao.
 - *AWS S3*: Lưu trữ file tĩnh như hình ảnh, log files và báo cáo.
- **Lớp ứng dụng (Application Layer)**: Giao diện người dùng cuối:
 - *Web Admin Portal*: Giao diện quản trị chính với dashboard thời gian thực, biểu đồ trực quan hóa dữ liệu và công cụ quản lý hệ thống.
 - *RESTful APIs*: Cung cấp các endpoint chuẩn cho tích hợp với hệ thống bên thứ ba.
 - *Real-time Notifications*: Hệ thống cảnh báo tức thì qua WebSocket cho các sự kiện quan trọng.

Kiến trúc này đảm bảo tính linh hoạt cao với khả năng mở rộng ngang (horizontal scaling) thông qua containerization (Docker) và orchestration tự động. Mô hình phân tán giúp hệ thống chịu tải tốt với hàng trăm thiết bị IoT đồng thời, đồng thời đảm bảo tính bảo mật với cơ chế xác thực đa lớp và mã hóa dữ liệu.

2.1.2 Nền tảng ứng dụng

Hệ thống sử dụng giao diện Web Admin làm nền tảng chính cho việc quản trị và giám sát. Thiết kế web-based mang lại các ưu điểm sau:

- **Truy cập đa nền tảng**: Web Admin có thể được truy cập từ bất kỳ thiết bị nào có trình duyệt web hiện đại (desktop, laptop, tablet), không yêu cầu cài đặt phần mềm đặc biệt.
- **Định danh và xác thực**: Sử dụng cơ chế đăng nhập dựa trên tài khoản quản trị viên với phân quyền chi tiết (Role-Based Access Control), đảm bảo tính bảo mật cao.
- **Thông báo thời gian thực**: Hệ thống tích hợp WebSocket để gửi cảnh báo và cập nhật dữ liệu thời gian thực trực tiếp trên giao diện web.
- **Công nghệ phát triển**: Giao diện được xây dựng bằng ReactJS với Vite, kết hợp với các thư viện trực quan hóa dữ liệu như Chart.js để tạo dashboard tương tác.

2.1.3 Hạ tầng Backend và Quy trình DevOps

Hệ thống Backend được xây dựng trên nền tảng điện toán đám mây riêng (Private Cloud) tại phòng thí nghiệm HPC Lab để đảm bảo tính bảo mật và chủ quyền dữ liệu. Quy trình triển khai áp dụng các tiêu chuẩn công nghiệp:

- **Áo hóa và Container hóa**: Sử dụng Docker để đóng gói các dịch vụ, đảm bảo môi trường vận hành đồng nhất giữa phát triển và sản xuất.



- **CI/CD Pipeline:** Tích hợp quy trình Tích hợp liên tục và Triển khai liên tục (CI/CD) để tự động hóa việc cập nhật phần mềm.
- **Kiểm soát chất lượng mã nguồn:** Sử dụng SonarQube để tự động quét và phân tích mã nguồn, phát hiện các lỗ hổng bảo mật và đảm bảo chất lượng code trước khi triển khai.
- **Giám sát hệ thống (Monitoring):** Sử dụng bộ công cụ Grafana và Prometheus để theo dõi hiệu năng server, tài nguyên hệ thống (CPU, RAM, Disk) và trạng thái các dịch vụ theo thời gian thực.

2.2 Giao thức công nghiệp

Dể khắc phục các nhược điểm về nhiễu tín hiệu và suy hao đường truyền của các cảm biến Analog truyền thống trong môi trường thực tế, hệ thống sử dụng chuẩn giao tiếp công nghiệp dựa trên sự kết hợp giữa RS485 (Lớp vật lý) và Modbus RTU (Lớp giao thức).

2.2.1 Chuẩn vật lý RS485 (Physical Layer)

RS485 (Recommended Standard 485) là chuẩn truyền thông nối tiếp vi sai (differential signaling), được thiết kế chuyên dụng cho các môi trường công nghiệp có nhiều nhiễu điện từ.

- **Cơ chế hoạt động:** Khác với chuẩn RS232 hay tín hiệu Analog (so sánh điện áp với dây đất - GND), RS485 sử dụng hai dây tín hiệu A và B xoắn lại với nhau (Twisted pair). Giá trị logic được xác định bằng hiệu điện thế giữa hai dây:

$$V_{diff} = V_A - V_B$$

- Logic 1: $V_A - V_B > 200mV$
- Logic 0: $V_A - V_B < -200mV$

- **Khả năng chống nhiễu (Noise Immunity):** Đây là đặc tính quan trọng nhất. Khi có nhiễu điện từ tác động lên đường dây, nó sẽ tác động đồng thời lên cả dây A và dây B với biên độ như nhau. Vì bộ thu chỉ quan tâm đến hiệu số điện áp $(V_A + V_{noise}) - (V_B + V_{noise}) = V_A - V_B$, nên thành phần nhiễu bị triệt tiêu hoàn toàn.

2.2.2 Giao thức Modbus RTU (Data Link Layer)

Modbus RTU là một giao thức dạng Master-Slave hoạt động trên nền tảng vật lý RS485. Dữ liệu được mã hóa dưới dạng nhị phân nhỏ gọn để tối ưu tốc độ.

- **Cấu trúc gói tin:** Mỗi gói tin bao gồm Địa chỉ Slave, Mã hàm (Function Code), Dữ liệu và mã kiểm lỗi CRC.
- **Độ tin cậy dữ liệu (Data Integrity):** Modbus RTU sử dụng cơ chế kiểm tra lỗi CRC-16 (Cyclic Redundancy Check). Khi thiết bị nhận được gói tin, nó sẽ tính toán lại mã CRC và so sánh với mã nhận được. Nếu không trùng khớp (do nhiễu đường truyền làm sai lệch bit), gói tin sẽ bị hủy bỏ ngay lập tức thay vì nhận sai giá trị. Điều này đảm bảo dữ liệu đầu vào cho thuật toán luôn chính xác.

2.2.3 Lý do lựa chọn RS485/Modbus thay vì Analog

Cảm biến Analog phụ thuộc vào bộ đọc ADC của ESP32, mà ADC của ESP32 nổi tiếng là có độ nhiễu cao và không tuyến tính. Cảm biến Modbus thực hiện chuyển đổi số (ADC) ngay tại đầu dò với chip chuyên dụng, sau đó gửi dữ liệu số về. Điều này giúp loại bỏ hoàn toàn sai số do đường dây và do bộ vi điều khiển, đảm bảo dữ liệu đầu vào cho thuật toán RFE là chính xác nhất. Việc chuyển đổi từ cảm biến Analog (0-10V hoặc 4-20mA) sang cảm biến Modbus RS485 mang lại những ưu điểm vượt trội cho tính ổn định của hệ thống:



Đặc điểm	Cảm biến Analog	Cảm biến Modbus RS485
Cơ chế truyền	Điện áp/Dòng điện liên tục	Gói tin số (Digital Packets)
Chống nhiễu	Kém: Dễ bị nhiễu cao tần làm sai lệch giá trị điện áp đọc về	Rất tốt: Tự triệt tiêu nhiễu nhờ tín hiệu vi sai
Khoảng cách	Ngắn (<20m). Dây dài gây sụt áp, sai số lớn	Xa (lên đến 1200m) mà không suy giảm chất lượng tín hiệu
Dộ chính xác	Phụ thuộc vào độ phân giải ADC của vi điều khiển (ESP32 ADC thường không tuyến tính)	Dữ liệu số được chuyển đổi chính xác ngay tại cảm biến (ADC chuyên dụng bên trong)
Mở rộng	Mỗi cảm biến 1 dây kết nối riêng	Tất cả märk song song trên 1 cặp dây Bus duy nhất

Bảng 2: So sánh cảm biến Analog và Modbus RS485 trong môi trường nông nghiệp

2.3 Bài toán phát hiện lỗi cảm biến trong hệ thống IoT

2.3.1 Lý thuyết về chất lượng dữ liệu cảm biến

Trong các hệ thống IoT và mạng cảm biến không dây (WSN), dữ liệu thu thập được thường chịu ảnh hưởng bởi các yếu tố môi trường, phần cứng xuống cấp hoặc nhiễu đường truyền. Chất lượng dữ liệu kém sẽ ảnh hưởng trực tiếp đến hiệu suất của các thuật toán học máy, đặc biệt là quá trình chọn lọc đặc trưng (Feature Selection). Để đảm bảo tính chính xác cho thuật toán Robust Feature Extraction (RFE), việc nhận diện và xử lý các lỗi cảm biến phổ biến là bước tiền xử lý bắt buộc. Dưới đây là định nghĩa và đặc điểm của ba loại lỗi thường gặp nhất: Drift, Stuck-at, và Spike.

2.3.1.a Lỗi trôi (Drift Fault)

Định nghĩa: Lỗi trôi (Drift) xảy ra khi giá trị đo được của cảm biến lêch dần khỏi giá trị thực theo thời gian một cách tuyến tính hoặc phi tuyến tính, mặc dù hiện tượng vật lý đang đó không thay đổi theo xu hướng đó. **Đặc điểm nhận diện:**

- Độ lệch (offset) tăng dần theo thời gian.
- Thường xuất hiện do sự xuống cấp của linh kiện cảm biến (aging) hoặc thay đổi chậm của môi trường (nhiệt độ, độ ẩm ảnh hưởng đến hiệu chuẩn).

Mô hình toán học: Nếu $x(t)$ là giá trị thực và $y(t)$ là giá trị đo được tại thời điểm t , lỗi trôi có thể được mô tả:

$$y(t) = x(t) + \delta(t)$$

Trong đó $\delta(t)$ là hàm lỗi tăng đơn điệu theo thời gian. **Tác động đến thuật toán RFE:** Lỗi Drift làm thay đổi phân phối dữ liệu. Điều này khiến RFE có thể đánh giá sai tầm quan trọng của đặc trưng (feature importance), vì mô hình có thể "học" nhầm xu hướng trôi này là một quy luật của dữ liệu thay vì loại bỏ nó.

2.3.1.b Lỗi Kẹt giá trị (Stuck-at Fault)

Định nghĩa: Lỗi kẹt (Stuck-at) là hiện tượng giá trị đầu ra của cảm biến giữ nguyên tại một hằng số bất chấp sự thay đổi của hiện tượng vật lý môi trường. **Đặc điểm nhận diện:**

- Stuck-at-Zero: Giá trị kẹt tại 0.
- Stuck-at-Constant: Giá trị kẹt tại một số thực C bất kỳ (ví dụ: giá trị max hoặc min của thang đo).
- Phương sai (Variance) của chuỗi dữ liệu trong khoảng thời gian lỗi bằng 0.

Tác động đến thuật toán RFE: Lỗi Stuck-at làm mất tính đa dạng của dữ liệu trong khoảng thời gian lỗi, dẫn đến việc RFE không thể trích xuất các đặc trưng động học (dynamic features) như tốc độ thay đổi hay độ biến động cục bộ, từ đó ảnh hưởng đến khả năng phân biệt giữa trạng thái bình thường và lỗi.



2.3.1.c Lỗi Gai nhiễu (Spike Fault)

Định nghĩa: Lỗi gai (Spike) là những biến động đột ngột, có biên độ lớn bất thường xuất hiện trong một khoảng thời gian rất ngắn, sau đó giá trị trở về mức bình thường. **Đặc điểm nhận diện:**

- Tốc độ thay đổi (rate of change) giữa hai điểm dữ liệu liên tiếp vượt quá giới hạn vật lý mà cảm biến có thể đo đạc.
- Xuất hiện ngẫu nhiên và rời rạc.

Tác động đến thuật toán RFE: Spike làm tăng phương sai ảo của dữ liệu và làm lệch các tham số thống kê (như giá trị trung bình, độ lệch chuẩn). Trong RFE, điều này có thể dẫn đến việc xếp hạng sai (mis-ranking) các đặc trưng, vì một số thuật toán đánh giá cao các đặc trưng có phương sai lớn (dù phương sai đó đến từ nhiễu).

2.3.2 Các phương pháp tiếp cận hiện có và hạn chế

Hiện nay, bài toán phát hiện lỗi cảm biến thường được giải quyết theo ba hướng chính, tuy nhiên mỗi hướng đều tồn tại những hạn chế khi áp dụng cho các thiết bị IoT có tài nguyên hạn chế (Low-power IoT devices):

- **Học máy cổ điển (Classic ML):** Thường gặp khó khăn trong việc nắm bắt các mẫu lỗi phức tạp hoặc phi tuyến tính trong chuỗi thời gian.
- **Học sâu (Deep Learning - DL):** Các mô hình như Autoencoder hay LSTM dù mạnh mẽ nhưng lại là những "hộp đen" (black-box), thiếu tính giải thích. Quan trọng hơn, chúng đòi hỏi tài nguyên tính toán và năng lượng lớn, không phù hợp để triển khai tại biên (Edge/Node).
- **Phương pháp dựa trên tương quan:** Hoạt động kém hiệu quả trong môi trường động và thường bỏ qua các thông tin nội tại của từng dòng dữ liệu riêng lẻ.

Từ những phân tích trên, đặt ra yêu cầu về một phương pháp trích xuất đặc trưng (Feature Extraction) chuyên biệt, vừa đảm bảo độ chính xác cao, vừa tối ưu hóa tài nguyên tính toán.

2.3.3 Cơ sở lý thuyết của phương pháp Robust Feature Extractor (RFE)

Để giải quyết bài toán trên, nghiên cứu đề xuất phương pháp Robust Feature Extractor (RFE). Thay vì tăng độ phức tạp của mô hình phân loại, RFE tập trung RFE hoạt động như một bộ lọc đa tầng biến đổi tín hiệu đầu vào x_t tại thời điểm t thành vector đặc trưng F_t . Các thành phần cốt lõi của RFE bao gồm:

- **Tốc độ thay đổi (Speed of Change):** Để nắm bắt sự thay đổi đột ngột (thường gặp ở lỗi Spike), RFE tính toán đạo hàm bậc nhất rời rạc:

$$v_t = x_t - x_{t-1}$$

Giá trị này đại diện cho "vận tốc" của tín hiệu.

- **Độ biến động cục bộ (Local Variance):** Đây là điểm cải tiến quan trọng của RFE. Thuật toán tính toán độ lệch chuẩn trượt (Rolling Standard Deviation) trên chính giá trị vận tốc v_t trong cửa sổ w :

$$\sigma_t = \sqrt{\frac{1}{w} \sum_{i=0}^{w-1} (v_{t-i} - \bar{v})^2}$$

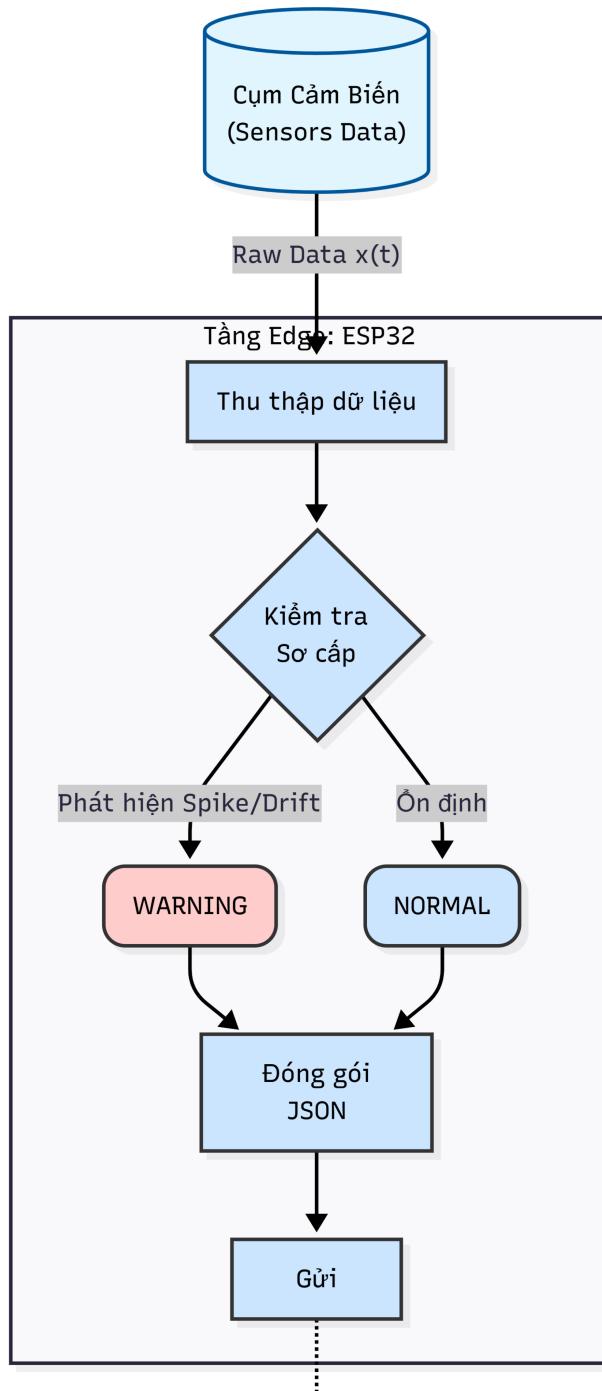
Đặc trưng này giúp phân biệt nhiều môi trường (biến động thấp) với lỗi cảm biến (biến động bất thường).

- **Xu hướng dài hạn (Smoothed Trends):** Sử dụng Trung bình động lũy thừa (EWMA) để làm mượt tín hiệu, giúp mô hình nhận diện các lỗi trôi (Drift) diễn ra từ từ:

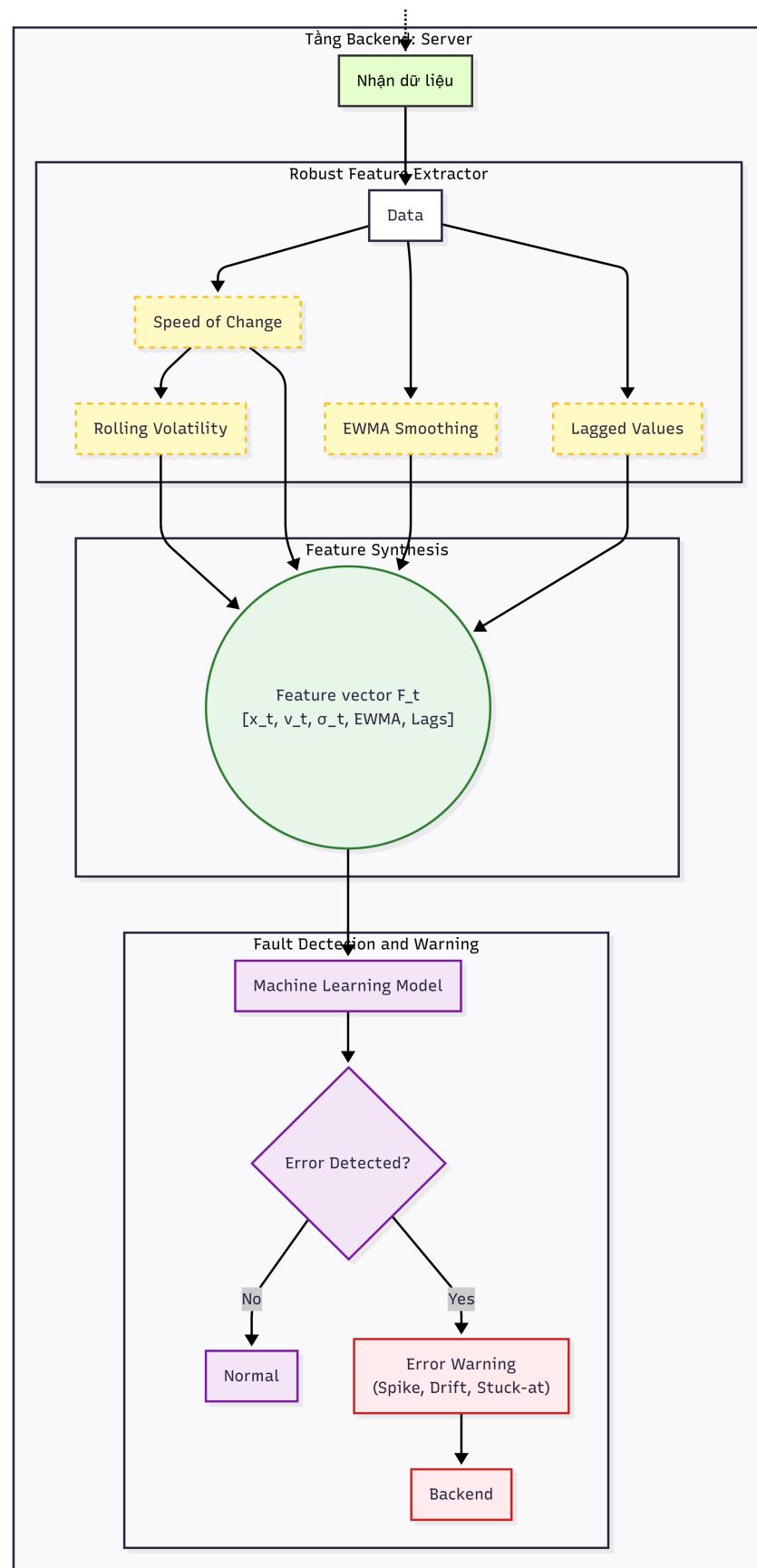
$$EWMA_t = \alpha \cdot x_t + (1 - \alpha) \cdot EWMA_{t-1}$$

- **Bộ nhớ thời gian (Temporal Memory):** Bổ sung các giá trị trễ (Lag features: x_{t-1}, x_{t-2}, \dots) để cung cấp ngữ cảnh quá khứ cho mô hình phân loại tại thời điểm hiện tại.

2.3.4 Sơ đồ Luồng xử lý tổng thể



Hình 1: Sơ đồ luồng xử lý sơ cấp ở Edge Node



Hình 2: Sơ đồ luồng xử lý phát hiện lỗi với RFE ở Backend Server



2.4 Edge Computing

2.4.1 Định nghĩa

Tính toán biên (Edge Computing) là một mô hình tính toán phân tán, trong đó việc xử lý và lưu trữ dữ liệu được thực hiện ngay tại hoặc gần nguồn tạo ra dữ liệu (các thiết bị cảm biến, node mạng), thay vì dựa hoàn toàn vào một máy chủ trung tâm (Cloud/Server) ở xa. Trong ngữ cảnh của hệ thống IoT (Internet of Things), các vi điều khiển như ESP32 hoặc máy tính nhúng như Raspberry Pi đóng vai trò là các Edge Node. Chúng không chỉ đơn thuần thu thập và chuyển tiếp dữ liệu (Pass-through), mà còn có khả năng thực thi các thuật toán tính toán tại chỗ.

2.4.2 Vai trò giảm tải cho Server

Việc chuyển dịch một phần năng lực xử lý từ Backend xuống Edge mang lại hiệu quả tối ưu hóa hệ thống dựa trên nguyên lý "lọc trước khi gửi". Cụ thể:

- **Giảm băng thông mạng (Bandwidth Optimization):** Thay vì truyền tải liên tục dữ liệu thô (raw data) với tần suất cao (ví dụ: 100 mẫu/giây), thiết bị biên chỉ gửi các dữ liệu có ý nghĩa, dữ liệu đã qua nén hoặc các sự kiện (events) quan trọng.

$$R_{send} = \alpha \times R_{raw}$$

Trong đó R_{send} là lượng dữ liệu gửi đi, R_{raw} là dữ liệu thu thập, và $\alpha < 1$ là tỷ lệ lọc.

- **Giảm độ trễ (Latency Reduction):** Các quyết định tức thời (như ngắt mạch khi phát hiện quá nhiệt/Spike) cần được thực hiện ngay tại biên (milimet giây) thay vì chờ Round-trip time (RTT) gửi lên Server xử lý rồi nhận lệnh về.
- **Tiết kiệm tài nguyên Server:** Server không phải tốn chu kỳ CPU để xử lý các tác vụ mức thấp như lọc nhiễu, chuẩn hóa định dạng hay loại bỏ các giá trị null/NaN. Server chỉ tập trung vào các tác vụ cao cấp như lưu trữ dài hạn, phân tích xu hướng (Trend Analysis) và chạy các thuật toán phức tạp như RFE (Rpbust Feature Extraction).

2.4.3 Mô hình xử lý dữ liệu tại biên để tài

Dựa trên lý thuyết Edge Computing, hệ thống đề xuất áp dụng quy trình xử lý 3 giai đoạn tại ESP32:

- **Giai đoạn Tiền xử lý (Preprocessing):** Áp dụng các bộ lọc số (Digital Filters) đơn giản như Trung bình trượt (Moving Average) hoặc Lọc trung vị (Median Filter) để loại bỏ nhiễu trắng và làm mượt dữ liệu trước khi xử lý tiếp.
- **Giai đoạn Gán nhãn sơ cấp (Primary Labeling):** Thiết bị biên so sánh dữ liệu với các ngưỡng an toàn (Thresholds) hoặc kiểm tra tính liên tục của tín hiệu để gán nhãn trạng thái (Metadata). Ví dụ: Gán nhãn Normal, Warning (cảnh báo trôi - Drift), hoặc Error (lỗi kẹt - Stuck-at).
- **Giai đoạn Quyết định truyền tải:** Dựa trên nhãn dữ liệu, ESP32 quyết định có gửi gói tin đi hay không, hoặc gửi cảnh báo ưu tiên cao. Điều này đảm bảo Server luôn nhận được dữ liệu "sạch" hoặc dữ liệu lỗi đã được định danh rõ ràng.

Tiêu chí	Cloud Computing	Edge Computing
Dữ liệu truyền	Dữ liệu thô	Dữ liệu đã qua xử lý sơ bộ
Băng thông	Cao	Thấp
Phản hồi	Độ trễ cao	Độ trễ thấp
Nhiệm vụ server	Xử lý chuyên sâu và sơ cấp	Chỉ xử lý chuyên sâu
Xử lý nhiễu	Nhiều được gửi	Nhiều bị loại bỏ sơ bộ tại nguồn

Bảng 3: So sánh Cloud Computing và Edge Computing trong hệ thống IoT



3 Khảo sát các giải pháp đã có

- 3.1 Khảo sát các ứng dụng, sản phẩm hoặc hệ thống liên quan
- 3.2 Khảo sát các mô hình, thuật toán, phương pháp
- 3.3 Bảng tổng hợp và phân tích khoảng trống



4 Phân tích yêu cầu

- 4.1 Mục đích
- 4.2 Phương pháp thu thập yêu cầu
- 4.3 Kết quả thu thập yêu cầu
- 4.4 Yêu cầu chức năng

Phân loại mức độ ưu tiên

Để đảm bảo tính khả thi trong giai đoạn Đề án Chuyên ngành, các yêu cầu chức năng được phân loại theo mức độ ưu tiên như sau:

- [CORE] - Chức năng tối thiểu bắt buộc: Đây là tập chức năng cốt lõi phải được hiện thực đầy đủ trong Đề án Tốt nghiệp, bao gồm các tính năng thiết yếu cho hệ thống hoạt động cơ bản và điểm khác biệt chính của đề tài.
- [OPTIONAL] - Chức năng mở rộng: Đây là các tính năng nâng cao giao diện người dùng, sẽ được bổ sung nếu còn thời gian và nguồn lực sau khi hoàn thành các chức năng CORE.

Dựa trên phân tích hiện trạng và mô hình nghiệp vụ, các yêu cầu chức năng được nhóm và phân loại như sau:

Nhóm 1: Tổng quan (Dashboard Overview)

Mã YC	Độ ưu tiên	Mô tả chi tiết
FR-03	CORE	Hệ thống phải hiển thị thông số tổng quan ngay khi đăng nhập, bao gồm: Tổng số người dùng, Số nông trại đang hoạt động (Active Farms), Số thiết bị đang kết nối (Connected Devices) và Thời gian hoạt động của hệ thống (System Uptime).
FR-04	OPTIONAL	Hệ thống phải trực quan hóa dữ liệu hoạt động hàng tuần (Weekly Activity) dưới dạng biểu đồ cột hoặc đường , cho phép Admin so sánh nhanh số lượng người dùng mới, thiết bị mới và bài viết mới.
FR-05	CORE	Hệ thống phải hiển thị danh sách Cảnh báo hệ thống (System Alerts) gần nhất (ví dụ: Nhiệt độ cao, Lịch bảo trì) ngay trên màn hình chính để Admin xử lý kịp thời.
FR-06	OPTIONAL	Hệ thống phải cung cấp các nút Thao tác tiện lợi cho phép Admin tạo nhanh Người dùng mới, Nông trại mới hoặc Thêm thiết bị mới chỉ với 1 thao tác.

Bảng 4: Yêu cầu chức năng - Nhóm 1: Tổng quan Dashboard

Nhóm 2: Quản lý Nông trại và Người dùng

Mã YC	Độ ưu tiên	Mô tả chi tiết
FR-07	CORE	(Quản lý Nông trại) Hệ thống phải cho phép Admin thực hiện đầy đủ các chức năng (CRUD): Tạo mới, Xem chi tiết, Cập nhật thông tin và Xóa (hoặc vô hiệu hóa) một nông trại khỏi hệ thống.
FR-08	CORE	(Quản lý Người dùng) Hệ thống phải cho phép quản lý danh sách người dùng (Thương lái/Chủ vườn), bao gồm việc cấp phát tài khoản và phân quyền truy cập (Role-based Access Control) vào từng nông trại cụ thể.
FR-09	CORE	(Gán thiết bị) Hệ thống phải cho phép Admin thực hiện thao tác gán (assign) một hoặc nhiều thiết bị IoT cụ thể vào một nông trại đã định.

Bảng 5: Yêu cầu chức năng - Nhóm 2: Quản lý Nông trại và Người dùng



Mã YC	Độ ưu tiên	Mô tả chi tiết
FR-01	CORE	(Phát hiện lỗi) Hệ thống phải có khả năng tự động giám sát dòng dữ liệu thời gian thực từ các cảm biến và phát hiện các trạng thái bất thường (ví dụ: mất kết nối quá 5 phút, giá trị vượt ngưỡng an toàn, hoặc dữ liệu bị "đóng băng").
FR-02	CORE	(Cảnh báo) Hệ thống phải gửi cảnh báo tức thì đến người quản trị (qua giao diện Dashboard, Email hoặc thông báo đẩy) ngay khi một lỗi cảm biến được xác nhận.
FR-10	CORE	(Đăng ký thiết bị) Hệ thống phải cho phép đăng ký thiết bị ESP32 mới vào hệ thống thông qua mã định danh duy nhất (Device ID/MAC Address).
FR-11	CORE	(Giám sát trạng thái) Hệ thống phải hiển thị trạng thái kết nối thời gian thực (Online/Offline) của từng thiết bị. Nếu thiết bị mất kết nối quá thời gian quy định (ví dụ: 5 phút), hệ thống phải tự động cập nhật trạng thái sang Offline.
FR-12	OPTIONAL	(Diều khiển từ xa) Hệ thống phải cho phép người dùng gửi lệnh điều khiển (Bật/Tắt) xuống các thiết bị chấp hành (Actuators) thông qua giao diện Web.

Bảng 6: Yêu cầu chức năng - Nhóm 3: Quản lý và Giám sát Thiết bị

Nhóm 3: Quản lý và Giám sát Thiết bị

Tóm tắt phân loại ưu tiên

Mức ưu tiên	Số lượng	Các chức năng bao gồm
CORE	8 chức năng	Tập tối thiểu bắt buộc cho Đề án Tốt nghiệp: phát hiện lỗi cảm biến (FR-01, FR-02), quản lý nông trại/người dùng (FR-07, FR-08, FR-09), quản lý thiết bị cơ bản (FR-10, FR-11), và thông số tổng quan với cảnh báo (FR-03, FR-05).
OPTIONAL	3 chức năng	Các tính năng nâng cao giao diện: biểu đồ thống kê (FR-04), thao tác nhanh (FR-06), và điều khiển từ xa (FR-12) - sẽ triển khai nếu còn thời gian.

Bảng 7: Tóm tắt phân loại ưu tiên các yêu cầu chức năng

4.5 Yêu cầu phi chức năng

NFR-01: Hiệu năng và Khả năng chịu tải

- Môi trường triển khai:** Hệ thống phải hoạt động ổn định trên hạ tầng thử nghiệm bao gồm 01 thiết bị phần cứng thực (ESP32) kết hợp với mạng lưới thiết bị mô phỏng (Simulated Devices).
- Khả năng chịu tải đồng thời:** Hệ thống phải duy trì kết nối và xử lý dữ liệu ổn định từ **50 thiết bị mô phỏng** gửi dữ liệu liên tục (tần suất 5 giây/bản tin) thông qua giao thức MQTT.
- Độ trễ xử lý (Latency):**
 - Đối với thiết bị thực: Độ trễ hiển thị dữ liệu lên Dashboard **dưới 5 giây** (trong điều kiện mạng tiêu chuẩn).
 - Đối với thiết bị mô phỏng: Đảm bảo không xảy ra hiện tượng mất gói tin (packet loss) tại Message Broker khi tải đạt đỉnh.
- Thời gian phản hồi Web:** Các thao tác truy xuất dữ liệu lịch sử hoặc tải danh sách thiết bị trên Web Admin phải hoàn tất trong vòng **dưới 2 giây**.

NFR-02: Khả năng mở rộng

Hệ thống phải hỗ trợ việc thêm mới thiết bị hoặc nông trại mà không cần tắt server để bảo trì

(Zero-downtime scaling). Kiến trúc Microservices cho phép mở rộng độc lập module IoT Data Processor.

NFR-03: Độ tin cậy và Tính sẵn sàng

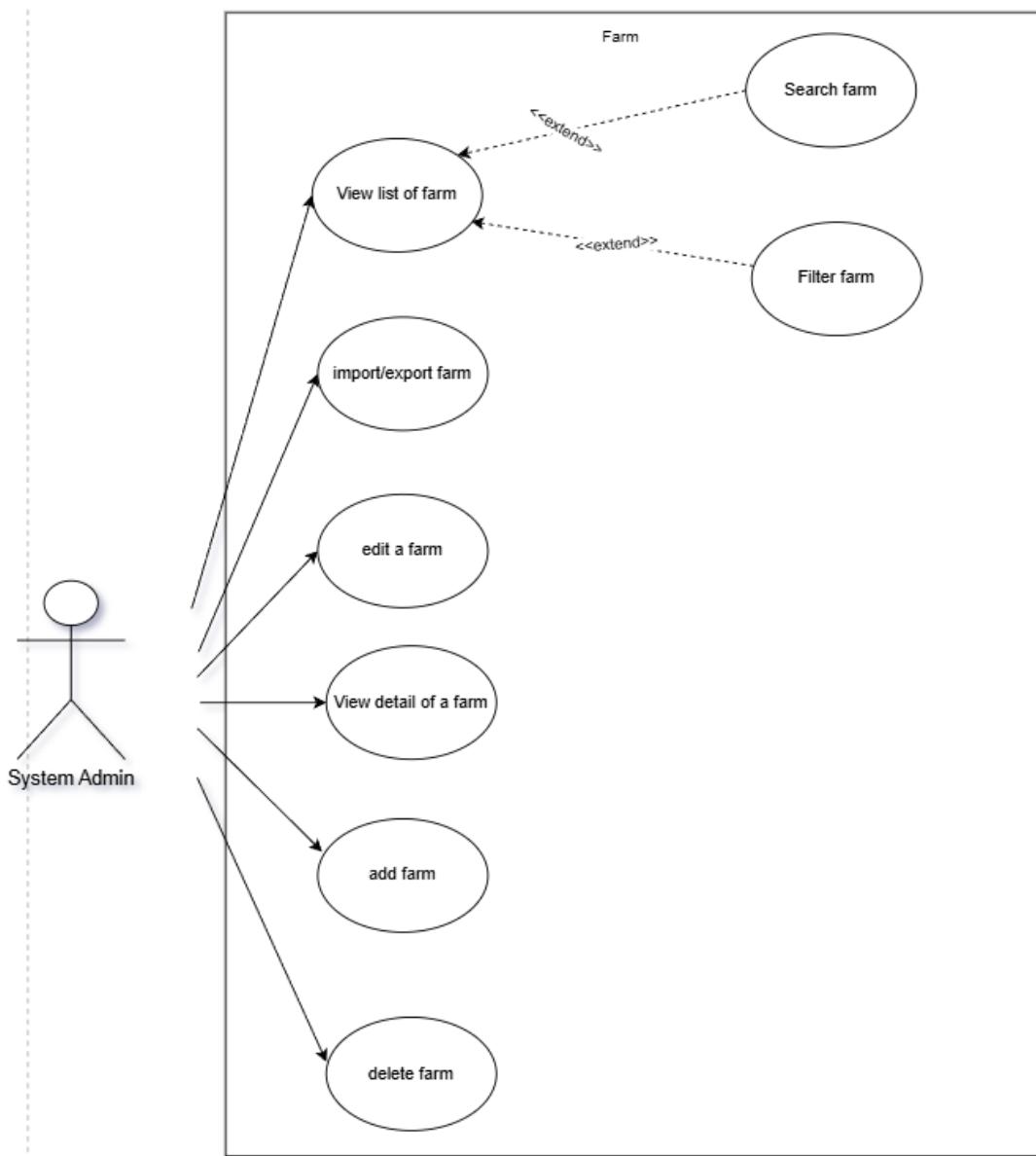
Hệ thống phải đảm bảo thời gian hoạt động (Uptime) đạt **99%**. Cơ chế **Auto-reconnect** phải hoạt động hiệu quả để Dashboard và Thiết bị tự động kết nối lại ngay khi đường truyền internet được khôi phục.

NFR-04: Bảo mật

Dữ liệu truyền từ thiết bị về Server phải được mã hóa hoặc xác thực quyền truy cập. Mật khẩu người dùng phải được băm (hashing) trước khi lưu vào cơ sở dữ liệu.

4.6 Các mô hình phân tích UML

4.6.1 Use case: Giám sát Nông trại



Hình 3: Sơ đồ quản lý farm



Mã số usecase	UC-01: Add Farm
Tên usecase	Tạo farm
Mô tả	Admin tạo 1 Farm
Actor	System Admin
Tiền điều kiện	Tài khoản Admin đã đăng nhập và có quyền truy cập module Farm.
Hậu điều kiện	Farm mới tạo được hiển thị trên danh sách farm
Trigger	Admin nhấn chọn menu “Farm” trên thanh điều hướng.
Luồng chính	<ol style="list-style-type: none">Admin truy cập màn hình Farm.Hệ thống kiểm tra quyền xem của Admin.Admin chọn nút “Add Farm”.Hệ thống hiển thị Form để nhập thông tin.Admin điền thông tin cho farm rồi ấn nút “Save”Hệ thống hiển thị cập nhật Farm mới vào danh sách
Quy tắc nghiệp vụ	<ul style="list-style-type: none">BR-01 (Authorization): Chỉ tài khoản có vai trò System Admin mới được phép tạo Farm.BR-02 (Required Fields): Các thông tin bắt buộc của Farm (ví dụ: Tên Farm, Vị trí) không được để trống.BR-03 (Unique Farm Name): Tên Farm phải là duy nhất và không được trùng lặp với các Farm đã tồn tại trong hệ thống.BR-04 (Data Validation): Dữ liệu nhập vào phải đúng định dạng và nằm trong phạm vi cho phép theo quy định của hệ thống.
Luồng thay thế / Mở rộng	<ul style="list-style-type: none">E-01 (Cancel Create Farm): Tại bước 4-5, Admin nhấn nút mũi tên “Back” ở góc bên trái hoặc nút “Cancel” cạnh nút “Save” → Hệ thống hủy thao tác tạo Farm và quay về màn hình danh sách Farm, không lưu dữ liệu.

Mã số usecase	UC-02: View Farm List
Tên usecase	Xem danh sách Farm
Mô tả	Admin xem danh sách các Farm có trong hệ thống
Actor	System Admin
Tiền điều kiện	Tài khoản Admin đã đăng nhập và có quyền truy cập module Farm
Hậu điều kiện	Danh sách Farm được hiển thị trên màn hình
Trigger	Admin nhấn chọn menu “Farm” trên thanh điều hướng
Luồng chính	<ol style="list-style-type: none">Admin truy cập menu Farm.Hệ thống kiểm tra quyền truy cập của Admin.Hệ thống truy vấn dữ liệu Farm từ cơ sở dữ liệu.Hệ thống hiển thị danh sách Farm.
Quy tắc nghiệp vụ	<ul style="list-style-type: none">BR-01 (Authorization): Chỉ tài khoản có vai trò System Admin mới được phép xem danh sách Farm.BR-02 (Pagination): Danh sách Farm được phân trang để đảm bảo hiệu năng hiển thị.



Mã số usecase	UC-05: Edit Farm
Tên usecase	Chỉnh sửa Farm
Mô tả	Admin chỉnh sửa thông tin Farm
Actor	System Admin
Tiền điều kiện	Farm tồn tại trong hệ thống
Hậu điều kiện	Thông tin Farm được cập nhật
Trigger	Admin chọn nút “Edit”
Luồng chính	<ol style="list-style-type: none">Admin chọn Farm cần chỉnh sửa.Hệ thống hiển thị form chỉnh sửa.Admin cập nhật thông tin.Admin nhấn “Save”.Hệ thống lưu và cập nhật danh sách Farm.
Quy tắc nghiệp vụ	<ul style="list-style-type: none">BR-01 (Authorization): Chỉ System Admin được phép chỉnh sửa Farm.BR-02 (Validation): Dữ liệu chỉnh sửa phải hợp lệ.

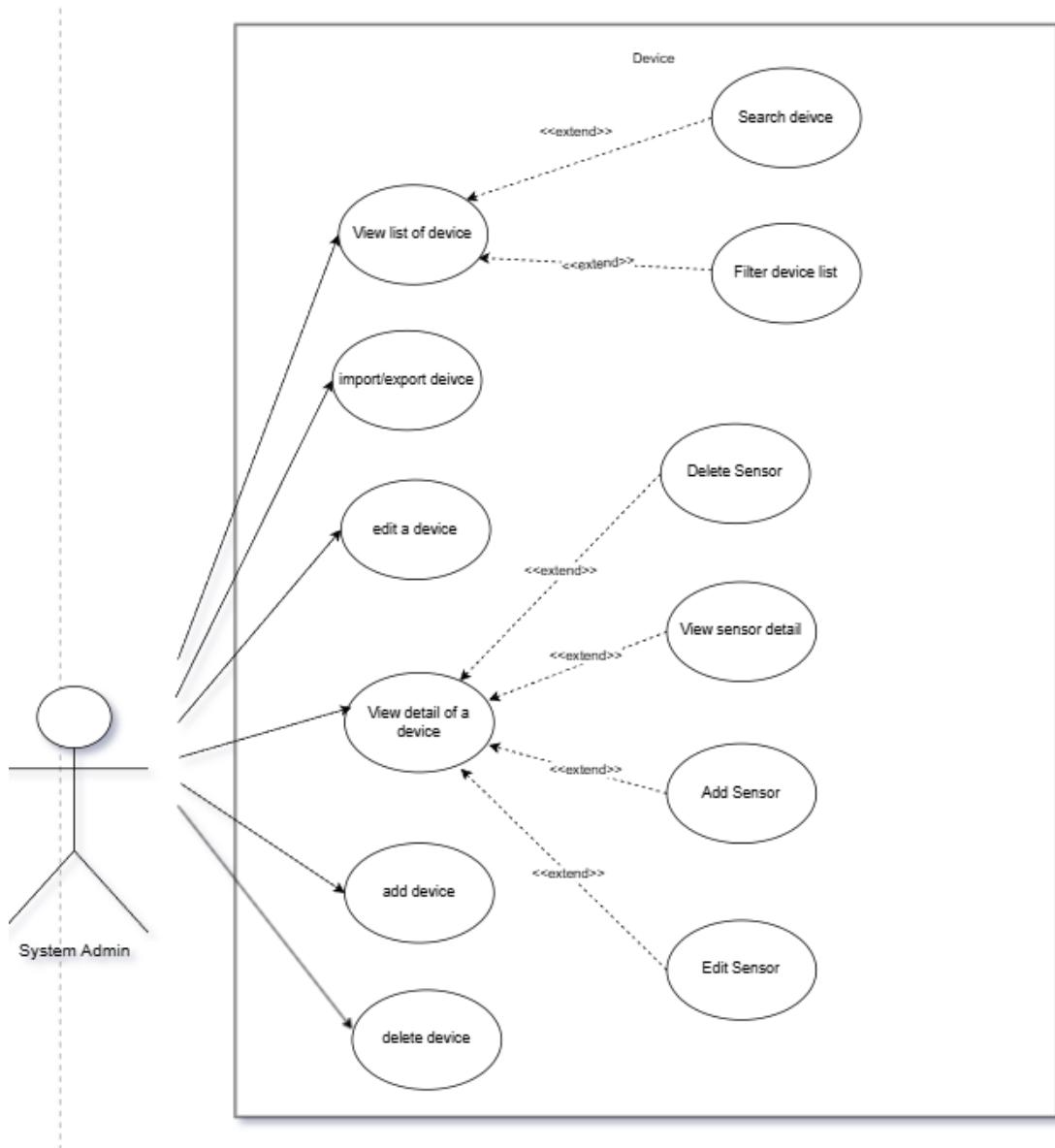
Mã số usecase	UC-06: Delete Farm
Tên usecase	Xóa Farm
Mô tả	Admin xóa Farm khỏi hệ thống
Actor	System Admin
Tiền điều kiện	Farm tồn tại
Hậu điều kiện	Farm bị xóa khỏi danh sách
Trigger	Admin chọn nút “Delete”
Luồng chính	<ol style="list-style-type: none">Admin chọn Farm cần xóa.Hệ thống hiển thị hộp thoại xác nhận.Admin xác nhận xóa.Hệ thống xóa Farm và cập nhật danh sách.
Quy tắc nghiệp vụ	<ul style="list-style-type: none">BR-01 (Confirmation): Phải xác nhận trước khi xóa.

Mã số usecase	UC-07: Import/Export Farm
Tên usecase	Import / Export Farm
Mô tả	Admin nhập hoặc xuất dữ liệu Farm
Actor	System Admin
Tiền điều kiện	Admin có quyền truy cập module Farm
Hậu điều kiện	Dữ liệu Farm được nhập hoặc xuất thành công
Trigger	Admin chọn chức năng Import hoặc Export
Luồng chính	<ol style="list-style-type: none">Admin chọn Import hoặc Export.Hệ thống hiển thị tùy chọn file.Admin xác nhận thao tác.Hệ thống xử lý dữ liệu.
Quy tắc nghiệp vụ	<ul style="list-style-type: none">BR-01 (File Format): Chỉ hỗ trợ định dạng cho phép (CSV, Excel).



Mã số usecase	UC-08: View Farm Detail
Tên usecase	Xem chi tiết Farm
Mô tả	Admin xem thông tin chi tiết của một Farm trong hệ thống
Actor	System Admin
Tiền điều kiện	<ul style="list-style-type: none">• Admin đã đăng nhập• Danh sách Farm đang được hiển thị
Hậu điều kiện	Thông tin chi tiết của Farm được hiển thị
Trigger	Admin nhấn chọn một Farm trong danh sách
Luồng chính	<ol style="list-style-type: none">1. Admin xem danh sách Farm.2. Admin nhấn chọn một Farm bất kỳ.3. Hệ thống kiểm tra quyền truy cập của Admin.4. Hệ thống truy vấn thông tin chi tiết của Farm.5. Hệ thống hiển thị màn hình chi tiết Farm.
Quy tắc nghiệp vụ	<ul style="list-style-type: none">• BR-01 (Authorization): Chỉ tài khoản có vai trò System Admin mới được phép xem chi tiết Farm.• BR-02 (Data Integrity): Farm phải tồn tại trong hệ thống tại thời điểm truy vấn.
Luồng thay thế / Mở rộng	<ul style="list-style-type: none">• E-01 (Back to List): Tại màn hình chi tiết, Admin nhấn nút mũi tên “Back” ở góc trên bên trái → Hệ thống quay về màn hình danh sách Farm.

4.6.2 Use case: Giám sát Thiết bị



(a) Sơ đồ quản lý thiết bị



Mã số usecase	UC-01: View device list
Tên usecase	Xem danh sách thiết bị
Mô tả	Admin xem danh sách các thiết bị trong hệ thống
Actor	System Admin
Tiền điều kiện	Admin đã đăng nhập và có quyền truy cập module Device
Hậu điều kiện	Danh sách thiết bị được hiển thị
Trigger	Admin chọn menu “Device”
Luồng chính	<ol style="list-style-type: none">Admin truy cập module Device.Hệ thống kiểm tra quyền truy cập.Hệ thống hiển thị danh sách thiết bị.
Quy tắc nghiệp vụ	<ul style="list-style-type: none">BR-01 (Authorization): Chỉ System Admin được phép xem danh sách thiết bị.

Mã số usecase	UC-02: View device detail
Tên usecase	Xem chi tiết thiết bị
Mô tả	Admin xem thông tin chi tiết của một thiết bị
Actor	System Admin
Tiền điều kiện	Danh sách thiết bị đã được hiển thị
Hậu điều kiện	Thông tin chi tiết thiết bị được hiển thị
Trigger	Admin chọn một thiết bị trong danh sách
Luồng chính	<ol style="list-style-type: none">Admin nhấn chọn một thiết bị.Hệ thống tải dữ liệu chi tiết thiết bị.Hệ thống hiển thị màn hình chi tiết thiết bị.
Luồng thay thế / Mở rộng	<ul style="list-style-type: none">E-01 (Back): Admin nhấn nút “Back” → hệ thống quay lại danh sách thiết bị.

Mã số usecase	UC-03: Add device
Tên usecase	Thêm thiết bị
Mô tả	Admin tạo mới một thiết bị
Actor	System Admin
Tiền điều kiện	Admin có quyền quản lý thiết bị
Hậu điều kiện	Thiết bị mới được lưu và hiển thị trong danh sách
Trigger	Admin nhấn nút “Add Device”
Luồng chính	<ol style="list-style-type: none">Admin mở màn hình Device.Admin chọn “Add Device”.Hệ thống hiển thị form nhập thông tin.Admin nhập thông tin và nhấn “Save”.Hệ thống lưu thiết bị và cập nhật danh sách.
Quy tắc nghiệp vụ	<ul style="list-style-type: none">BR-01: Các trường bắt buộc không được để trống.

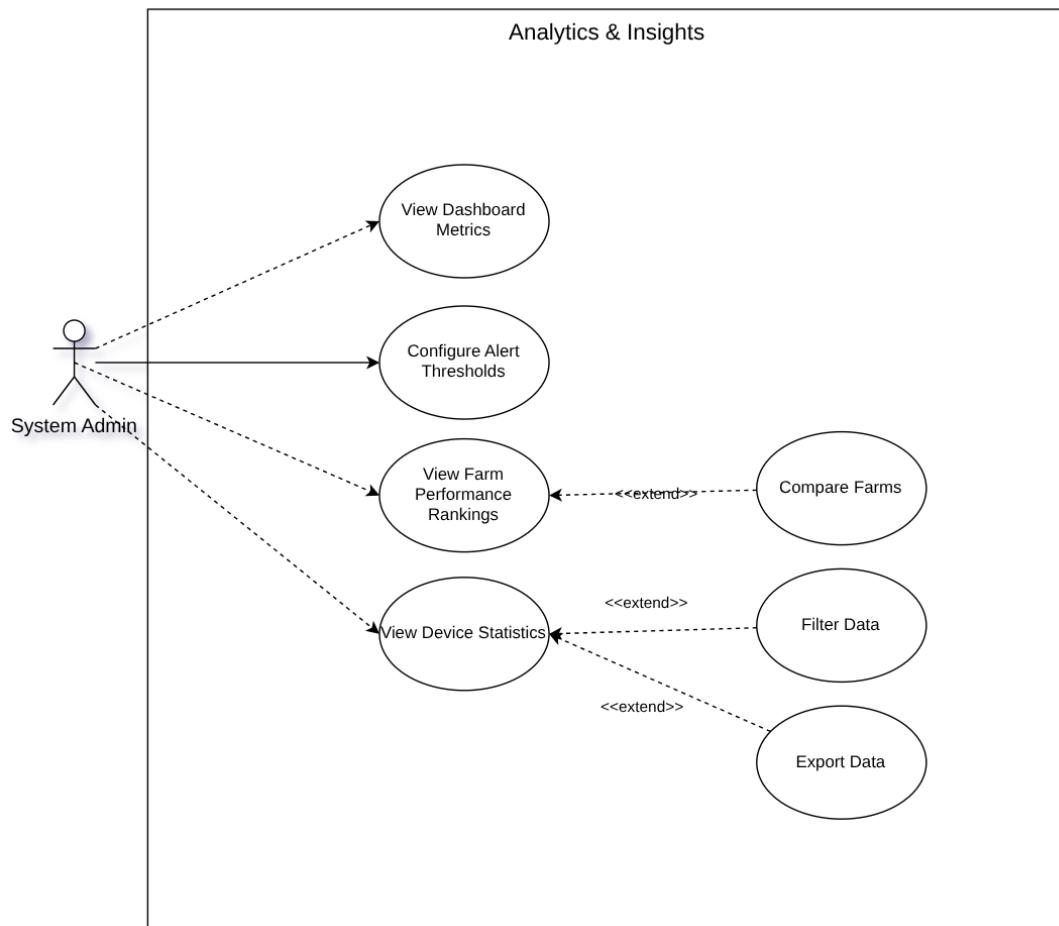


Mã số usecase	UC-04: Edit device
Tên usecase	Chỉnh sửa thiết bị
Mô tả	Admin cập nhật thông tin thiết bị
Actor	System Admin
Tiền điều kiện	Thiết bị đã tồn tại
Hậu điều kiện	Thông tin thiết bị được cập nhật
Trigger	Admin chọn “Edit” tại thiết bị
Luồng chính	<ol style="list-style-type: none">Admin chọn thiết bị cần chỉnh sửa.Hệ thống hiển thị form chỉnh sửa.Admin cập nhật thông tin và nhấn “Save”.Hệ thống lưu thay đổi.
Quy tắc nghiệp vụ	<ul style="list-style-type: none">BR-01 (Authorization): Chỉ tài khoản có vai trò System Admin mới được phép chỉnh sửa thiết bị.BR-02 (Data Integrity): Thiết bị phải tồn tại trong hệ thống tại thời điểm chỉnh sửa.BR-03 (Data Validation): Dữ liệu chỉnh sửa phải đúng định dạng và nằm trong phạm vi cho phép của hệ thống.
Luồng thay thế / Mở rộng	<ul style="list-style-type: none">E-01 (Cancel Edit): Tại bước 3–4, Admin nhấn nút “Cancel” hoặc “Back” → Hệ thống hủy thao tác chỉnh sửa và không lưu dữ liệu.

Mã số usecase	UC-05: Delete device
Tên usecase	Xóa thiết bị
Mô tả	Admin xóa thiết bị khỏi hệ thống
Actor	System Admin
Tiền điều kiện	Thiết bị tồn tại trong hệ thống
Hậu điều kiện	Thiết bị bị xóa khỏi danh sách
Trigger	Admin chọn “Delete” tại thiết bị
Luồng chính	<ol style="list-style-type: none">Admin chọn thiết bị cần xóa.Hệ thống hiển thị hộp thoại xác nhận.Admin xác nhận xóa.Hệ thống xóa thiết bị.
Quy tắc nghiệp vụ	<ul style="list-style-type: none">BR-01 (Authorization): Chỉ tài khoản có vai trò System Admin mới được phép xóa thiết bị.BR-02 (Data Integrity): Thiết bị phải tồn tại trong hệ thống tại thời điểm xóa.
Luồng thay thế / Mở rộng	<ul style="list-style-type: none">E-01 (Cancel Delete): Tại bước xác nhận, Admin chọn “Cancel” → Hệ thống hủy thao tác xóa thiết bị.E-02 (Delete Not Allowed): Nếu thiết bị còn sensor liên kết → Hệ thống hiển thị thông báo không thể xóa thiết bị.

Mã số usecase	UC-06: Import/Export device
Tên usecase	Import / Export thiết bị
Mô tả	Admin nhập hoặc xuất danh sách thiết bị
Actor	System Admin
Tiền điều kiện	Admin có quyền quản lý thiết bị
Hậu điều kiện	Dữ liệu thiết bị được nhập hoặc xuất thành công
Trigger	Admin chọn “Import/Export Device”
Luồng chính	<ol style="list-style-type: none"> 1. Admin chọn chức năng Import hoặc Export. 2. Hệ thống xử lý dữ liệu. 3. Hệ thống thông báo kết quả.

4.6.3 Use case: Giám sát và Phân tích Hệ thống

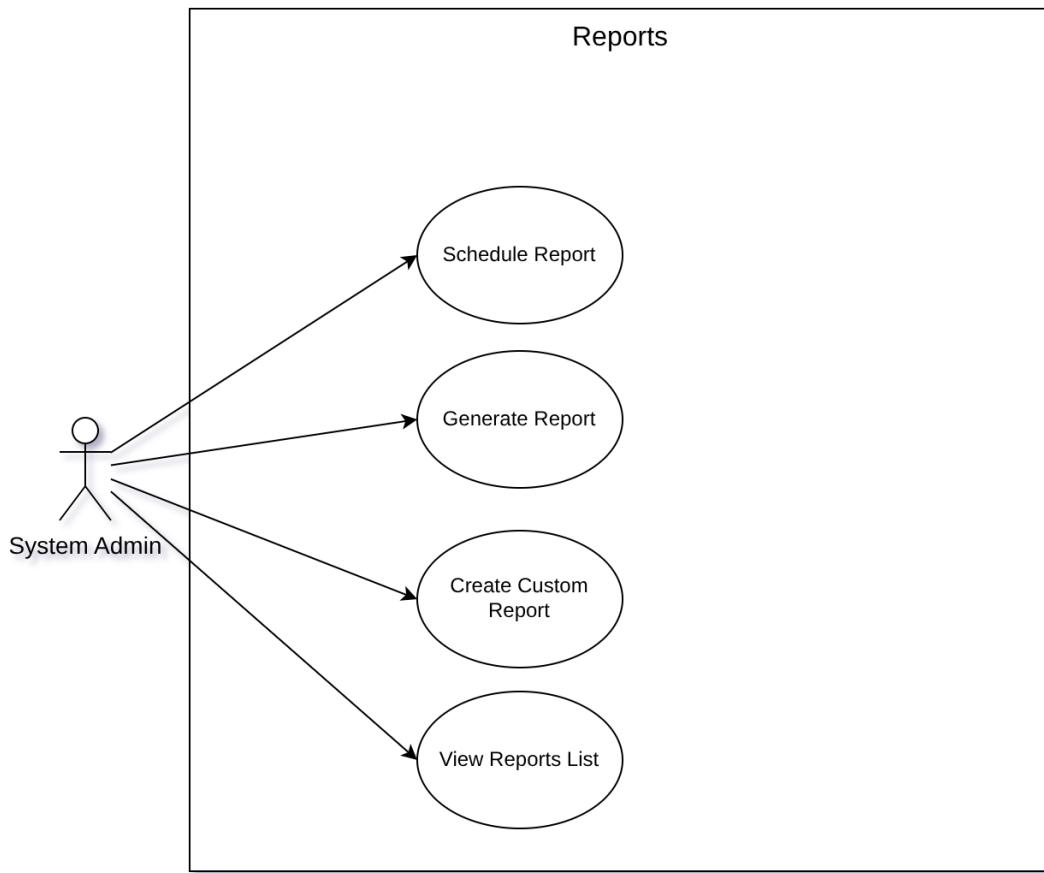


Hình 5: Sơ đồ luồng quản lý và tạo báo cáo



Mã số usecase	UC-01: Giám sát và Phân tích Hệ thống
Tên usecase	Giám sát Phân tích Hệ thống
Mô tả	Admin theo dõi tổng quan các chỉ số hoạt động, xếp hạng hiệu suất nông trại và thống kê thiết bị trên Dashboard.
Actor	System Admin
Tiền điều kiện	Tài khoản Admin đã đăng nhập và có quyền truy cập module Giám sát.
Hậu điều kiện	Dữ liệu thống kê được hiển thị đầy đủ và cập nhật mới nhất.
Trigger	Admin nhấn chọn menu “Giám sát & Thống kê” trên thanh điều hướng.
Luồng chính	<ol style="list-style-type: none">Admin truy cập màn hình Analytics.Hệ thống kiểm tra quyền xem của Admin.Hệ thống tải dữ liệu tổng hợp từ cơ sở dữ liệu.Hệ thống hiển thị Dashboard Metrics (doanh thu, sản lượng, nhiệt độ trung bình).Hệ thống hiển thị Farm Performance Rankings (xếp hạng hiệu suất).Hệ thống hiển thị Device Statistics (biểu đồ trạng thái Online/Offline).
Quy tắc nghiệp vụ	<ul style="list-style-type: none">BR-01 (Data Latency): Dữ liệu hiển thị trên Dashboard phải được cập nhật gần thời gian thực (Real-time), độ trễ tối đa không quá 30 giây.BR-02 (Default View): Mặc định hiển thị dữ liệu tổng hợp của toàn hệ thống trong 7 ngày gần nhất.BR-03 (Access Control): Chỉ tài khoản Admin có quyền “Manage” mới hiển thị nút cấu hình “Configure Alert Thresholds”.BR-04 (Ranking Logic): Xếp hạng nông trại dựa trên chỉ số KPI tổng hợp (tỷ lệ sản lượng / mức tiêu thụ năng lượng).
Luồng thay thế / Mở rộng	<ul style="list-style-type: none">E-01 (Filter): Tại bước 4-6, Admin chọn bộ lọc thời gian hoặc khu vực → Hệ thống truy vấn lại và cập nhật hiển thị.E-02 (Export): Admin bấm “Export” → Hệ thống kiểm tra định dạng file (PDF/CSV) và tiến hành tải xuống.E-03 (Alert Config): Admin thay đổi ngưỡng cảnh báo → Hệ thống lưu quy tắc mới vào cơ sở dữ liệu và áp dụng ngay lập tức.

4.6.4 Use case: Quản lý Báo cáo



Hình 6: Sơ đồ luồng quản lý và tạo báo cáo

Hình thể hiện quy trình xem danh sách báo cáo, tạo mới, lập lịch hoặc tạo báo cáo tùy chỉnh và cập nhật lại danh sách.



Mã số usecase	UC-03: Manage Reports
Tên usecase	Quản lý và tạo báo cáo
Mô tả	Admin xem danh sách báo cáo đã lưu, tạo báo cáo mới tức thì hoặc lập lịch gửi báo cáo tự động.
Actor	System Admin
Tiền điều kiện	Admin đã đăng nhập thành công.
Hậu điều kiện	Danh sách báo cáo được cập nhật; file báo cáo được tạo ra.
Trigger	Admin truy cập trang “Reports”.
Luồng chính	<ol style="list-style-type: none">Admin mở trang Reports.Hệ thống truy vấn cơ sở dữ liệu báo cáo.Hệ thống hiển thị danh sách Recent Reports (bao gồm: Tên, Ngày tạo, Người tạo, Loại báo cáo).Admin xem thông tin hoặc tải về các báo cáo cũ.
Quy tắc nghiệp vụ	<ul style="list-style-type: none">BR-08 (Retention Policy): Báo cáo chỉ được lưu trữ trực tuyến trong 90 ngày. Sau thời gian này, dữ liệu sẽ được chuyển sang lưu trữ lạnh (Archive).BR-09 (File Format): Hệ thống phải hỗ trợ xuất báo cáo ra hai định dạng: PDF (để in ấn/trình ký) và Excel/CSV (để phân tích số liệu).BR-10 (Schedule Limit): Mỗi tài khoản Admin chỉ được thiết lập tối đa 5 lịch báo cáo tự động để đảm bảo hiệu năng hệ thống.
Luồng thay thế / Mở rộng	<ul style="list-style-type: none">E-01 (Generate): Admin bấm “Generate New” → Hệ thống thu thập dữ liệu hiện tại và tạo file báo cáo ngay lập tức.E-02 (Schedule): Admin bấm “Schedule” → Hệ thống hiển thị form chọn tần suất (hàng ngày/tuần) → Lưu lịch chạy tự động (Cron job).E-03 (Custom Report): Admin chọn các trường dữ liệu tùy chỉnh → Bấm “Create” → Hệ thống tạo báo cáo theo mẫu riêng.



5 Giải pháp đề xuất

5.1 Mục tiêu của giải pháp

Giải pháp được đề xuất nhằm xây dựng một hệ thống giám sát nông nghiệp thông minh dựa trên công nghệ Internet vạn vật (IoT), với mục tiêu chính là cung cấp một hệ thống quản lý tích hợp cho các nông trại hiện đại. Hệ thống được thiết kế để đáp ứng các mục tiêu cụ thể sau:

5.1.1 Mục tiêu 1: Tự động hóa quá trình thu thập và xử lý dữ liệu môi trường

Hệ thống phải có khả năng tự động thu thập dữ liệu từ mạng lưới cảm biến phân tán (nhiệt độ, độ ẩm không khí, độ ẩm đất, ánh sáng) và xử lý chúng theo thời gian thực mà không cần sự can thiệp thủ công. Mục tiêu này nhằm giảm thiểu công sức quản lý của người nông dân, đồng thời đảm bảo tính liên tục và chính xác của dữ liệu giám sát.

5.1.2 Mục tiêu 2: Phát hiện và cảnh báo sớm các sự cố hệ thống

Hệ thống phải tích hợp các cơ chế phát hiện lỗi tự động (như mất kết nối thiết bị, dữ liệu bất thường, hoặc giá trị vượt ngưỡng an toàn) và gửi cảnh báo tức thì đến người quản trị. Mục tiêu này giúp giảm thiểu thời gian phản ứng khi xảy ra sự cố, từ đó hạn chế thiệt hại về năng suất cây trồng và tài sản.

5.1.3 Mục tiêu 3: Cung cấp giao diện quản lý trực quan và dễ sử dụng

Hệ thống phải cung cấp một Dashboard Web Admin với các tính năng trực quan hóa dữ liệu (biểu đồ, bảng thống kê) và các thao tác quản lý tiện lợi, giúp người quản trị nắm bắt tình trạng hệ thống một cách nhanh chóng và thực hiện các tác vụ quản lý hiệu quả.

5.1.4 Mục tiêu 4: Đảm bảo khả năng mở rộng và hiệu năng cao

Hệ thống phải được thiết kế với kiến trúc có khả năng mở rộng (Scalable Architecture), cho phép thêm mới thiết bị hoặc nông trại mà không cần tắt server để bảo trì. Đồng thời, hệ thống phải đảm bảo xử lý ổn định dữ liệu từ hàng chục thiết bị đồng thời với độ trễ thấp.

5.1.5 Mục tiêu 5: Tối ưu hóa chi phí vận hành và bảo trì

Hệ thống phải sử dụng các công nghệ mã nguồn mở và kiến trúc tiết kiệm tài nguyên, giúp giảm thiểu chi phí triển khai và vận hành. Việc sử dụng các giao thức nhẹ như MQTT và cơ sở dữ liệu chuyên dụng cho Time-series (TimescaleDB) nhằm tối ưu hóa hiệu năng xử lý dữ liệu lớn.

5.1.6 Mục tiêu 6: Đảm bảo tính bảo mật và độ tin cậy

Hệ thống phải tích hợp các cơ chế bảo mật đa lớp (xác thực người dùng, phân quyền truy cập, mã hóa dữ liệu) và đảm bảo thời gian hoạt động (Uptime) đạt 99%, với khả năng tự động kết nối lại khi mất kết nối mạng.



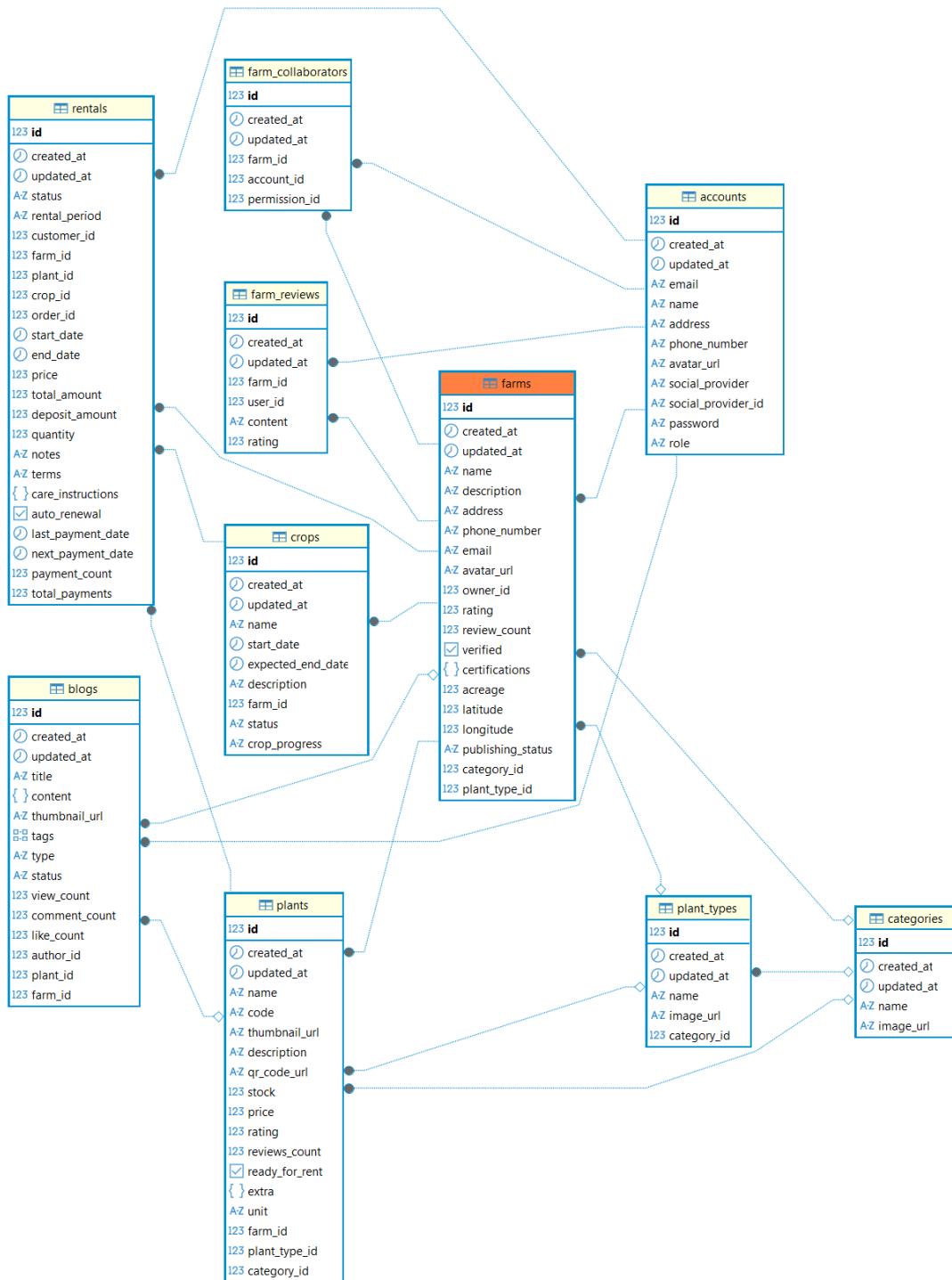
5.2 Thiết kế kiến trúc (Mức khái niệm)

Mô hình Quan hệ Thực thể (ERD) chi tiết của hệ thống được trình bày tại Phụ lục A. Thiết kế này đảm bảo tính toàn vẹn dữ liệu và khả năng mở rộng cho các nghiệp vụ giám sát nông nghiệp thông minh.

Hệ thống cơ sở dữ liệu được tổ chức thành các phân hệ chính như sau:

5.2.1 Phân hệ Quản lý Nông trại:

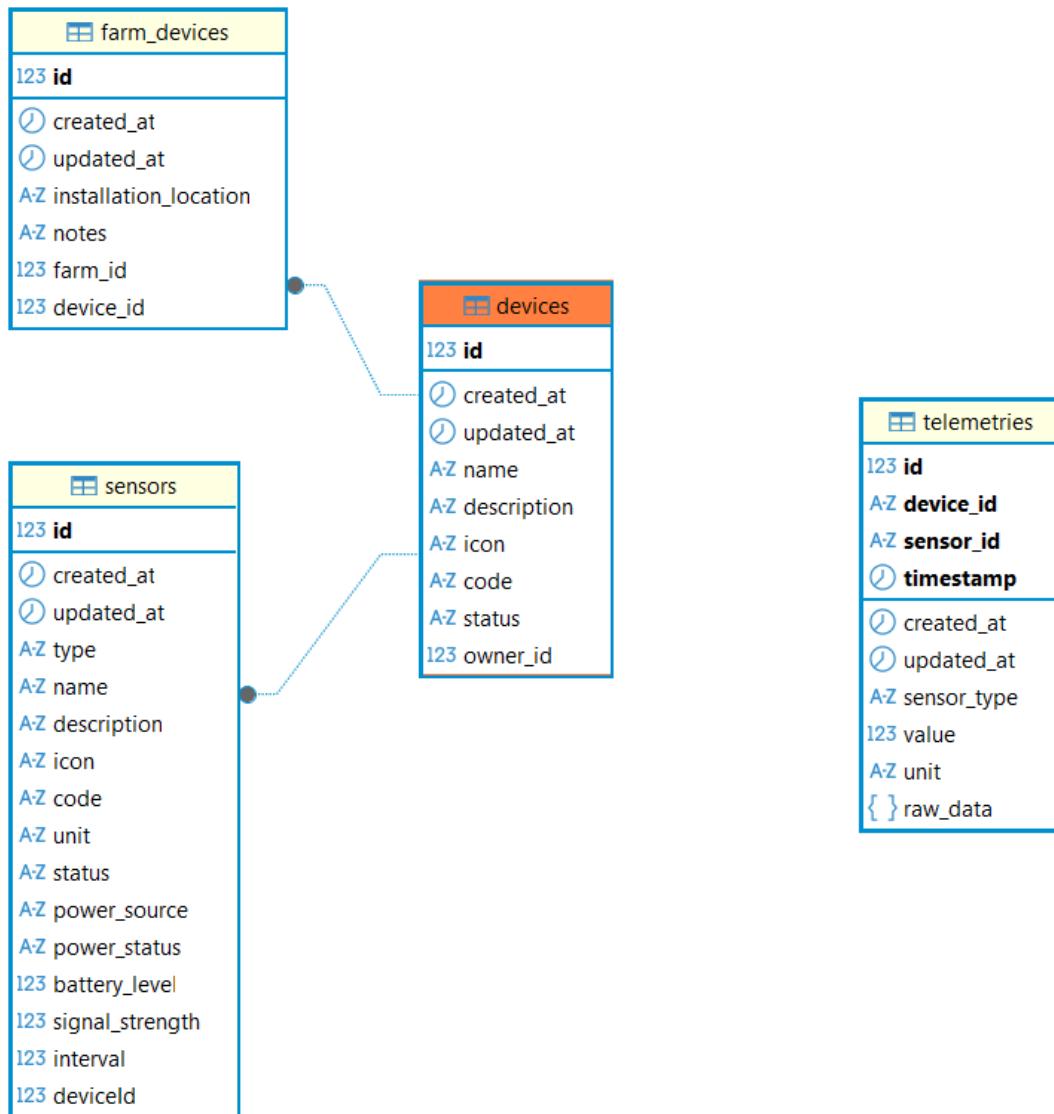
Đây là nhóm thực thể trung tâm, bao gồm bảng **farms** (lưu thông tin nông trại) liên kết 1-nhiều với **plants** (cây trồng) và **crops** (mùa vụ). Cấu trúc này cho phép quản lý chi tiết quy trình canh tác từ lúc xuồng giống đến khi thu hoạch, bao gồm cả việc theo dõi sức khỏe cây trồng qua bảng **plant_health_statuses**.



Hình 7: Sơ đồ thực thể quan hệ của phân hệ Quản lý Nông trại

5.2.2 Phân hệ IoT và Thu thập dữ liệu:

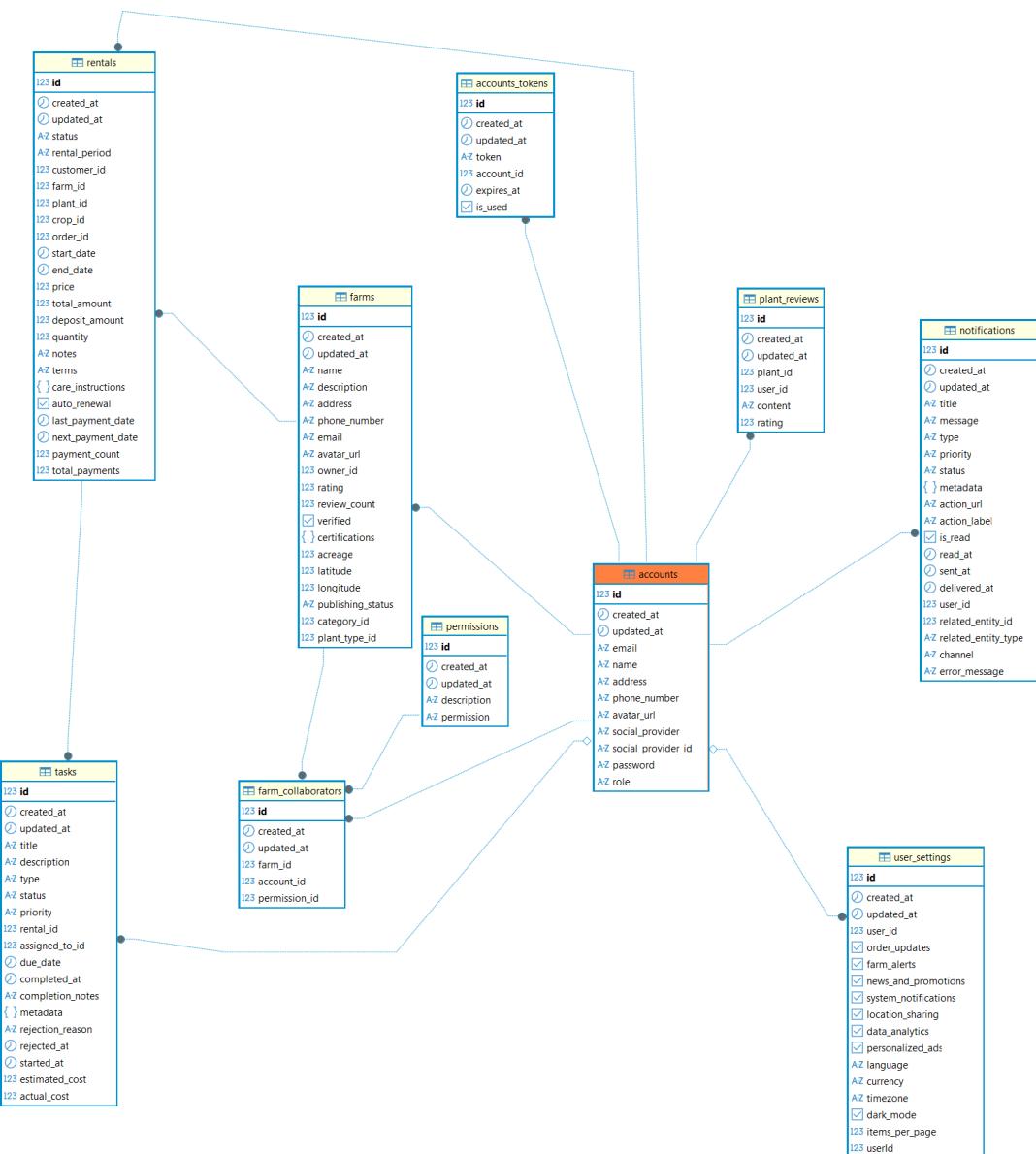
Các thiết bị phần cứng được quản lý qua bảng **devices** và **sensors**. Dữ liệu quan trọng nhất của hệ thống là **telemtries**, nơi lưu trữ hàng triệu bản ghi dữ liệu cảm biến (nhiệt độ, độ ẩm, ánh sáng) được gửi về liên tục. Quan hệ giữa **devices** và **farms** giúp xác định thiết bị nào đang hoạt động tại khu vực nào.



Hình 8: Sơ đồ thực thể quan hệ của phân hệ IoT và Thu thập dữ liệu

5.2.3 Phân hệ Người dùng và Phân quyền:

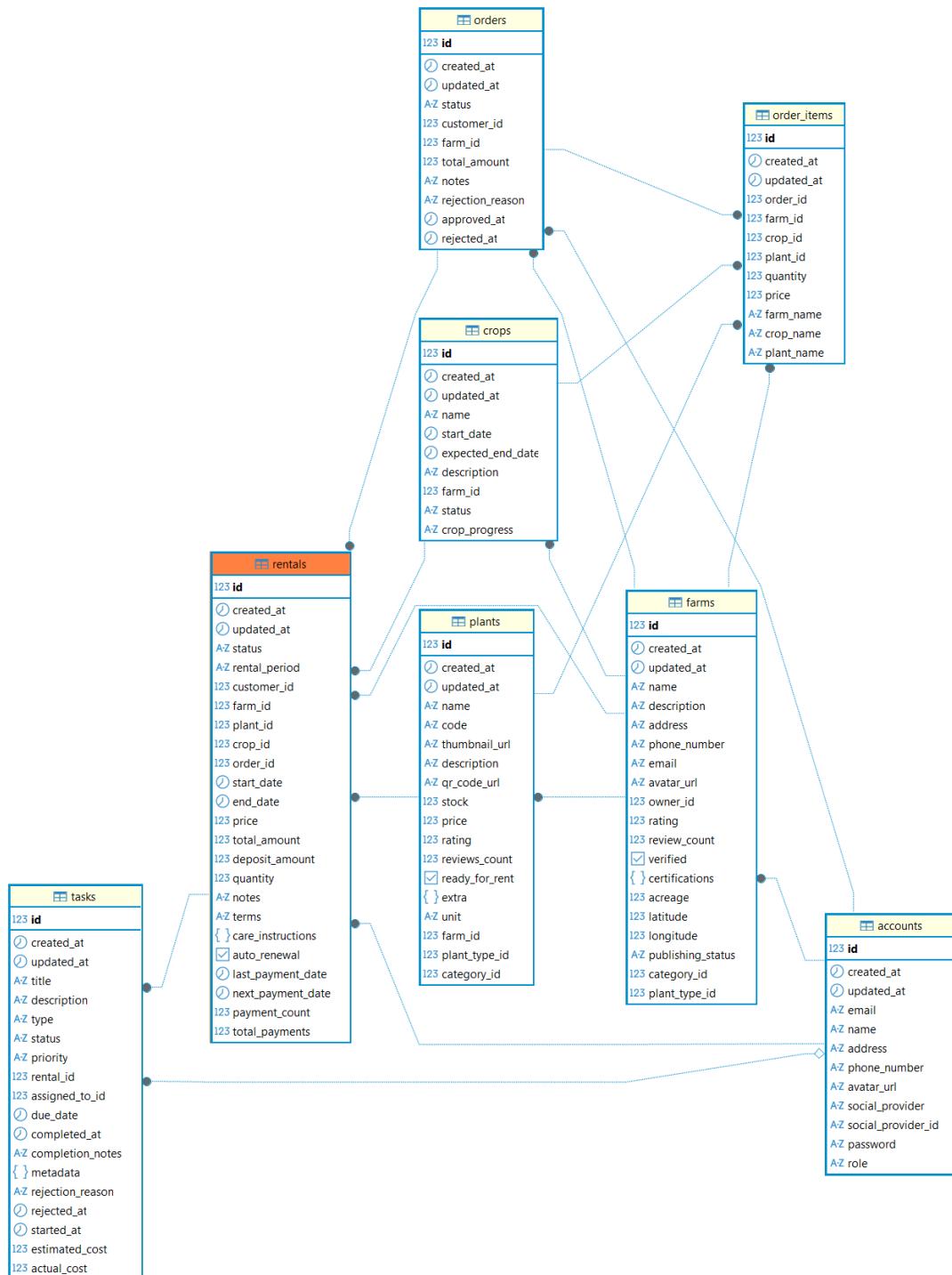
Hệ thống sử dụng bảng `accounts` để quản lý người dùng, kết hợp với `permissions` và `user_settings` để thực hiện cơ chế phân quyền, đảm bảo chỉ Admin hoặc chủ nông trại mới có quyền truy cập các dữ liệu nhạy cảm hoặc thực hiện cấu hình thiết bị.



Hình 9: Sơ đồ thực thể quan hệ của phân hệ Người dùng và Phân quyền

5.2.4 Phân hệ Kinh doanh

Các nghiệp vụ kinh doanh như thuê thiết bị, đặt hàng, thanh toán được quản lý thông qua bảng `rentals` và `orders`.



Hình 10: Sơ đồ thực thể quan hệ của phân hệ Kinh doanh

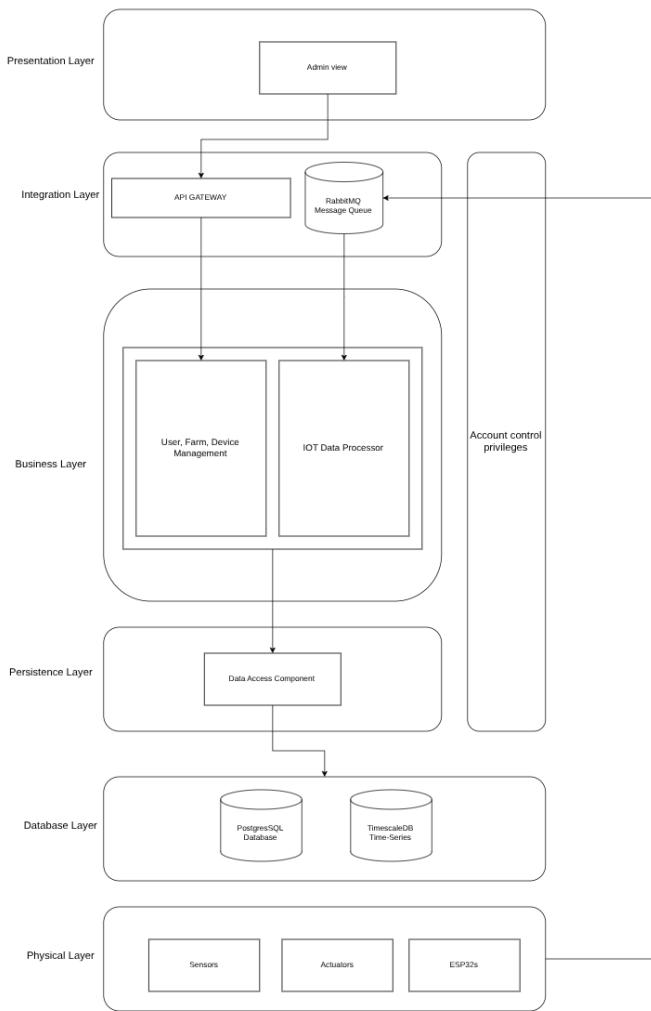
Và các bảng khác được trình bày tại Phụ lục A.

5.3 Thiết kế Kiến trúc Hệ thống

5.3.1 Kiến trúc Phân lớp (Layered Architecture)

Kiến trúc hệ thống được thiết kế theo mô hình Phân lớp (Layered Architecture) kết hợp với hướng dịch vụ (Service-oriented). Việc phân chia này giúp tách biệt các mối quan tâm (Separation of Concerns), dễ dàng bảo trì và mở rộng độc lập từng thành phần.

Sơ đồ dưới đây minh họa các tầng logic và luồng dữ liệu trong hệ thống:



Hình 11: Sơ đồ Kiến trúc Phân lớp của Hệ thống

Hệ thống được tổ chức thành 6 tầng chính và 1 module bảo mật xuyên suốt:

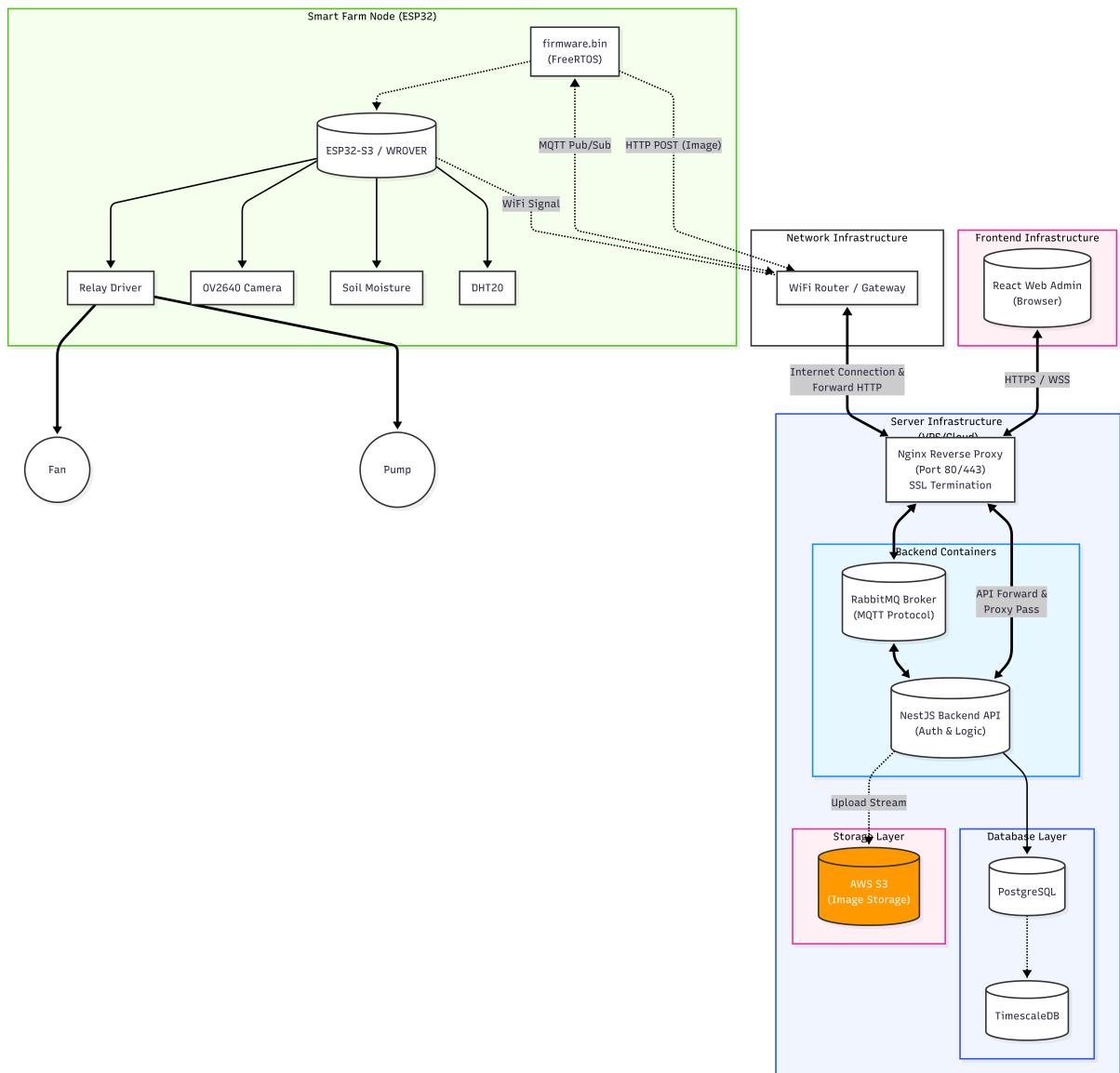
- **Presentation Layer (Tầng Giao diện):** Là điểm tiếp xúc với người dùng cuối (System Admin). Tầng này gửi các yêu cầu HTTP đến hệ thống thông qua giao diện Web Admin để thực hiện các tác vụ quản lý.
 - **Integration Layer (Tầng Tích hợp):** Dóng vai trò là cổng vào duy nhất (Entry Point) và điều phối luồng dữ liệu, bao gồm 2 thành phần:
 - *API Gateway:* Tiếp nhận và định tuyến các yêu cầu RESTful từ Web Admin xuông các dịch vụ nghiệp vụ tương ứng.
 - *RabbitMQ Message Queue:* Dóng vai trò Broker trung gian, tiếp nhận hàng loạt dữ liệu bất đồng bộ từ các thiết bị IoT (Sensors/ESP32), giúp giảm tải cho hệ thống xử lý chính (Decoupling).
 - **Business Layer (Tầng Nghiệp vụ):** Nơi chứa toàn bộ logic xử lý của hệ thống:
 - *User & Farm Management:* Xử lý các logic về quản lý người dùng, nông trại, và cấu hình thiết bị.



- **IoT Data Processor:** Service chuyên biệt (Worker) để tiêu thụ dữ liệu từ RabbitMQ, xử lý tính toán, cảnh báo và chuẩn hóa dữ liệu trước khi lưu trữ.
- **Persistence Layer (Tầng Truy xuất Dữ liệu):** Cung cấp lớp trừu tượng hóa (Abstraction) để giao tiếp với cơ sở dữ liệu (Data Access Component), giúp tách biệt logic nghiệp vụ khỏi các câu lệnh truy vấn SQL cụ thể.
- **Database Layer (Tầng Dữ liệu):** Sử dụng chiến lược lưu trữ đa mô hình (Polyglot Persistence):
 - *PostgreSQL:* Lưu trữ dữ liệu quan hệ có cấu trúc (Users, Farms, Devices).
 - *TimescaleDB:* Cơ sở dữ liệu chuyên dụng cho Time-series để lưu trữ hàng triệu bản ghi nhật ký cảm biến (Telemetries) với hiệu năng cao.
 - *AWS S3:* Lưu trữ các file tĩnh (hình ảnh cây trồng, log files).
- **Physical Layer (Tầng Vật lý):** Bao gồm các thiết bị phần cứng (ESP32, Sensors, Actuators) thu thập dữ liệu môi trường và thực thi các lệnh điều khiển từ server.

5.3.1.1 Module Bảo mật (Cross-cutting Concern): Thành phần *Account Control Privileges* bao quát từ tầng Integration xuống tầng Business, đảm bảo mọi yêu cầu đi qua API Gateway đều được xác thực (Authentication) và phân quyền (Authorization) chặt chẽ trước khi được xử lý.

5.3.2 Kiến trúc Triển khai (Deployment Architecture)



Hình 12: Deployment Diagram

Sơ đồ triển khai (Hình 12) mô tả cách thức ánh xạ các thành phần phần mềm vào hạ tầng phần cứng vật lý và môi trường mạng. Hệ thống được triển khai theo mô hình tập trung (Centralized Cloud Architecture) kết hợp với các thiết bị biên (Edge Devices).

Các thành phần triển khai chính bao gồm:

- **Field Node (Nút thiết bị tại nông trại):**

- *Vị trí điều khiển:* Sử dụng ESP32-S3 chạy Firmware dựa trên hệ điều hành thời gian thực FreeRTOS, đảm bảo khả năng xử lý đa nhiệm (đọc cảm biến và gửi mạng song song).
- *Kết nối vật lý:* Các cảm biến (DHT20, Soil Moisture) và camera (OV2640) được kết nối trực tiếp qua các giao tiếp I2C, Analog và DVP. Các thiết bị chấp hành (Máy bơm, Quạt) được điều khiển qua Module Relay.

- **Server Infrastructure (Hệ thống Backend):** Toàn bộ hệ thống Backend được đóng gói và triển khai trên nền tảng Docker, giúp đảm bảo tính nhất quán giữa môi trường phát triển và vận hành (Production). Hệ thống bao gồm các Container chính:



- *Nginx Reverse Proxy*: Dóng vai trò lớp bảo mật đầu tiên, tiếp nhận mọi yêu cầu từ Internet (Port 80/443), chấm dứt SSL (SSL Termination) và điều phối traffic vào các dịch vụ bên trong.
- *Application Container*: Chạy NestJS Backend để xử lý API và logic nghiệp vụ.
- *Message Broker Container*: Chạy RabbitMQ để tiếp nhận dữ liệu IoT qua giao thức MQTT.
- *Database Cluster*: Chạy PostgreSQL tích hợp Extension TimescaleDB, phục vụ lưu trữ dữ liệu lai (Hybrid Storage).

- **Cơ chế Giao tiếp (Communication Protocols):**

- *Device - Server*: Sử dụng giao thức **MQTT** qua WiFi/4G để truyền tải dữ liệu cảm biến (nhẹ, thời gian thực) và giao thức **HTTP** để tải lên hình ảnh (dữ liệu lớn).
- *Client - Server*: Web Admin giao tiếp với hệ thống thông qua **HTTPS** (bảo mật) và **Web-Socket** (để nhận cập nhật dữ liệu thời gian thực trên Dashboard).

5.3.2.1 Đánh đổi kiến trúc và Rủi ro Mặc dù kiến trúc phân lớp mang lại nhiều lợi ích về tính mô-đun và khả năng mở rộng, việc triển khai đòi hỏi đánh đổi về độ phức tạp. Nhóm đã cân nhắc kỹ lưỡng giữa lợi ích dài hạn (scalability, maintainability) và chi phí triển khai ban đầu. Chi tiết về phân tích đánh đổi được trình bày ở phần riêng sau.

5.4 Phân tích đánh đổi kiến trúc và Quản lý rủi ro

Việc lựa chọn kiến trúc phân lớp (Layered Architecture) kết hợp với các thành phần Microservices (như Worker tách rời) và Polyglot Persistence (đa cơ sở dữ liệu) mang lại khả năng mở rộng lớn, nhưng đồng thời cũng đặt ra bài toán về độ phức tạp khi triển khai. Dưới đây là phân tích chi tiết các đánh đổi (Trade-offs) và biện pháp giảm thiểu rủi ro của nhóm.

5.5 Tại sao không chọn kiến trúc Monolithic đơn giản?

Mặc dù kiến trúc nguyên khối (Monolithic) đơn giản, dễ triển khai (chỉ cần 1 Server + 1 DB), nhưng nhóm quyết định chọn kiến trúc phân tán hướng dịch vụ vì các lý do sau:

- **Khả năng mở rộng độc lập (Scalability)**: Module xử lý dữ liệu IoT (Worker) cần tài nguyên CPU cao để chạy thuật toán RFE, trong khi Web Server phục vụ API chỉ cần ít tài nguyên. Việc tách rời giúp nhóm có thể nhân bản (scale) Worker mà không ảnh hưởng đến Web API.
- **Phù hợp làm việc nhóm**: Với 3 thành viên (Firmware, Backend, Frontend/AI), kiến trúc phân lớp giúp phân chia ranh giới công việc rõ ràng (Decoupling), tránh việc code chồng chéo (Merge conflicts) quá nhiều.
- **Chuẩn bị cho Đò án Tốt nghiệp**: Đò án chuyên ngành là bước đệm. Việc xây dựng khung kiến trúc chuẩn ngay từ đầu giúp giai đoạn 2 chỉ cần tập trung phát triển tính năng mà không phải đập đi xây lại (Refactoring).



5.6 Dánh đổi kỹ thuật

Bảng 8: Bảng phân tích đánh đổi khi lựa chọn công nghệ

Quyết định kiến trúc	Lợi ích (Pros)	Cái giá phải trả (Cons)
Sử dụng Message Queue (RabbitMQ)	Dảm bảo tính bất đồng bộ, không mất dữ liệu khi Server quá tải (Back-pressure), giảm độ trễ phản hồi API.	Tăng độ phức tạp vận hành. Phải quản lý thêm một service Broker, khó debug luồng dữ liệu hơn so với gọi HTTP trực tiếp.
Polyglot Persistence (TimescaleDB + PostgreSQL)	Tối ưu hóa truy vấn chuỗi thời gian (nhanh hơn 10x), nén dữ liệu tốt.	Phải quản lý schema phức tạp hơn. Cần kiến thức chuyên sâu về cả Relational và Time-series DB.
Containerization (Docker)	Môi trường đồng nhất giữa Dev và Prod, dễ dàng triển khai.	Tốn tài nguyên hệ thống (RAM/CPU) hơn so với chạy Native trên máy ảo đơn giản.

5.7 Rủi ro và Chiến lược giảm thiểu

Với quy mô của một đồ án sinh viên, việc vận hành hệ thống phức tạp tiềm ẩn nhiều rủi ro. Nhóm đã xác định và có phương án cụ thể:

Rủi ro 1: Quá tải về mặt triển khai (DevOps Overhead)

Việc cấu hình Nginx, Docker, RabbitMQ tốn nhiều thời gian.

→ **Giải pháp:** Nhóm sử dụng Docker Compose để đóng gói toàn bộ môi trường (Infrastructure as Code). Chỉ cần 1 lệnh docker-compose up là dựng được toàn bộ hệ thống, giúp thành viên phát triển Frontend/Firmware không cần cài đặt phức tạp.

Rủi ro 2: Khó kiểm soát lỗi (Debugging)

Luồng dữ liệu đi qua nhiều trạm (MQTT → RabbitMQ → NestJS → DB) khiến việc tìm nguyên nhân lỗi khó khăn.

→ **Giải pháp:** Nhóm xây dựng cơ chế Centralized Logging đơn giản, ghi log tập trung tại mỗi điểm nút quan trọng để truy vết (Traceability).

Kết luận: Kiến trúc này tuy có độ phức tạp cao hơn mức trung bình của một đồ án sinh viên, nhưng là sự đầu tư cần thiết để giải quyết bài toán "Real-time" và "Big Data" của Nông nghiệp thông minh, đồng thời tiệm cận với quy chuẩn thực tế của doanh nghiệp.

5.8 Đề xuất hiện thực và lựa chọn công nghệ

5.8.1 Quản lý lược đồ dữ liệu cảm biến từ xa

- Mục tiêu:** đảm bảo rằng mọi dữ liệu (telemetry) gửi từ hàng ngàn cảm biến về máy chủ đều:
 - Thống nhất (Consistent):** Dữ liệu luôn tuân theo một định dạng chuẩn.
 - Chất lượng (Valid):** Dữ liệu không bị sai, thiếu hoặc lỗi định dạng.
 - Dễ dàng mở rộng (Extensible):** Dễ dàng thêm cảm biến mới hoặc phiên bản firmware mới mà không làm sập hệ thống.
 - Dễ bảo trì (Maintainable):** Khi schema thay đổi (ví dụ: thêm cảm biến đo độ pH), máy chủ biết cách xử lý phiên bản cũ và mới.
- Đề xuất cấu trúc:** Sử dụng định dạng JSON vì tính linh hoạt và dễ tiếp cận. Cấu trúc được chia thành 3 schema chính:
 - Schema chung (device_base):** Tất cả các gói tin (packet) đều phải chứa các thông tin cơ bản này để định tuyến và nhận diện.



- * `device_id(String)`: Mã định danh thiết bị.
- * `timestamp(int64)`: Thời gian gửi dữ liệu.
- * `schema_id(String)`: Tên của schema mà gói tin này đang sử dụng.
- * `schema_version(String)`: Phiên bản của schema.

– **Schema 1 (env_data):** Dữ liệu môi trường.

```
{  
    "device_id": "env-sensor-zone-a-01",  
    "timestamp": 1678886400000,  
    "schema_id": "env_data",  
    "schema_version": "v1.1",  
    "data": {  
        "temperature_celsius": 28.5,  
        "humidity_percent": 75.2  
    }  
}
```

– **Schema 2 (camera_data):** Dữ liệu hình ảnh từ camera.

```
{  
    "device_id": "cam-zone-a-01",  
    "timestamp": 1678887000000,  
    "schema_id": "camera_event",  
    "schema_version": "v1.0",  
    "data": {  
        "image_url": "https://storage.server.com/images/12345.jpg",  
        "file_size": 51200  
    }  
}
```

– **Schema 3 (device_health):** Dữ liệu tình trạng thiết bị.

```
{  
    "device_id": "cam-zone-a-01",  
    "timestamp": 1678886500000,  
    "schema_id": "device_health",  
    "schema_version": "v1.0",  
    "data": {  
        "status": "online", // "online", "offline", "error"  
        "battery_percent": 85.0, // If battery-powered  
        "uptime_seconds": 3600,  
        "error_code": 0 // 0 = OK, optional error codes  
    }  
}
```

• **Đề xuất hệ thống quản lý:** Cách máy chủ biết `env_data v1.0` và `v1.1` khác gì nhau:

– **Kho lưu trữ Schema (Schema Registry):**

- * Là một cơ sở dữ liệu (hoặc một Git repository) chứa các tệp tin định nghĩa schema.
- * Sẽ có các tệp như là "`env_data_v1.0.0.json`", "`camera_event_v1.0.0.json`",....
- * **Ưu điểm:** Cả team phát triển thiết bị (firmware) và team phát triển phần mềm (software) đều nhìn vào đây để làm việc.



- **Quản lý Phiên bản (Versioning):** Sử dụng Semantic Versioning (vMAJOR.MINOR.PATCH).
 - * PATCH (v1.0.1): Sửa lỗi nhỏ, không ảnh hưởng cấu trúc.
 - * MINOR (v1.1.0): Thêm trường dữ liệu mới, vẫn tương thích ngược.
 - * MAJOR (v2.0.0): Thay đổi lớn, không tương thích ngược.
- **Xác thực Schema (Schema Validation)**
 - * Dữ liệu từ cảm biến gửi đến (ví dụ: qua MQTT Broker).
 - * Một dịch vụ "Ingestor" (bộ tiếp nhận) sẽ đọc gói tin.
 - * Nó thấy **schema_id**: "env_data" và **schema_version**: "v1.1".
 - * Nó lập tức tra cứu trong Schema Registry để lấy tệp định nghĩa **env_data_v1.1.json**.
 - * Nó dùng tệp định nghĩa này để xác thực (validate) gói tin nhận được.
 - Nếu hợp lệ: Đẩy dữ liệu vào database (ví dụ: TimeScaleDB, InfluxDB).
 - Nếu không hợp lệ: Gói tin bị loại bỏ và gửi cảnh báo (ví dụ: "Thiết bị 'env-sensor-zone-a-01' đang gửi dữ liệu rác!").

- **Ưu điểm của đề xuất này:**

- **Ngăn chặn dữ liệu rác:** Hệ thống tự động loại bỏ dữ liệu sai định dạng ngay từ đầu vào.
- **Gỡ lỗi dễ dàng:** Biết chính xác thiết bị nào đang gửi sai phiên bản schema.
- **Dễ dàng mở rộng:** Khi muốn thêm cảm biến độ ẩm đất (v1.1), các thiết bị v1.0 cũ vẫn hoạt động bình thường song song với các thiết bị v1.1 mới.
- **Tính độc lập:** Xử lý camera (dữ liệu nặng) riêng biệt với telemetry (dữ liệu nhẹ) giúp hệ thống nhanh và ổn định.

5.8.2 Đề xuất cải tiến cho thuật toán RFE

5.8.2.a Vấn đề và hạn chế của RFE nguyên bản

Thuật toán Robust Feature Extractor (RFE) nguyên bản hoạt động dựa trên cơ chế phân tích chuỗi thời gian đơn biến (Univariate Time-series Analysis). Tức là, việc đánh giá trạng thái của một cảm biến chỉ dựa vào dữ liệu quá khứ của chính nó. Mặc dù hiệu quả trong việc phát hiện các lỗi cục bộ (như gai nhiễu, kẹt giá trị), phương pháp này tồn tại một hạn chế lớn khi gặp các biến động môi trường đồng loạt.

- Ví dụ: Khi hệ thống nhà màng mở lưới che nắng vào buổi trưa, nhiệt độ của toàn bộ khu vườn sẽ tăng vọt trong thời gian ngắn.
- Vấn đề: RFE cơ bản sẽ ghi nhận "Tốc độ thay đổi" (v_t) tăng cao đột ngột và có thể nhầm lẫn đây là lỗi trôi (Drift) hoặc gai (Spike), dẫn đến cảnh báo sai (False Positive).

5.8.2.b Giải pháp đề xuất:

Để khắc phục hạn chế trên, nghiên cứu đề xuất mở rộng vector đặc trưng bằng cách bổ sung thông tin tương quan không gian. Ý tưởng cốt lõi là so sánh hành vi của cảm biến mục tiêu với các cảm biến lân cận trong cùng một khu vực canh tác. Vector đặc trưng đầu ra F_t của thuật toán cải tiến sẽ có dạng:

$$F_t = \underbrace{[x_t, v_t, \sigma_t, EWMA_t, Lags]}_{\text{Temporal}}, \underbrace{S_t}_{\text{Spatial}}$$

Trong đó, S_t là chỉ số Độ lệch Không gian (Spatial Deviation), được tính toán nhằm lượng hóa mức độ khác biệt của cảm biến hiện tại so với tập thể.



5.8.2.c Cơ chế hoạt động

Giả sử tại thời điểm t , ta có giá trị của cảm biến mục tiêu là x_t và tập hợp giá trị của k cảm biến lân cận là $N_t = \{y_t^{(1)}, y_t^{(2)}, \dots, y_t^{(k)}\}$. Chỉ số S_t được tính toán đơn giản theo công thức:

$$S_t = |x_t - \text{median}(N_t)|$$

Trong đó:

$$\text{median}(N) = \begin{cases} y_{(\frac{k+1}{2})} & \text{nếu } k \text{ lẻ} \\ \frac{1}{2}(y_{(\frac{k}{2})} + y_{(\frac{k}{2}+1)}) & \text{nếu } k \text{ chẵn} \end{cases}$$

(Sử dụng trung vị (median) thay vì trung bình (mean) để tăng cường khả năng chịu đựng giá trị ngoại lai (Outliers).).

Về lí do sử dụng median thay vì mean:

- Mean tính toán dựa trên độ lớn của tất cả các phần tử, nên nó rất nhạy cảm với nhiễu. Một giá trị nhiễu lớn có thể kéo lệch toàn bộ giá trị tham chiếu.
- Median là một đại lượng thống kê thứ tự, có tính kháng nhiễu cao hơn hẳn. Nó coi các giá trị lỗi đột biến là các phần tử nằm ở rìa của phân phối và loại bỏ chúng, giúp thuật toán không bị báo động giả.

5.8.2.d Kết luận

Việc chuyển đổi từ phân tích đơn biến sang đa biến thông qua chỉ số S_t mang lại hai lợi ích thiết thực cho hệ thống Smart Farm:

- **Giảm tỷ lệ báo động giả:** Phân biệt rõ ràng giữa sự cố thiết bị và các hiện tượng thời tiết cực đoan.
- **Tăng độ tin cậy:** Hệ thống tận dụng được lợi thế của mạng lưới thiết bị IoT dày đặc để tự kiểm chứng chéo dữ liệu lẫn nhau ngay tại biên trước khi gửi về máy chủ.

Hạn chế của phương pháp cải tiến đề xuất:

- Mặc dù sử dụng bộ lọc trung vị (Median Filter) giúp tăng cường khả năng kháng nhiễu so với trung bình cộng, phương pháp này vẫn tồn tại giới hạn về điểm gãy. Theo lý thuyết thống kê, điểm gãy của trung vị là 50%. Điều này có nghĩa là nếu hơn 50% số lượng cảm biến tham chiếu trong mạng lưới gặp sự cố đồng thời và sai lệch theo cùng một hướng, giá trị trung vị sẽ bị sai lệch theo, dẫn đến khả năng tính toán chỉ số tương quan không gian S_t không còn chính xác.
- Tuy nhiên, trong thực tế vận hành hệ thống IoT nông nghiệp:
 - Các lỗi cảm biến phần cứng thường mang tính ngẫu nhiên và độc lập, xác suất xảy ra đồng thời trên diện rộng là thấp.
 - Trường hợp các giá trị thay đổi đồng loạt thường là dấu hiệu của sự thay đổi môi trường thực tế hoặc sự cố cấp hệ thống (mất nguồn), lúc này hệ thống giám sát cần chuyển sang cơ chế cảnh báo mức độ cao hơn thay vì phát hiện lỗi mức node.

Hướng khắc phục: Để giải quyết triệt để vấn đề này, các nghiên cứu tiếp theo có thể tích hợp cơ chế trọng số tin cậy. Hệ thống sẽ đánh giá độ tin cậy của từng node lân cận dựa trên lịch sử hoạt động của chúng. Các node có tiền sử lỗi thường xuyên sẽ bị giảm trọng số hoặc loại bỏ khỏi tập tham chiếu khi tính toán Median.

5.9 Thiết kế các thành phần chính

5.9.1 Thiết kế thành phần phần mềm: Quản lý cảm biến hướng dữ liệu

Phạm vi và Kế thừa: Đề tài được thực hiện trong bối cảnh kế thừa kết quả triển khai hạ tầng IoT thực tế tại **Tomochan Farm** (thuộc dự án nghiên cứu của phòng thí nghiệm TIST Lab - HCMUT). Trên cơ sở hệ thống thu thập vật lý đã có (PoC), nhóm tập trung nghiên cứu xây dựng **Kiến trúc quản**



lý dữ liệu cảm biến (Data-centric Sensor Management) và phát triển các thuật toán nhằm giám sát chất lượng, trạng thái và độ tin cậy của thiết bị dựa trên dòng dữ liệu thực tế.

Phần này trình bày các đóng góp chính của khối Khoa học Máy tính (CS) trong việc thiết kế và xây dựng hệ thống quản lý, tập trung vào bốn khía cạnh: quản lý cảm biến hướng dữ liệu, pipeline phát hiện lỗi, quản lý vòng đời thiết bị, và kiến trúc hệ thống tổng thể.

5.9.1.a Quản lý cảm biến hướng dữ liệu (Data-centric Sensor Management)

Mục tiêu của thành phần này là chứng minh rằng hệ thống coi cảm biến như một **thực thể dữ liệu (Data Entity)** chứ không chỉ đơn giản là một thiết bị phần cứng. Cách tiếp cận này cho phép quản lý và giám sát cảm biến dựa trên hành vi dữ liệu thay vì chỉ dựa trên trạng thái kết nối.

Mô hình hóa thực thể cảm biến: Hệ thống không chỉ quản lý định danh thiết bị (MAC address, Device ID), mà quản lý **hồ sơ hành vi (Behavior Profile)** của từng cảm biến. Mỗi cảm biến được định nghĩa bởi các đặc trưng dữ liệu quan trọng:

- **Tần suất gửi tin (Transmission Frequency):** Mô hình học và theo dõi chu kỳ gửi dữ liệu thực tế của cảm biến. Nếu cảm biến đột ngột thay đổi tần suất (ví dụ: từ 5 giây/lần thành 30 giây/lần), hệ thống nhận diện đây có thể là dấu hiệu lỗi hoặc suy giảm hiệu năng.
- **Dải giá trị cho phép (Value Range):** Mỗi loại cảm biến có một dải giá trị hợp lý dựa trên đặc tính vật lý và điều kiện môi trường. Ví dụ, cảm biến nhiệt độ trong nhà kính thường dao động trong khoảng 15-40°C. Hệ thống lưu trữ và cập nhật các ngưỡng này dựa trên dữ liệu lịch sử.
- **Độ ổn định tín hiệu (Signal Variance):** Hệ thống tính toán và theo dõi độ lệch chuẩn (standard deviation) của dữ liệu cảm biến trong cửa sổ thời gian nhất định. Một cảm biến khỏe mạnh sẽ có độ lệch chuẩn ổn định, trong khi cảm biến bị suy giảm sẽ có độ lệch chuẩn tăng dần theo thời gian.
- **Pattern theo thời gian (Temporal Pattern):** Hệ thống học và lưu trữ các pattern bình thường của cảm biến theo chu kỳ (ví dụ: nhiệt độ tăng vào ban ngày, giảm vào ban đêm). Các giá trị vi phạm pattern này (ví dụ: nhiệt độ tăng vào ban đêm) có thể là dấu hiệu lỗi.

Giám sát chất lượng dữ liệu (Data Quality Monitoring - DQM): Module CS tiếp nhận dòng dữ liệu từ CE và thực hiện phân tích để phát hiện các bất thường sơ cấp ngay trong quá trình ingestion:

- **Phát hiện dữ liệu bị thiếu (Missing Data Detection):** Hệ thống theo dõi timestamp của các gói tin nhận được. Nếu khoảng cách giữa hai gói tin liên tiếp vượt quá ngưỡng cho phép (ví dụ: 3 lần chu kỳ gửi bình thường), hệ thống đánh dấu cảm biến có vấn đề về kết nối.
- **Phát hiện trôi dạt (Drift Detection):** Thuật toán theo dõi xu hướng dài hạn của dữ liệu. Nếu giá trị cảm biến tăng/giảm một cách từ từ và liên tục (ví dụ: nhiệt độ tăng 0.1°C mỗi ngày trong 10 ngày), đây có thể là dấu hiệu cảm biến bị trôi (sensor drift) do lão hóa hoặc ảnh hưởng môi trường.
- **Phát hiện giá trị bị kẹt (Stuck-at Detection):** Hệ thống phát hiện khi cảm biến báo cùng một giá trị trong thời gian dài (ví dụ: độ ẩm luôn là 75.2% trong 1 giờ). Đây là dấu hiệu cảm biến bị "đóng băng"(frozen) hoặc mất kết nối với cảm biến vật lý.
- **Phân biệt nhiễu và biến động thực tế:** Một thách thức quan trọng là phân biệt giữa biến động môi trường thật và lỗi thiết bị. Ví dụ:

- *Biến động thật:* Khi trời mưa, độ ẩm không khí tăng vọt từ 60% lên 95%. Đây là biến động hợp lý và hệ thống không nên cảnh báo.
- *Lỗi thiết bị:* Trời nắng, nhiệt độ cao (35°C) nhưng độ ẩm đột ngột tăng vọt lên 95%. Điều này không hợp lý và hệ thống nên cảnh báo có thể cảm biến độ ẩm bị lỗi.

Để giải quyết, hệ thống sử dụng cơ chế **Cross-correlation** giữa các cảm biến: nếu chỉ một cảm biến báo giá trị bất thường trong khi các cảm biến khác cùng khu vực báo giá trị bình thường, khả năng cao đây là lỗi thiết bị.



5.9.1.b Pipeline phát hiện và phân loại lỗi cảm biến (Sensor Fault Detection Pipeline)

Dây là thành phần cốt lõi của hệ thống CS, tập trung vào việc tự động hóa quá trình phát hiện và phân loại lỗi cảm biến dựa trên phân tích dữ liệu.

Luồng xử lý (Processing Pipeline): Pipeline được thiết kế theo mô hình ba giai đoạn: **Ingest → Clean → Analyze → Label**

1. **Ingest (Tiếp nhận dữ liệu):** Module tiếp nhận nhận dòng dữ liệu thô (Raw Telemetry Stream) từ CE thông qua Message Queue (RabbitMQ). Việc sử dụng hàng đợi bản tin giúp đảm bảo không mất mát dữ liệu telemetry khi lưu lượng tăng đột biến, và cho phép hệ thống xử lý bất đồng bộ với khả năng mở rộng độc lập.
2. **Clean (Làm sạch dữ liệu):** Dữ liệu được kiểm tra tính hợp lệ cơ bản:
 - Xác thực schema JSON (kiểm tra các trường bắt buộc: device_id, timestamp, data)
 - Kiểm tra timestamp hợp lệ (không quá khứ xa hoặc tương lai)
 - Loại bỏ các giá trị ngoại lai rõ ràng (outliers) dựa trên range checking
3. **Analyze (Phân tích):** Áp dụng thuật toán trích xuất đặc trưng (RFE - Robust Feature Extractor) kết hợp với mô hình phân lớp (Random Forest/XGBoost) để phân tích pattern và phát hiện bất thường.
4. **Label (Gán nhãn trạng thái):** Dựa trên kết quả phân tích, hệ thống gán nhãn trạng thái sức khỏe cảm biến:
 - **Normal:** Cảm biến hoạt động bình thường, dữ liệu trong phạm vi hợp lệ
 - **Suspect:** Có dấu hiệu bất thường nhưng chưa rõ ràng, cần theo dõi thêm
 - **Faulty:** Cảm biến có lỗi rõ ràng (drift, stuck-at, hardware failure)

Lưu ý quan trọng: Output của pipeline là **trạng thái sức khỏe cảm biến**, không phải dự báo thời tiết hay phân tích môi trường. Mục tiêu là trả lời câu hỏi: "Cảm biến này có đang hoạt động đúng không?" thay vì "Nhiệt độ sẽ là bao nhiêu ngày mai?"

Cơ chế xác thực đa nguồn (Multi-modal Cross-validation): Để tăng độ tin cậy của phát hiện lỗi, hệ thống sử dụng nhiều nguồn dữ liệu để xác thực chéo:

- **Tương quan chéo (Cross-correlation):** So sánh dữ liệu giữa các cảm biến lân cận trong cùng một khu vực canh tác. Nếu một cảm biến báo nhiệt độ 40°C, nhưng ba cảm biến xung quanh đều báo 30°C, hệ thống nghi ngờ cảm biến đó có vấn đề và đánh dấu trạng thái "Suspect" hoặc "Faulty".
- **Tham chiếu hình ảnh (Visual Verification):** Sử dụng dữ liệu từ Camera Stream như một kênh tham chiếu phụ (Ground Truth) để giảm tỷ lệ báo động giả (False Positive).
 - Ví dụ: Cảm biến độ ẩm đất báo giá trị "0" (khô hạn - lỗi Stuck-at-0), nhưng camera phân tích hình ảnh thấy đất sẫm màu (dấu hiệu ướt). Hệ thống kết luận: Cảm biến độ ẩm đất có thể bị hỏng hoặc mất tiếp xúc với đất.
- **Kiểm tra tính nhất quán theo thời gian (Temporal Consistency):** Hệ thống so sánh giá trị hiện tại với giá trị trung bình trong cửa sổ thời gian gần đây (ví dụ: 1 giờ). Nếu giá trị hiện tại khác biệt quá lớn so với xu hướng (ví dụ: $\pm 3\sigma$), hệ thống đánh dấu "Suspect".

Thuật toán RFE và cải tiến: Nhóm nghiên cứu đề xuất sử dụng thuật toán **Robust Feature Extractor (RFE)** kết hợp với mô hình phân lớp để phát hiện lỗi cảm biến. Điểm đổi mới là mở rộng vector đặc trưng bằng cách bổ sung thông tin tương quan không gian (Spatial Correlation) ngoài phân tích theo thời gian (Temporal Analysis).

Vector đặc trưng đầu ra F_t có dạng:

$$F_t = \underbrace{[x_t, v_t, \sigma_t, EWMA_t, Lags]}_{\text{Temporal Features}}, \underbrace{S_t}_{\text{Spatial Deviation}}]$$



Trong đó S_t là chỉ số độ lệch không gian, được tính toán dựa trên so sánh với các cảm biến lân cận:

$$S_t = |x_t - \text{median}(N_t)|$$

Việc sử dụng median thay vì mean giúp tăng khả năng kháng nhiễu, đảm bảo thuật toán không bị ảnh hưởng bởi các giá trị ngoại lai từ cảm biến khác.

5.9.1.c Quản lý vòng đời thiết bị dựa trên dữ liệu (Data-driven Lifecycle Management)

Thành phần này thể hiện tính ứng dụng lâu dài của hệ thống, không chỉ phát hiện lỗi mà còn hỗ trợ quản lý và bảo trì thiết bị một cách chủ động.

Theo dõi xu hướng suy giảm (Degradation Tracking): Hệ thống lưu trữ và phân tích lịch sử hoạt động của cảm biến để vẽ biểu đồ "sức khỏe" theo thời gian:

- **Chỉ số sức khỏe cảm biến (Sensor Health Index):** Được tính toán dựa trên nhiều yếu tố:
 - Tỷ lệ dữ liệu hợp lệ so với tổng số gói tin nhận được
 - Độ lệch chuẩn của dữ liệu (tăng dần cho thấy cảm biến đang suy giảm)
 - Tần suất cảnh báo lỗi (số lần cảm biến được đánh dấu "Suspect" hoặc "Faulty")
 - Độ chính xác so với cảm biến tham chiếu (cross-correlation)
- **Dự đoán suy giảm (Degradation Prediction):** Dựa trên xu hướng lịch sử, hệ thống có thể dự đoán khi nào cảm biến sẽ cần hiệu chuẩn hoặc thay thế. Ví dụ: Nếu độ lệch chuẩn tăng dần từ 0.5°C lên 2.0°C trong 6 tháng, hệ thống cảnh báo cảm biến có thể cần hiệu chuẩn sớm.

Hỗ trợ ra quyết định bảo trì (Maintenance Decision Support): Thay vì đợi cảm biến hỏng hẳn mới thay thế (reactive maintenance), hệ thống hỗ trợ bảo trì chủ động (predictive maintenance):

- **Cảnh báo sớm (Early Warning):** Khi độ lệch chuẩn tăng dần theo thời gian, hệ thống gửi cảnh báo "Cảm biến đang có dấu hiệu suy giảm" thay vì chờ đến khi cảm biến hoàn toàn hỏng.
- **Đề xuất hành động (Action Recommendation):** Dựa trên mức độ suy giảm, hệ thống đề xuất:
 - *Hiệu chuẩn (Calibrate)*: Nếu cảm biến chỉ bị lệch nhẹ, có thể hiệu chuẩn lại
 - *Thay thế (Replace)*: Nếu cảm biến đã suy giảm nghiêm trọng hoặc có nhiều lỗi, nên thay thế
 - *Kiểm tra vật lý (Physical Inspection)*: Nếu có dấu hiệu lỗi hardware, cần kiểm tra kết nối vật lý
- **Lịch sử bảo trì (Maintenance History):** Hệ thống lưu trữ lịch sử các lần hiệu chuẩn và thay thế, giúp phân tích hiệu quả bảo trì và dự đoán chi phí bảo trì trong tương lai.

5.9.1.d Kiến trúc hệ thống và Giao diện quản trị (System Architecture & Administration Interface)

Pipeline dữ liệu (Logical Data Pipeline): Kiến trúc hệ thống được thiết kế theo mô hình pipeline xử lý dữ liệu logic, mô tả luồng đi của dữ liệu từ thiết bị đến người dùng:

- **Ingest (Nhận dữ liệu MQTT):** Hệ thống tiếp nhận dữ liệu từ thiết bị IoT thông qua Message Broker (RabbitMQ). Việc sử dụng hàng đợi bản tin đảm bảo không mất mát dữ liệu telemetry khi lưu lượng tăng đột biến và cho phép xử lý bất đồng bộ.
- **Clean (Lọc nhiễu thô):** Dữ liệu được làm sạch và xác thực schema ngay khi tiếp nhận, loại bỏ các gói tin không hợp lệ trước khi đưa vào pipeline xử lý.
- **Analyze (Chạy RFE Model):** Worker service chạy thuật toán RFE và mô hình ML để phân tích và phát hiện lỗi. Service này có thể scale độc lập dựa trên số lượng thiết bị.



- **Label (Gán nhãn trạng thái):** Kết quả phân tích được gán nhãn (Normal/Suspect/Faulty) và lưu trữ vào database cùng với dữ liệu gốc.
- **Store (Lưu trữ):** Dữ liệu được lưu trữ vào Time-series Database (TimescaleDB) để hỗ trợ truy vấn hiệu quả dữ liệu lịch sử.
- **Present (Trình bày):** Dữ liệu được trình bày lên Dashboard thông qua WebSocket để cập nhật real-time, hoặc qua REST API để truy vấn lịch sử.

Tính tổng quát hóa (Generalization & Hardware Agnostic): Hệ thống CS được thiết kế với nguyên tắc **hardware agnostic** (không phụ thuộc phần cứng cụ thể):

- **API theo chuẩn mở:** Hệ thống định nghĩa cấu trúc gói tin JSON chuẩn cho tất cả loại cảm biến. bất kỳ thiết bị nào (hãng A, hãng B, hoặc tự chế) chỉ cần tuân thủ cấu trúc này đều có thể tích hợp vào hệ thống.
- **Schema Registry:** Hệ thống sử dụng Schema Registry để quản lý các phiên bản schema khác nhau. Khi firmware được cập nhật và thêm cảm biến mới (ví dụ: thêm cảm biến độ pH), hệ thống chỉ cần đăng ký schema mới mà không cần thay đổi code xử lý.
- **Versioning và Backward Compatibility:** Hệ thống hỗ trợ nhiều phiên bản schema đồng thời. Thiết bị cũ (v1.0) và thiết bị mới (v1.1) có thể hoạt động song song mà không ảnh hưởng lẫn nhau.

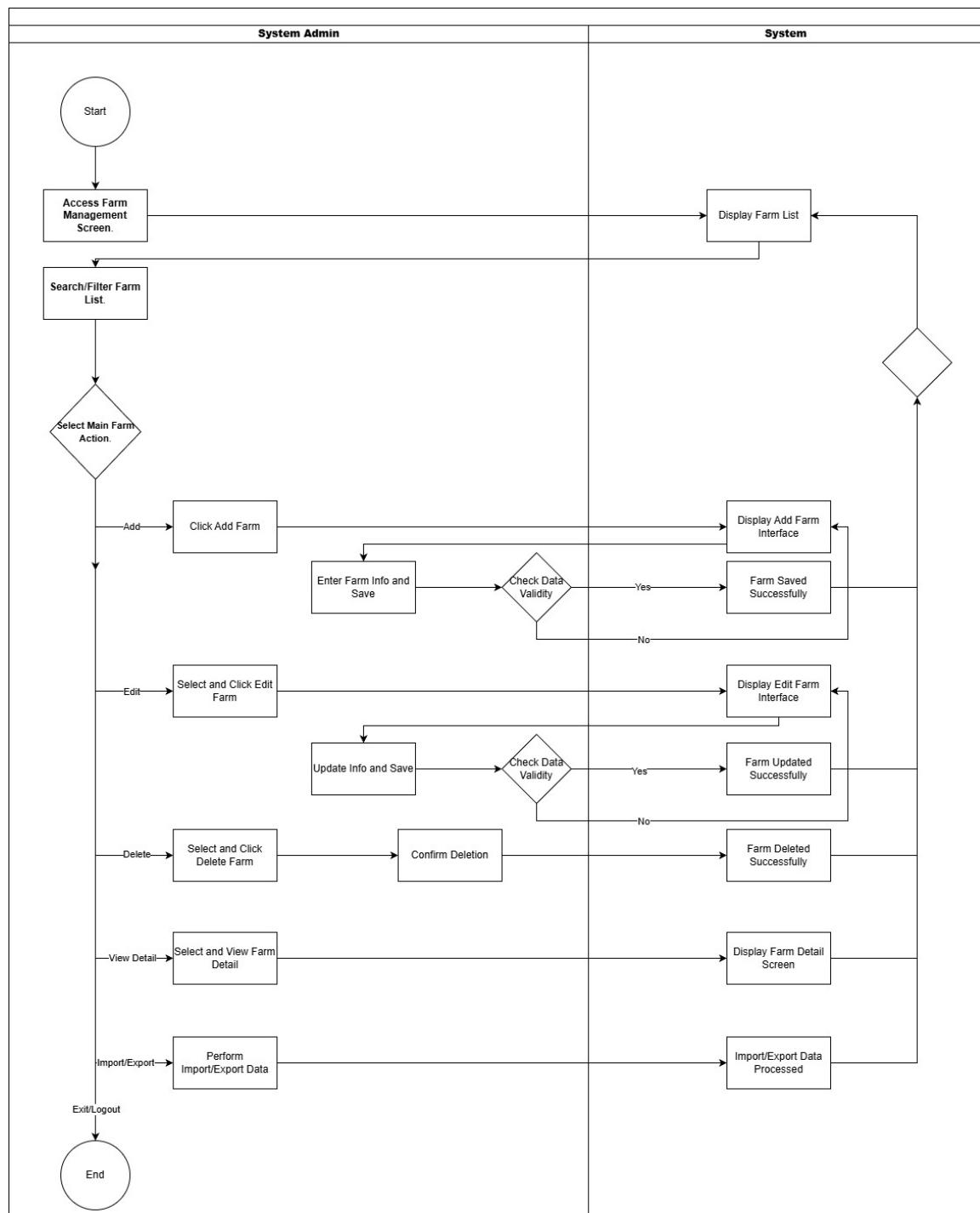
Dashboard quản trị với chỉ số tin cậy (Administration Dashboard with Confidence Scores): Giao diện quản trị không chỉ hiển thị số đo (25°C, 80%), mà còn hiển thị **Chỉ số tin cậy (Confidence Score)** của từng giá trị:

- **Ví dụ hiển thị Normal:**
 - Nhiệt độ: 30°C (*Dộ tin cậy: 98% - Normal*)
 - Độ ẩm: 75% (*Dộ tin cậy: 95% - Normal*)
- **Ví dụ hiển thị Suspect:**
 - Độ ẩm đất: 10% (*Dộ tin cậy: 20% - Suspect/Drift Error*)
 - Nhiệt độ: 45°C (*Dộ tin cậy: 15% - Suspect/Cross-correlation failed*)
- **Các thông tin bổ sung:**
 - **Health Trend:** Biểu đồ xu hướng sức khỏe cảm biến theo thời gian (tăng/giảm/ ổn định)
 - **Last Calibration:** Ngày hiệu chuẩn gần nhất
 - **Maintenance Recommendation:** Đề xuất hành động (nếu có)
 - **Alert History:** Lịch sử cảnh báo và cách xử lý

Các thành phần kỹ thuật phục vụ quản lý cảm biến: Mỗi thành phần công nghệ được lựa chọn và thiết kế với mục đích cụ thể phục vụ quản lý cảm biến:

- **Message Queue (RabbitMQ):** Sử dụng hàng đợi bản tin để đảm bảo không mất mát dữ liệu telemetry khi lưu lượng tăng đột biến, và cho phép xử lý bất đồng bộ với khả năng mở rộng độc lập module phân tích.
- **Time-series Database (TimescaleDB):** Lưu trữ dữ liệu cảm biến với tối ưu hóa cho truy vấn theo thời gian, hỗ trợ hiệu quả việc phân tích xu hướng và phát hiện drift.
- **In-memory Cache:** Sử dụng bộ nhớ đệm để đảm bảo truy xuất trạng thái cảm biến thời gian thực với độ trễ thấp, phục vụ hiển thị Dashboard real-time.
- **WebSocket Server:** Cung cấp kênh truyền tải hai chiều để push cập nhật trạng thái cảm biến và cảnh báo lỗi lên Dashboard ngay lập tức, không cần client phải polling liên tục.

5.9.1.e Quy trình quản lý Farm:



Hình 13: Activity Diagram mô tả luồng quản lý farm

Quy trình quản lý trang trại

Tổng quan sơ đồ

Sơ đồ mô tả quy trình quản lý trang trại (Farm Management), thể hiện sự tương tác giữa **System Admin** (Quản trị viên hệ thống) và **System** (Hệ thống). Quy trình bao gồm các thao tác cơ bản như xem danh sách, tìm kiếm, thêm, chỉnh sửa, xóa, xem chi tiết và nhập/xuất dữ liệu trang trại.



Mô tả chi tiết luồng hoạt động

• Giai đoạn khởi đầu và hiển thị:

- **Truy cập:** Quản trị viên truy cập vào màn hình quản lý trang trại (*Access Farm Management Screen*).
- **Hiển thị:** Hệ thống hiển thị danh sách các trang trại hiện có (*Display Farm List*).
- **Tìm kiếm/Lọc:** Quản trị viên có thể thực hiện tìm kiếm hoặc lọc danh sách trang trại (*Search/Filter Farm List*).

• Giai đoạn lựa chọn tác vụ (*Select Main Farm Action*):

Tại bước này, Quản trị viên có thể lựa chọn một trong các nhánh xử lý sau:

– Nhánh thêm trang trại (Add):

- * Quản trị viên chọn thêm trang trại (*Click Add Farm*).
- * Hệ thống hiển thị giao diện thêm mới (*Display Add Farm Interface*).
- * Quản trị viên nhập thông tin trang trại và lưu (*Enter Farm Info and Save*).
- * Hệ thống kiểm tra tính hợp lệ của dữ liệu (*Check Data Validity*).
 - Nếu dữ liệu hợp lệ, hệ thống thông báo *Farm Saved Successfully* và quay lại màn hình danh sách.
 - Nếu dữ liệu không hợp lệ, hệ thống yêu cầu nhập lại tại giao diện thêm mới.

– Nhánh chỉnh sửa trang trại (Edit):

- * Quản trị viên chọn và chỉnh sửa trang trại (*Select and Click Edit Farm*).
- * Hệ thống hiển thị giao diện chỉnh sửa.
- * Quản trị viên cập nhật thông tin và lưu (*Update Info and Save*).
- * Hệ thống kiểm tra dữ liệu và thông báo *Farm Updated Successfully* nếu hợp lệ.

– Nhánh xóa trang trại (Delete):

- * Quản trị viên chọn và xóa trang trại (*Select and Click Delete Farm*).
- * Quản trị viên xác nhận thao tác xóa (*Confirm Deletion*).
- * Hệ thống thực hiện xóa và thông báo *Farm Deleted Successfully*.

– Nhánh xem chi tiết (View Detail):

- * Quản trị viên chọn xem chi tiết trang trại (*Select and View Farm Detail*).
- * Hệ thống hiển thị màn hình chi tiết trang trại (*Display Farm Detail Screen*).

– Nhánh nhập/xuất dữ liệu (Import/Export):

- * Quản trị viên thực hiện thao tác nhập hoặc xuất dữ liệu (*Perform Import/Export Data*).
- * Hệ thống xử lý và phản hồi kết quả (*Import/Export Data Processed*).

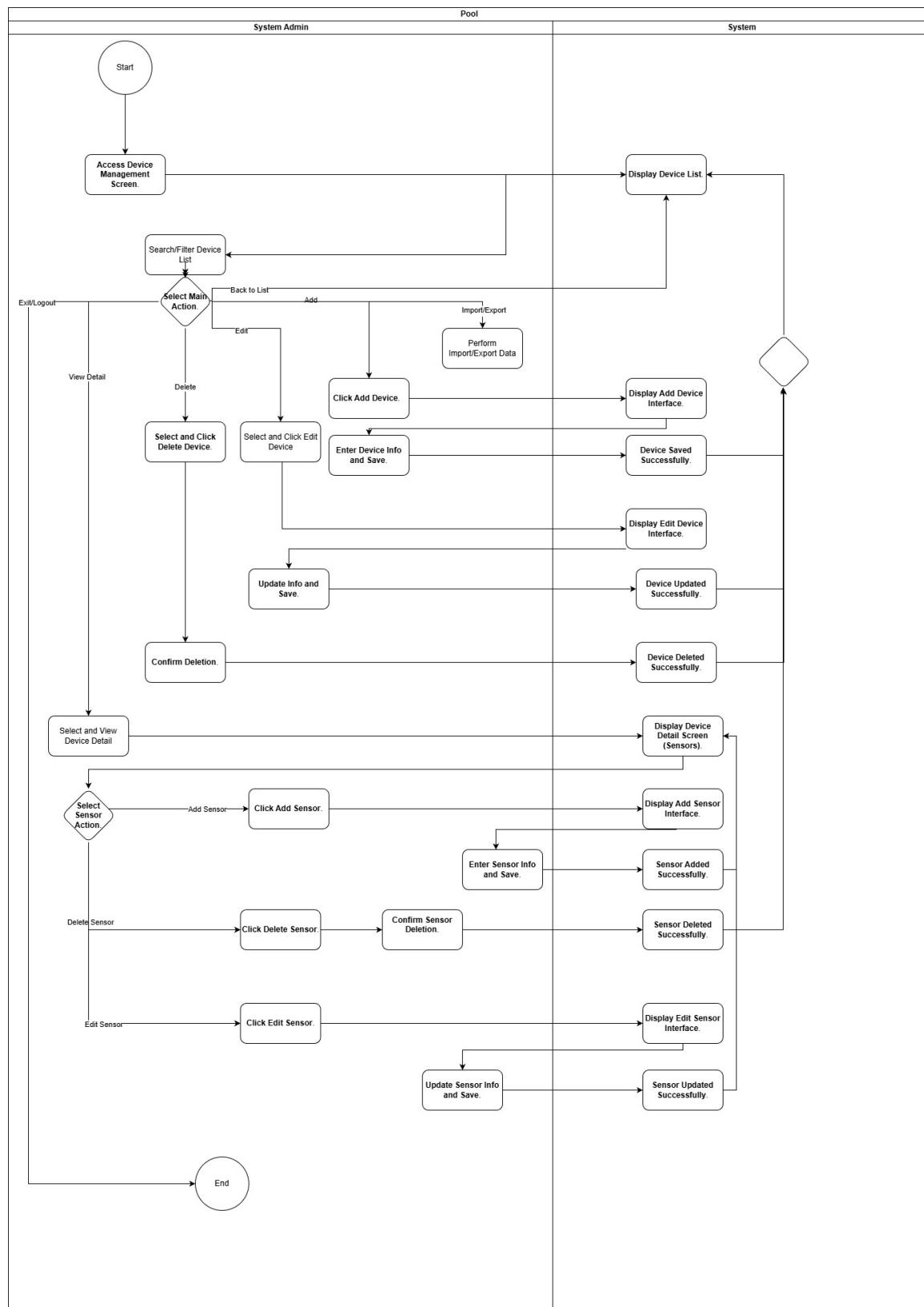
• Giai đoạn kết thúc:

Sau khi hoàn tất các tác vụ quản lý, Quản trị viên có thể lựa chọn thoát hoặc đăng xuất (*Exit/Logout*) để kết thúc quy trình tại điểm *End*.

Quy tắc điều hướng và dữ liệu

- **Vòng lặp thao tác:** Sau mỗi tác vụ thành công như lưu, cập nhật, xóa hoặc xử lý dữ liệu, hệ thống đều điều hướng người dùng quay trở lại màn hình hiển thị danh sách trang trại nhằm đảm bảo luồng làm việc liên tục.
- **Kiểm tra điều kiện dữ liệu:** Các thao tác thêm và chỉnh sửa trang trại bắt buộc phải trải qua bước kiểm tra tính hợp lệ của dữ liệu trước khi hệ thống ghi nhận thay đổi.

5.9.1.f Quy trình quản lý thiết bị:



Hình 14: Activity Diagram mô tả luồng quản lý thiết bị

Mô tả quy trình quản lý thiết bị và cảm biến



Tổng quan sơ đồ

Sơ đồ mô tả quy trình quản lý các thiết bị trong hệ thống với hai cấp độ quản lý chính: cấp độ thiết bị (Device) và cấp độ chi tiết bên trong là cảm biến (Sensor). Quy trình thể hiện rõ sự phân chia trách nhiệm giữa Quản trị viên hệ thống (System Admin) và Hệ thống (System) trong suốt quá trình thao tác.

Mô tả chi tiết luồng hoạt động

- Giai đoạn khởi đầu và điều hướng:

- **Truy cập hệ thống:** Quản trị viên truy cập vào màn hình quản lý thiết bị (*Access Device Management Screen*).
- **Hiển thị danh sách:** Hệ thống phản hồi bằng cách hiển thị danh sách các thiết bị hiện có (*Display Device List*).
- **Tìm kiếm/Lọc:** Quản trị viên có thể thực hiện tìm kiếm hoặc lọc thiết bị nhằm thu hẹp phạm vi quản lý.

- Giai đoạn lựa chọn hành động chính (*Select Main Action*): Tại bước này, Quản trị viên có thể rẽ nhánh sang các tác vụ CRUD khác nhau:

- **Nhánh thêm thiết bị (Add Device):** Quản trị viên nhấn nút thêm thiết bị, hệ thống hiển thị giao diện nhập liệu. Sau khi nhập thông tin và lưu, hệ thống thông báo *Device Saved Successfully* và quay lại màn hình danh sách.
- **Nhánh sửa thiết bị (Edit Device):** Quản trị viên chọn thiết bị cần chỉnh sửa, hệ thống hiển thị giao diện chỉnh sửa. Sau khi cập nhật và lưu, hệ thống thông báo *Device Updated Successfully*.
- **Nhánh xóa thiết bị (Delete Device):** Quản trị viên chọn thiết bị và thực hiện bước xác nhận xóa (*Confirm Deletion*). Hệ thống tiến hành xóa và thông báo kết quả thành công.
- **Nhánh nhập/xuất dữ liệu (Import/Export):** Quản trị viên thực hiện thao tác nhập hoặc xuất dữ liệu thiết bị, hệ thống xử lý dữ liệu tương ứng.

- Giai đoạn quản lý chi tiết cảm biến (*Sensor Action*): Đây là điểm khác biệt quan trọng của sơ đồ khi Quản trị viên lựa chọn xem chi tiết thiết bị (*View Detail*).

- **Hiển thị chi tiết:** Hệ thống chuyển sang màn hình chi tiết thiết bị, hiển thị danh sách các cảm biến thuộc thiết bị đó (*Display Device Detail Screen (Sensors)*).
- **Thao tác với cảm biến:** Quản trị viên có thể thực hiện các thao tác quản lý cảm biến bao gồm:
 - * **Add Sensor:** Nhập thông tin và lưu cảm biến mới.
 - * **Edit Sensor:** Cập nhật thông tin cảm biến hiện tại.
 - * **Delete Sensor:** Xác nhận và xóa cảm biến khỏi thiết bị.
- **Kết quả:** Sau mỗi thao tác, hệ thống đều hiển thị thông báo xác nhận trạng thái thành công.

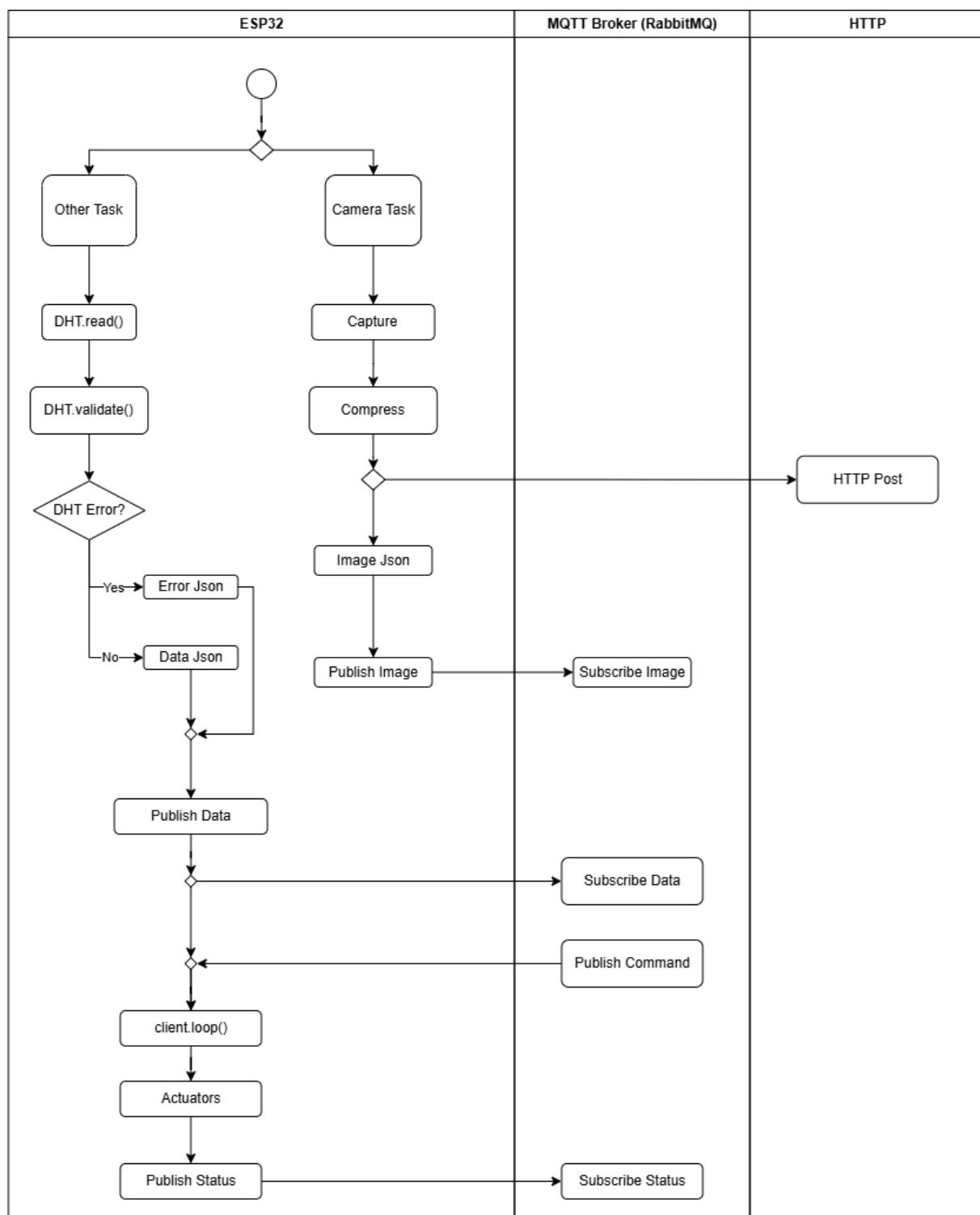
- **Giai đoạn kết thúc:** Quản trị viên có thể quay lại danh sách thiết bị chính (*Back to List*) để tiếp tục quản lý hoặc chọn thoát/đăng xuất (*Exit/Logout*) để kết thúc phiên làm việc tại điểm *End*.

Quy tắc nghiệp vụ đặc trưng

- **Quản lý phân cấp:** Thiết bị và cảm biến có mối quan hệ cha-con, do đó việc quản lý cảm biến chỉ được thực hiện sau khi người dùng truy cập vào màn hình chi tiết của thiết bị.
- **Vòng lặp tương tác:** Sau mỗi thao tác lưu hoặc cập nhật thành công, hệ thống luôn có luồng quay trở lại màn hình danh sách thông qua các nút điều hướng, cho phép Quản trị viên tiếp tục các thao tác quản lý.

5.9.2 Firmware

5.9.2.a Activity Diagram

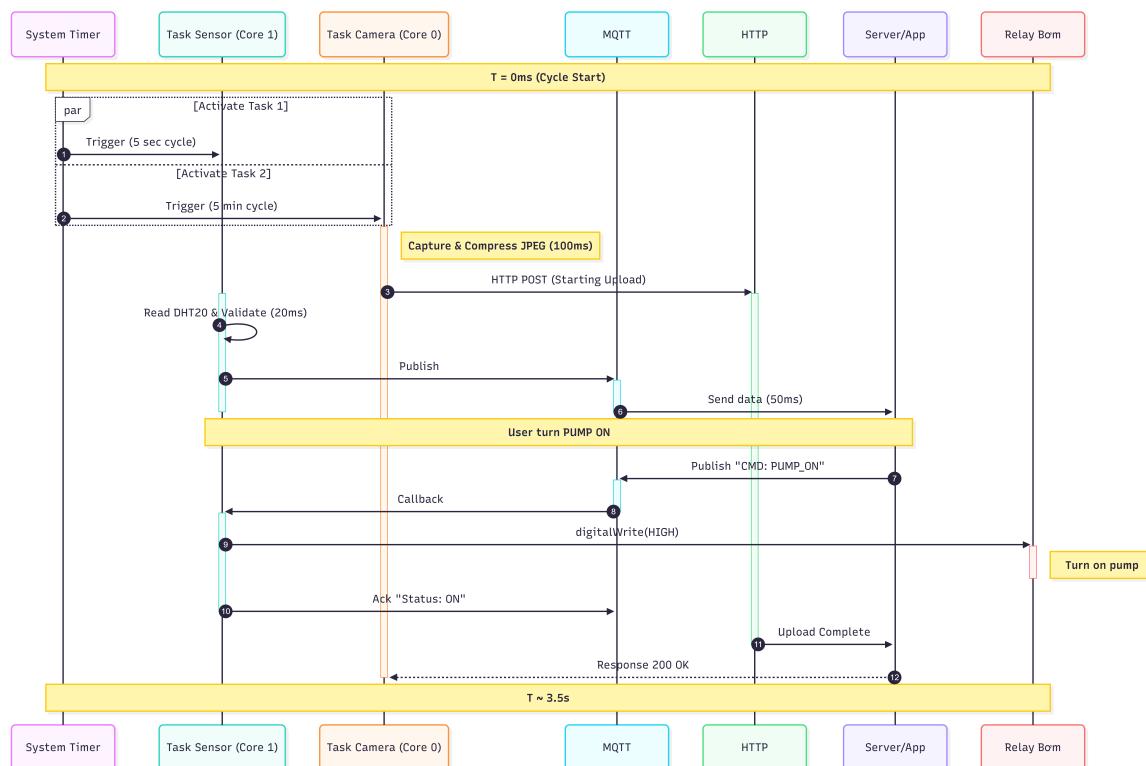


Hình 15: Activity Diagram

Hình trên minh họa luồng xử lý đa nhiệm của hệ thống, được xây dựng trên nền tảng hệ điều hành thời gian thực FreeRTOS. Để tận dụng sức mạnh phần cứng của ESP32, firmware được thiết kế phân chia thành hai luồng tác vụ (Tasks) chạy song song trên hai lõi CPU riêng biệt nhằm tránh hiện tượng nghẽn cổ chai (blocking):

- Luồng 1 (Core 1 - Sensor & Control Task):** Đây là luồng xử lý chính, chịu trách nhiệm khởi tạo kết nối WiFi/MQTT. Trong vòng lặp vô hạn (Client Loop), luồng này thực hiện đọc dữ liệu cảm biến (DHT20, độ ẩm đất), kiểm tra tính hợp lệ của dữ liệu (Data Valid), đóng gói JSON và gửi về Broker thông qua giao thức MQTT. Đồng thời, nó cũng lắng nghe các lệnh điều khiển (MQTT Cmd) để kích hoạt Relay/Còi báo động tức thì.
- Luồng 2 (Core 0 - Camera Task):** Được dành riêng cho tác vụ xử lý hình ảnh vốn tiêu tốn nhiều tài nguyên. Quy trình bao gồm khởi tạo Camera, chụp ảnh, nén định dạng JPEG và gửi lên Server thông qua giao thức HTTP POST. Việc tách rời này đảm bảo rằng quá trình gửi ảnh không làm gián đoạn việc gửi dữ liệu cảm biến thời gian thực ở Core 1.

5.9.2.b Sequence Diagram



Hình 16: Sequence Diagram

Hệ thống bao gồm 7 thực thể chính:

- System Timer:** Bộ định thời hệ thống, đóng vai trò kích hoạt các tác vụ theo chu kỳ.
- Task Sensor (Core 1):** Tác vụ xử lý cảm biến, chạy trên lõi 1 của vi điều khiển.
- Task Camera (Core 0):** Tác vụ xử lý camera (nâng về tính toán), chạy trên lõi 0.
- MQTT:** Giao thức truyền tin nhắn nhẹ, dùng để gửi dữ liệu cảm biến và nhận lệnh điều khiển.
- HTTP:** Giao thức dùng để tải dữ liệu lớn (hình ảnh JPEG) lên server.
- Server/App:** Nơi nhận dữ liệu và là giao diện để người dùng tương tác.
- Relay Bom:** Thiết bị ngoại vi thực hiện hành động vật lý (bật/tắt bom).

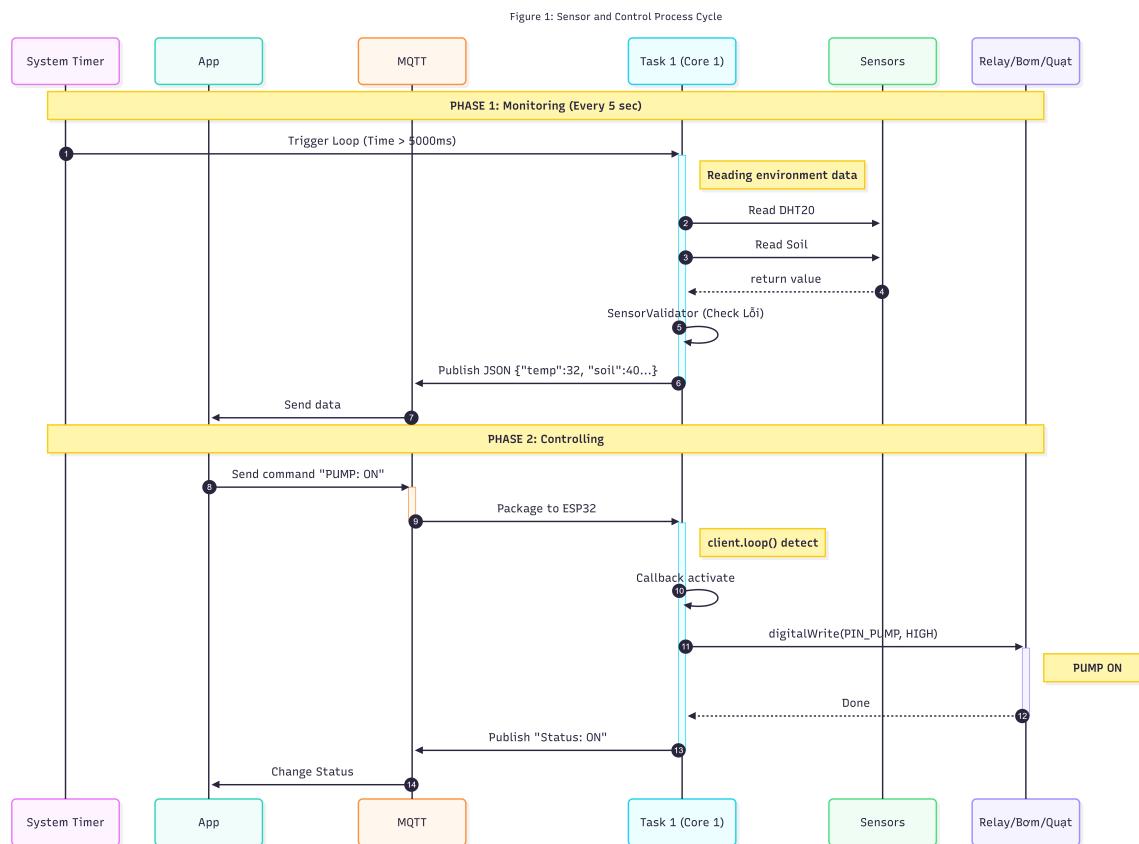
Quy trình được chia thành các giai đoạn chính trong một chu kỳ từ T = 0ms đến T = 3500ms như sau:

- Kích hoạt và Xử lý dữ liệu tại chỗ (T = 0ms)
 - System Timer gửi tín hiệu Trigger song song để kích hoạt hai tác vụ:



- * Task 1 (Sensor): Chu kỳ 5 giây/lần.
- * Task 2 (Camera): Chu kỳ 5 phút/lần.
- **Tại Task Camera:** Thực hiện chụp và nén ảnh JPEG (Capture & Compress). Quá trình này mất khoảng 100ms.
- **Tại Task Sensor:** Thực hiện đọc dữ liệu từ cảm biến DHT20 và xác thực dữ liệu. Quá trình này mất khoảng 20ms.
- Truyền tải dữ liệu lên Server (Upstream)
 - **Dữ liệu ảnh:** Task Camera sử dụng giao thức HTTP POST để bắt đầu tải ảnh lên Server.
 - **Dữ liệu cảm biến:** Task Sensor sử dụng giao thức MQTT để Publish (xuất bản) dữ liệu lên broker, sau đó dữ liệu được chuyển đến Server/App trong khoảng 50ms.
- Tương tác người dùng và Điều khiển (Downstream)
 - Người dùng thực hiện hành động "User turn PUMP ON" trên App.
 - Server gửi một tin nhắn MQTT với nội dung CMD: PUMP_ON.
 - MQTT Broker gửi một tín hiệu Callback về cho Task Sensor trên ESP32.
 - Task Sensor thực hiện lệnh digitalWrite(HIGH) gửi đến Relay Bơm.
 - Relay Bơm phản hồi trạng thái bằng cách thực hiện hành động "Turn on pump".
 - Sau khi bật thành công, Task Sensor gửi một gói tin xác nhận (Ack "Status: ON") ngược lại phía MQTT để cập nhật lên Server.
- Kết thúc chu kỳ ($T \approx 3.5s$)
 - Quá trình tải ảnh lên (HTTP) hoàn tất (Upload Complete).
 - Server phản hồi mã trạng thái 200 OK cho tác vụ Camera.
 - Toàn bộ chu kỳ kết thúc tại mốc xấp xỉ 3.5 giây.

5.9.2.c Timing Diagram



Hình 17: Timing Diagram for sensor processing and control tasks

Các thành phần tham gia:

- System Timer**: Bộ định thời kích hoạt vòng lặp Monitoring.
- App**: Giao diện người dùng trên điện thoại hoặc máy tính.
- MQTT**: Broker đóng vai trò trung gian truyền tin.
- Task 1 (Core 1)**: Tác vụ xử lý chính chạy trên lõi 1 của vi điều khiển.
- Sensors**: Các cảm biến vật lý (DHT20 cho nhiệt độ/độ ẩm và Soil cho độ ẩm đất).
- Relay/Bơm/Quạt**: Các thiết bị chấp hành.

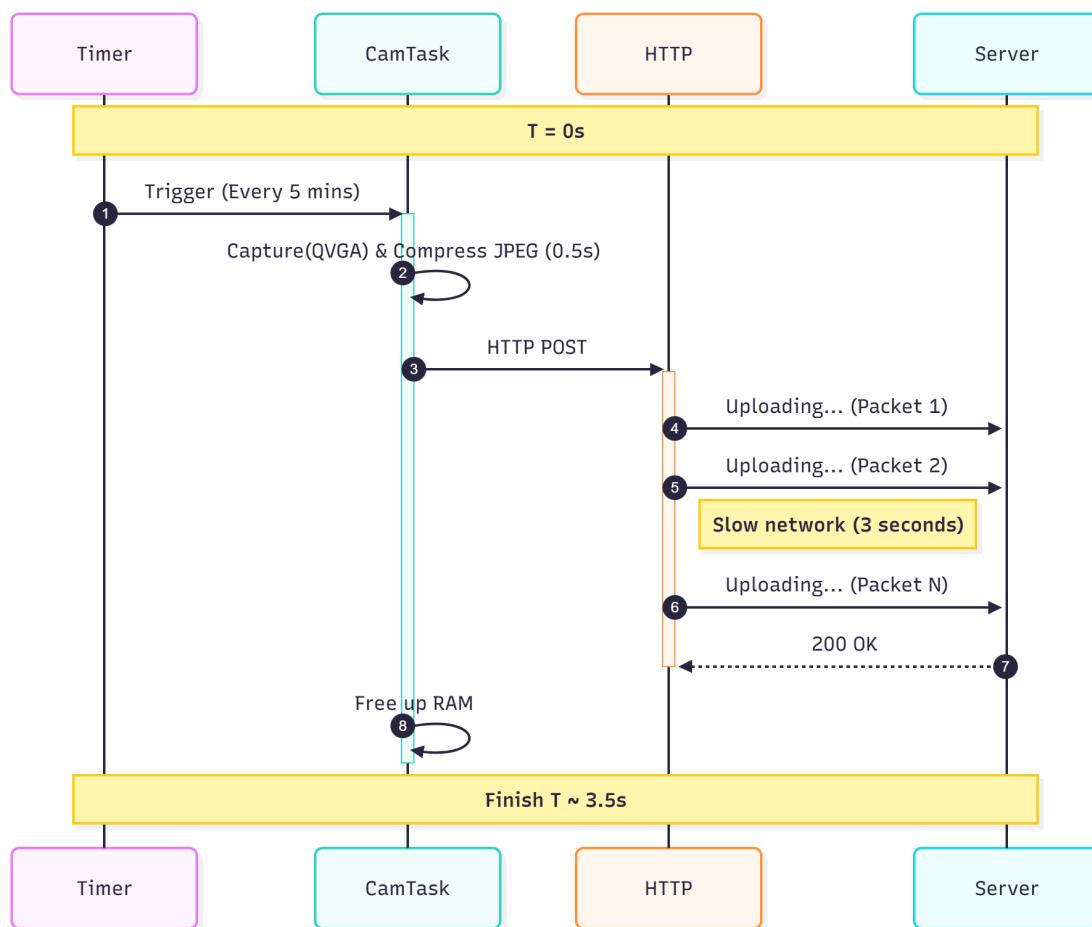
Chi tiết quy trình:

- PHASE 1: Monitoring (Giám sát - Chu kỳ 5 giây)
 - **Kích hoạt**: System Timer gửi tín hiệu Trigger Loop sau mỗi 5000ms (5 giây).
 - **Đọc dữ liệu**: Task 1 thực hiện đọc liên tiếp hai loại cảm biến: DHT20 và Soil (cảm biến đất).
 - **Xử lý dữ liệu**: Sau khi nhận giá trị (return value), hệ thống chạy bước SensorValidator. Đây là điểm quan trọng để kiểm tra dữ liệu có bị lỗi (NULL hoặc giá trị ảo) hay không trước khi gửi đi.
 - **Truyền tin**: Dữ liệu sau khi kiểm tra được đóng gói thành định dạng JSON (ví dụ: "temp":32, "soil":40...) và Publish lên MQTT.
 - **Cập nhật**: MQTT chuyển dữ liệu này tới App để người dùng theo dõi thời gian thực.

- PHASE 2: Controlling (Diều khiển - Theo sự kiện)

- **Lệnh từ người dùng:** Từ App, người dùng nhấn nút bơm, gửi lệnh "PUMP: ON" tới MQTT.
- **Nhận lệnh:** MQTT chuyển gói tin này tới ESP32.
- **Cơ chế Callback:** Tại Task 1, hàm client.loop() liên tục kiểm tra tín hiệu đến. Khi thấy tin nhắn, nó kích hoạt hàm Callback.
- **Thực thi vật lý:** Task 1 thực hiện lệnh digitalWrite(PIN_PUMP, HIGH) để đóng Relay, kích hoạt bơm.
- **Xác nhận (Feedback Loop):**
 - * Sau khi Relay phản hồi "Done", Task 1 gửi ngược lại MQTT một bản tin Publish "Status: ON".
 - * MQTT cập nhật trạng thái này lên App để hiển thị nút bấm đã chuyển sang màu xanh (hoặc trạng thái ON).

Figure 2: Camera Process Cycle



Hình 18: Timing Diagram for camera task

Các thành phần tham gia:

- **Timer:** Bộ định thời, đóng vai trò kích hoạt sự kiện theo chu kỳ.
- **CamTask:** Tác vụ xử lý camera (chụp ảnh, nén dữ liệu).

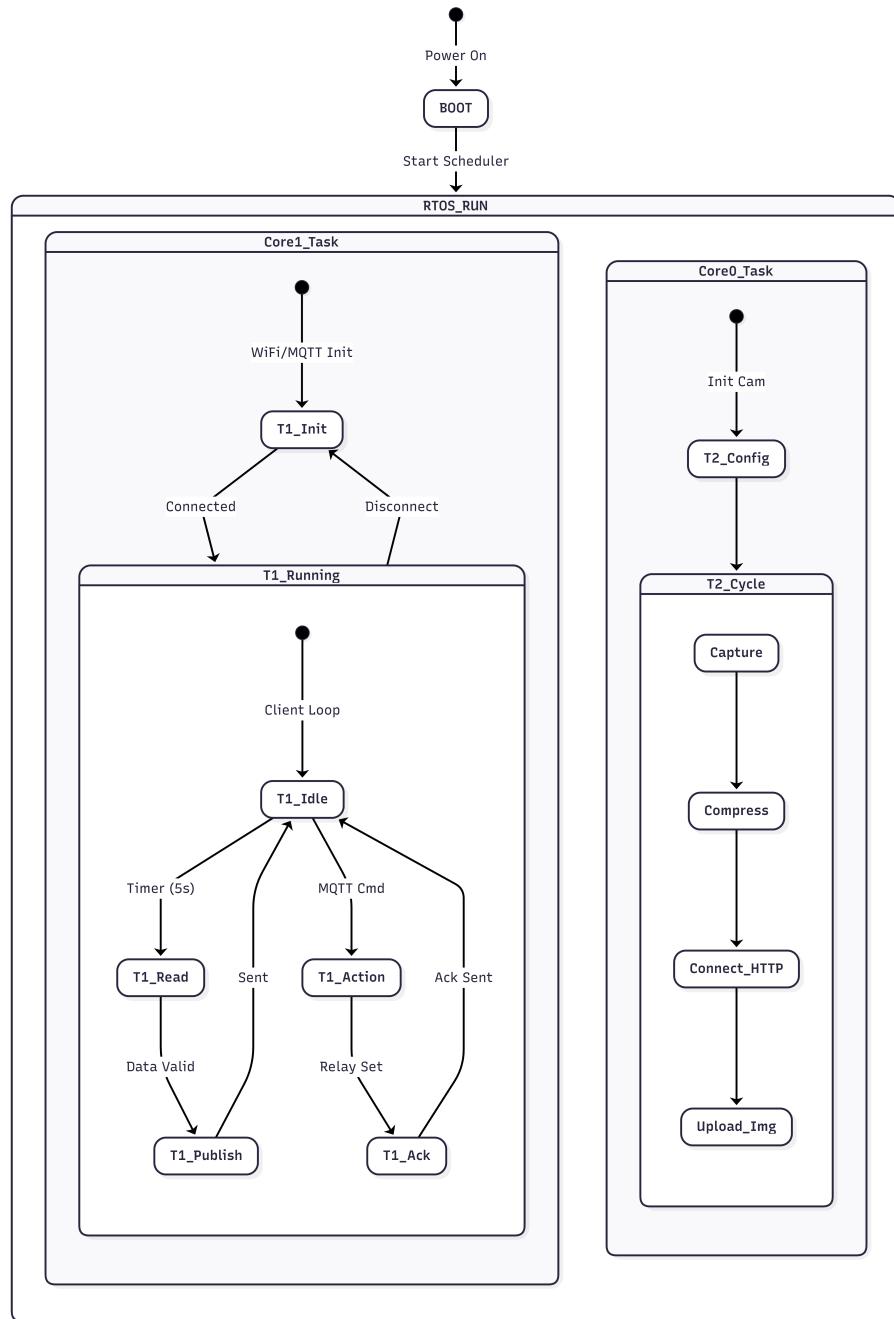


- **HTTP:** Giao thức/Thành phần đảm nhận việc truyền tải dữ liệu qua mạng.
- **Server:** Máy chủ từ xa tiếp nhận và lưu trữ hình ảnh.

Chi tiết quy trình xảy ra trong một chu kỳ mỗi 3500ms:

- **Trigger (Every 5 mins):** Cứ mỗi 5 phút, Timer gửi một tín hiệu kích hoạt cho CamTask. Đây là cơ chế tiết kiệm năng lượng, thay vì chạy liên tục, thiết bị chỉ thức dậy để làm việc rồi có thể nghỉ (Deep Sleep).
- **Capture (QVGA) & Compress JPEG (0.5s):** CamTask thực hiện chụp ảnh ở độ phân giải QVGA (320×240 pixel) và nén sang định dạng JPEG. Quá trình này tiêu tốn 0.5s.
- **HTTP POST:** Sau khi có file ảnh đã nén, CamTask chuyển dữ liệu cho thành phần HTTP để chuẩn bị gửi đi bằng phương thức POST.
- **Uploading... (Packet 1):** Dữ liệu ảnh được chia nhỏ thành nhiều gói tin (Packets) để gửi đi. Gói tin đầu tiên được gửi tới Server.
- **Slow network (3 seconds):** Đây là điểm đáng chú ý nhất. Sơ đồ giả định một tình huống mạng chậm. Việc gửi các gói tin (từ Packet 2 đến Packet N) kéo dài tới 3 giây.
- **Uploading... (Packet N):** Gói tin cuối cùng được gửi hoàn tất.
- **200 OK:** Sau khi nhận đủ các gói tin và xử lý xong, Server phản hồi mã trạng thái 200 OK, xác nhận việc tải ảnh lên đã thành công.
- **Free up RAM:** Sau khi nhận xác nhận từ server, CamTask thực hiện giải phóng bộ nhớ (RAM) đã dùng để chứa bức ảnh. Đây là bước cực kỳ quan trọng trong lập trình nhúng (như ESP32) để tránh lỗi tràn bộ nhớ (Memory Leak).

5.9.2.d State Machine Diagram



Hình 19: State Machine Diagram

Biểu đồ mô tả hoạt động của Firmware trên nền tảng FreeRTOS, trong đó trạng thái vận hành chính (RTOS_RUN) được chia thành hai luồng tác vụ chạy song song trên hai lõi CPU:

- Core 1 Task (Cảm biến & Kết nối):** Chịu trách nhiệm duy trì kết nối WiFi/MQTT. Hệ thống chuyển đổi linh hoạt giữa trạng thái Khởi tạo (T1_Init) và Vận hành (T1_Running). Trong trạng thái vận hành, vi điều khiển hoạt động theo cơ chế hướng sự kiện (Event-driven): tự động đọc cảm biến khi hết Timer (5s) hoặc xử lý lệnh điều khiển ngay khi nhận được tín hiệu từ MQTT.
- Core 0 Task (Camera):** Hoạt động độc lập theo chu trình tuần tự khép kín (T2_Cycle): Chụp ảnh → Nén dữ liệu → Tải lên qua HTTP.



5.10 Cách đánh giá giải pháp

Để đảm bảo tính khoa học và thực tiễn của đề tài, nhóm nghiên cứu đề xuất phương pháp đánh giá hệ thống dựa trên 3 trụ cột chính: Hiệu năng hệ thống (System Performance), Độ chính xác của thuật toán (Algorithmic Accuracy) và Trải nghiệm người dùng (User Experience).

5.10.1 Đánh giá Hiệu năng Hệ thống (Performance Evaluation)

Phần này tập trung vào khả năng chịu tải và độ ổn định của hạ tầng IoT (RabbitMQ, NestJS, TimescaleDB). Nhóm sẽ sử dụng các công cụ kiểm thử tự động (như JMeter hoặc Script mô phỏng) để giả lập tải.

- Kích bản thử nghiệm:** Giả lập 50 thiết bị ảo gửi dữ liệu liên tục với tần suất 5 giây/gói tin trong thời gian 60 phút.
- Các chỉ số đo lường (Metrics):**
 - Throughput (Thông lượng):* Số lượng bản tin xử lý được trên giây (msg/s).
 - End-to-End Latency:* Thời gian trễ từ khi thiết bị gửi dữ liệu đến khi hiển thị trên Dashboard (Mục tiêu: < 5 giây).
 - Resource Usage:* Mức tiêu thụ CPU và RAM của Server (Docker Containers) khi chịu tải đỉnh.
 - Packet Loss Rate:* Tỷ lệ gói tin bị mất tại Message Broker.

5.10.2 Đánh giá Hiệu quả Thuật toán Phát hiện lỗi (Algorithmic Evaluation)

Đối với module *Sensor Fault Detection*, nhóm sẽ đánh giá dựa trên phương pháp "Tiêm lỗi" (Fault Injection) - chủ động tạo ra các dữ liệu sai lệch để kiểm tra khả năng phát hiện của hệ thống.

Bảng 9: Các kịch bản đánh giá thuật toán phát hiện lỗi

STT	Loại lỗi giả lập	Tiêu chí Đạt (Pass Criteria)
1	Mất kết nối (Hard Fault): Ngắt nguồn thiết bị đột ngột.	Hệ thống cập nhật trạng thái "Offline" và gửi cảnh báo trong vòng 3 chu kỳ gửi tin (khoảng 15s).
2	Dữ liệu bất thường (Outlier): Gửi giá trị nhiệt độ đột biến (ví dụ: 100°C).	Hệ thống phát hiện giá trị vượt ngưỡng (Threshold) và ghi nhận cảnh báo.
3	Dữ liệu đóng băng (Stuck): Gửi liên tục một giá trị không đổi trong 30 phút.	Hệ thống phát hiện phương sai (Variance) bằng 0 và cảnh báo lỗi cảm biến bị kẹt.

5.10.3 Đánh giá Trải nghiệm và Chức năng (Functional & UX Evaluation)

Đánh giá mức độ hoàn thiện của sản phẩm đối với người dùng cuối (System Admin/Thương lái) thông qua kiểm thử chấp nhận (UAT - User Acceptance Testing).

- Phương pháp:** Thực hiện danh sách kiểm tra (Checklist) các chức năng nghiệp vụ cốt lõi trên giao diện Web Admin.
- Các chỉ số đánh giá:**
 - Tỷ lệ hoàn thành tác vụ (Task Completion Rate):* Người dùng có thực hiện được các thao tác (Thêm Farm, Gán thiết bị, Xem báo cáo) mà không gặp lỗi không?
 - Thời gian phản hồi giao diện (UI Response Time):* Các thao tác chuyển trang, load dữ liệu lịch sử phải hoàn tất dưới 2 giây.
 - Tính trực quan:* Biểu đồ dữ liệu và các cảnh báo lỗi có dễ hiểu và dễ nhận biết hay không.



6 Kế hoạch thực hiện

6.1 Mục tiêu theo từng giai đoạn

6.1.0.a Firmware

- **Giai đoạn 1:** Nghiên cứu và Chuẩn bị (Tuần 1 - Tuần 4)
 - Nghiên cứu lý thuyết về kiến trúc ESP32 (Dual-core), giao thức Camera DVP, MQTT và FreeRTOS.
 - Lựa chọn và mua sắm linh kiện phần cứng (ESP32-S3/Wrover, OV2640, DHT20, Relay, cảm biến đất).
 - Thiết kế sơ đồ nguyên lý và lắp ráp mạch thử nghiệm trên Breadboard.(Bên thứ 3 hỗ trợ)
- **Giai đoạn 2:** Phát triển và Tích hợp hệ thống (Tuần 5 - Tuần 11)
 - Phần cứng: Hoàn thiện mạch in (PCB) hoặc mạch hàn thủ công chắc chắn.(Bên thứ 3 hỗ trợ)
 - Firmware: Viết chương trình điều khiển trung tâm sử dụng FreeRTOS. Tách tác vụ xử lý ảnh (Core 0) và tác vụ điều khiển/kết nối (Core 1).
 - Kết nối: Xây dựng giao thức truyền nhận JSON qua MQTT và HTTP Streaming.
- **Giai đoạn 3:** Kiểm thử, Tối ưu và Viết báo cáo (Tuần 12 - Tuần 15)
 - Chạy thử nghiệm hệ thống trong môi trường thực tế (vườn mô phỏng).
 - Đo đặc thông số: Độ trễ (latency), độ ổn định WiFi, nhiệt độ chip.
 - Viết báo cáo thuyết minh và chuẩn bị slide bảo vệ.

6.1.0.b Software

- **Giai đoạn 1:** Khởi tạo nền tảng và Cơ sở dữ liệu (Tuần 1 - Tuần 4)
 - Thiết lập môi trường phát triển (Docker, Git).
 - Xây dựng cấu trúc Database: Quan hệ (PostgreSQL) cho quản lý User/Farm và Chuỗi thời gian (TimescaleDB) cho dữ liệu cảm biến.
 - Xây dựng module xác thực (Authentication) và phân quyền (RBAC) làm nền tảng bảo mật.
- **Giai đoạn 2:** Phát triển Backend Core và Tích hợp IoT (Tuần 5 - Tuần 9)
 - Phát triển các API quản lý nghiệp vụ (Farm, Device Management).
 - Xây dựng IoT Data Processor: Dịch vụ tiêu thụ dữ liệu từ RabbitMQ, xử lý Validate Schema (JSON) và lưu trữ vào TimescaleDB.
 - Triển khai cơ chế cảnh báo thời gian thực (Real-time Alert).
- **Giai đoạn 3:** Phát triển Frontend và Dashboard (Tuần 10 - Tuần 13)
 - Xây dựng giao diện người dùng (Web Admin).
 - Trực quan hóa dữ liệu: Biểu đồ nhiệt độ/độ ẩm, thống kê trạng thái thiết bị.
 - Tích hợp module Báo cáo (Report) và Xử lý sự cố (Troubleshoot).
- **Giai đoạn 4:** Kiểm thử, Tối ưu và Triển khai (Tuần 14 - Tuần 15)
 - Kiểm thử tải (Load Testing) với hàng nghìn request giả lập.
 - Tối ưu truy vấn Database (Indexing, Compression).
 - Đóng gói (Containerization) và viết tài liệu hướng dẫn sử dụng.



6.2 Lịch trình và mốc thời gian

6.2.0.a Firmware

Bảng 10: Bảng phân bổ lịch trình thực hiện đồ án 15 tuần

Tuần	Công việc chi tiết	Kết quả bàn giao
1-2	<ul style="list-style-type: none">Nghiên cứu Datasheet ESP32, OV2640.Tìm hiểu thư viện ArduinoJson, PubSubClient.Dặt mua linh kiện phần cứng (Bên thứ 3 hỗ trợ).	<ul style="list-style-type: none">Chương 1 (Cơ sở lý thuyết).Danh sách linh kiện đầy đủ.
3	<ul style="list-style-type: none">Thiết kế sơ đồ khối và sơ đồ nguyên lý (Schematic) (Bên thứ 3 hỗ trợ).Kiểm tra hoạt động riêng lẻ của module cảm biến, Relay.	<ul style="list-style-type: none">Bản vẽ sơ đồ nguyên lý.Code kiểm thử đơn giản.
4	<ul style="list-style-type: none">Lắp ráp phần cứng hoàn chỉnh (trên mạch in hoặc đục lỗ) (Bên thứ 3 hỗ trợ).Kiểm tra nguồn, đo đặc chống nhiễu (Bên thứ 3 hỗ trợ).	<ul style="list-style-type: none">Mô hình phần cứng thô.Chương 2 (Thiết kế phần cứng).
5-6	<ul style="list-style-type: none">Cấu hình FreeRTOS: Tạo Task, Queue, Semaphore.Lập trình đọc cảm biến (DHT, Đất) và điều khiển Relay.	<ul style="list-style-type: none">Code khung sườn.Dữ liệu hiển thị trên Serial Monitor.
7-8	<ul style="list-style-type: none">Xử lý Camera: Cấu hình giao thức DVP, lấy dữ liệu ảnh JPEG.Lập trình Web Server để stream ảnh qua HTTP.	<ul style="list-style-type: none">Hình ảnh hiển thị trên trình duyệt.Frame rate đạt mức ổn định (>10fps).
9	<ul style="list-style-type: none">Kết nối IoT: Lập trình giao thức MQTT.Đóng gói dữ liệu dạng JSON.	<ul style="list-style-type: none">Gửi thành công JSON lên Broker.Nhận lệnh điều khiển từ xa.
10-11	<ul style="list-style-type: none">Tích hợp hệ thống: Ghép module Camera + Cảm biến + MQTT.Xử lý xung đột tài nguyên giữa 2 nhân (Core 0/1).	<ul style="list-style-type: none">Firmware hoàn chỉnh v1.0.0.Chương 3 (Thiết kế phần mềm).
12	<ul style="list-style-type: none">Chạy thử nghiệm hệ thống liên tục (Stress test).Tinh chỉnh ngưỡng cảm biến, tối ưu bộ nhớ.	<ul style="list-style-type: none">Số liệu thực nghiệm.Bảng đánh giá độ ổn định.
13	<ul style="list-style-type: none">Tổng hợp số liệu, vẽ biểu đồ kết quả.	<ul style="list-style-type: none">Chương 4 (Kết quả thực nghiệm).
14	<ul style="list-style-type: none">Hoàn thiện toàn bộ báo cáo thuyết minh.Chỉnh sửa định dạng, trích dẫn tài liệu tham khảo.	<ul style="list-style-type: none">Bản thảo báo cáo hoàn chỉnh.
15	<ul style="list-style-type: none">Soạn thảo Slide thuyết trình.Quay video demo và luyện tập bảo vệ.	<ul style="list-style-type: none">Slide Powerpoint.Video Demo sản phẩm.



6.2.0.b Software

Tuần	Công việc chi tiết	Kết quả bàn giao
1-2	Khởi tạo nền tảng: - Thiết kế ERD vật lý cho PostgreSQL và TimescaleDB. - Cài đặt Docker (Postgres, RabbitMQ). - Khởi tạo cấu trúc dự án NestJS (Monorepo).	- File docker-compose.yml. - Script khởi tạo Database.
3-4	Quản lý người dùng (Auth): - Phát triển API Đăng ký, Đăng nhập (JWT). - Xây dựng cơ chế phân quyền RBAC (Admin/User).	- API Documentation (Swagger). - Sơ đồ phân quyền hoạt động.
5-6	Quản lý Farm & Device: - API CRUD Nông trại, Khu vực. - API Đăng ký thiết bị và gán vào Farm.	- Các API quản lý chính hoàn thiện. - Dữ liệu lưu trữ thành công vào Postgres.
7-8	Tích hợp IoT (Core): - Cấu hình RabbitMQ Consumer trong NestJS. - Xử lý Validate JSON Schema đầu vào. - Lưu trữ dữ liệu cảm biến vào TimescaleDB.	- Dữ liệu từ Queue được lưu vào bảng Hypertable. - Log được các gói tin lõi định dạng.
9	Cảnh báo thời gian thực: - Xây dựng logic so sánh ngưỡng (Threshold). - Tích hợp Socket.io để đẩy thông báo lên Web.	- Cảnh báo hiển thị ngay lập tức khi vượt ngưỡng.
10	Frontend - Cơ bản: - Dụng khung Web Admin (React). - Giao diện Login, Quản lý Farm/Device.	- Giao diện quản lý cơ bản hoạt động.
11-12	Frontend - Nâng cao: - Vẽ biểu đồ (Charts) nhiệt độ/độ ẩm. - Trang Troubleshoot và Xuất báo cáo.	- Dashboard hiển thị trực quan. - Chức năng Export PDF/Excel.
13	Tích hợp hệ thống: - Ghép nối ESP32 (thực/ảo) - Backend - Web. - Kiểm thử luồng dữ liệu E2E.	- Hệ thống hoạt động thông suốt từ cảm biến lên Web.
14	Tối ưu hóa: - Indexing và Materialized Views cho DB. - Cấu hình nén dữ liệu TimescaleDB.	- Báo cáo hiệu năng hệ thống (Response time).
15	Đóng gói & Báo cáo: - Deploy lên Server thực tế. - Viết tài liệu hướng dẫn sử dụng.	- Link Demo sản phẩm. - Slide thuyết trình.



6.3 Rủi ro và phương án giảm thiểu

6.3.0.a Firmware

Bảng 12: Bảng phân tích rủi ro và phương án giảm thiểu

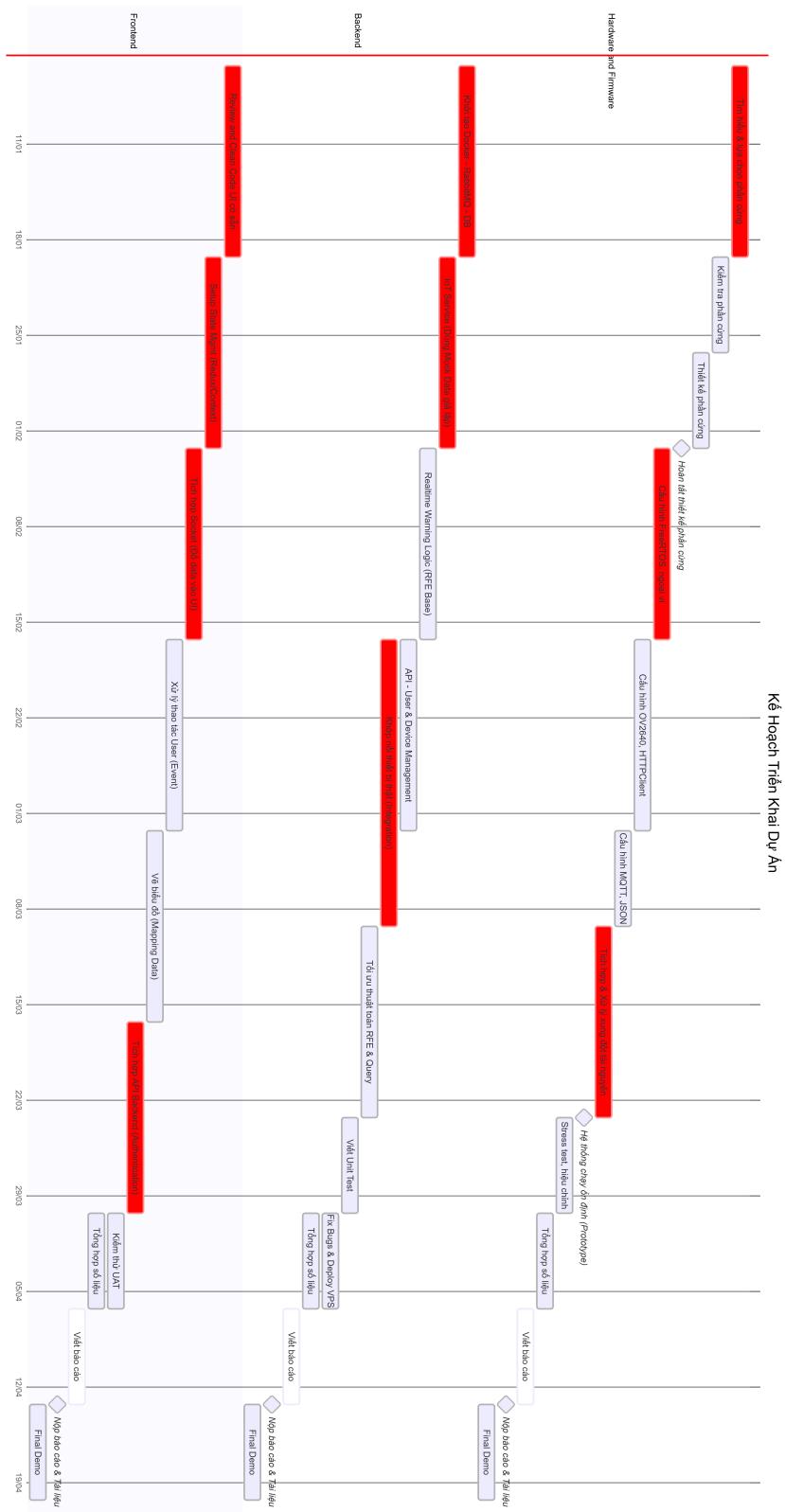
STT	Rủi ro (Risk)	Mức độ	Phương án giảm thiểu
1	Hỏng hóc phần cứng: Cháy ESP32 hoặc Camera do đấu sai nguồn/ngắn mạch.	Cao	<ul style="list-style-type: none">Mua dự phòng 01 bộ linh kiện.Kiểm tra kỹ mạch bằng VOM trước khi cấp nguồn.
2	Độ trễ truyền ảnh cao: Video bị giật, lag khi mạng WiFi yếu.	Trung bình	<ul style="list-style-type: none">Tách luồng gửi ảnh sang Core 0 độc lập.
3	Chậm tiến độ: Do vướng mắc thuật toán phức tạp.	Thấp	<ul style="list-style-type: none">Ưu tiên xử lý phần Camera trước (phần khó nhất).Tham khảo cộng đồng Open Source ESP32.

6.3.0.b Software

Bảng 13: Phân tích rủi ro phần mềm và phương án xử lý

STT	Rủi ro (Risk)	Mức độ	Phương án giảm thiểu
1	Nghẽn cổ chai Database: Khi hàng trăm thiết bị gửi dữ liệu cùng lúc, việc Insert từng dòng gây quá tải.	Cao	<ul style="list-style-type: none">Sử dụng kỹ thuật Batch Insert (gom 100-500 bản ghi/lần).Tận dụng kiến trúc Hypertables của TimescaleDB để phân mảnh dữ liệu.
2	Mất toàn vẹn dữ liệu (Data Inconsistency): Lỗi xảy ra giữa chừng khi tạo Nông trại và gán Thiết bị.	Cao	<ul style="list-style-type: none">Sử dụng Database Transaction (ACID) để đảm bảo "All or Nothing".Cơ chế Rollback tự động khi có lỗi.
3	Gián đoạn thời gian thực: Kết nối WebSocket bị ngắt khiến Dashboard không cập nhật số liệu.	Trung bình	<ul style="list-style-type: none">Cơ chế Heartbeat (Ping/Pong) để phát hiện mất kết nối.Tự động kết nối lại (Auto-reconnect) ở phía Frontend React.
4	Mất dữ liệu tại Broker: RabbitMQ bị đầy hàng đợi (Queue overflow) nếu Backend xử lý chậm.	Trung bình	<ul style="list-style-type: none">Cấu hình Message Acknowledgment (chỉ xóa tin khi đã xử lý xong).Tăng số lượng Worker (Consumer) trong NestJS để xử lý song song.
5	Dữ liệu rác (Invalid Schema): Thiết bị gửi sai định dạng JSON làm lỗi parser hệ thống.	Cao	<ul style="list-style-type: none">Triển khai lớp Validation (DTO) chặt chẽ tại đầu vào (Gateway).Ghi log gói tin lỗi ra bảng riêng (Dead Letter Queue) để debug.
6	Sự cố Vận hành (Deployment): Docker Container bị crash do lỗi bộ nhớ hoặc lỗi runtime.	Cao	<ul style="list-style-type: none">Cấu hình Docker Restart Policy (<code>restart: always</code>).Thiết lập Health Check định kỳ cho các service.

6.4 Gantt chart



Hình 20: Biểu đồ Gantt thể hiện tiến độ thực hiện đề tài



7 Kết luận

7.1 Tóm tắt vấn đề và hướng tiếp cận

Dề tài "Hệ thống quản lý nông trại thông minh" được phát triển nhằm giải quyết bài toán thực tế về sự phân mảnh trong quản lý chuỗi nông trại của các thương lái và nhà đầu tư nông nghiệp. Hướng tiếp cận chủ đạo của nhóm là xây dựng một nền tảng quản lý tập trung (All-in-one), tích hợp khả năng giám sát dữ liệu thời gian thực từ thiết bị đa nguồn, đồng thời đơn giản hóa trải nghiệm vận hành cho đối tượng người dùng không chuyên về kỹ thuật.

7.2 Công việc đã hoàn thành trong giai đoạn Đồ án Chuyên ngành

Trong khuôn khổ giai đoạn thiết kế, nhóm đã hoàn thành các nhiệm vụ nền tảng sau:

- Khảo sát và Phân tích:** Đã thực hiện khảo sát các giải pháp hiện có, từ đó xác định rõ yêu cầu bài toán gồm các chức năng quản lý cốt lõi và các ràng buộc phi chức năng về hiệu năng, độ trễ.
- Thiết kế Kiến trúc tổng thể:** Đã đề xuất mô hình kiến trúc phân lớp (Layered Architecture) hoàn chỉnh, kết hợp sức mạnh của **NestJS** (Backend), **RabbitMQ** (Message Queuing) và **TimescaleDB** (Time-series Data). Đây là nền tảng vững chắc để đảm bảo khả năng mở rộng (Scalability) và tính ổn định khi số lượng thiết bị gia tăng.
- Thiết kế Chi tiết Module IoT:** Đã hoàn thiện thiết kế Firmware trên **ESP32** với cơ chế đa luồng (FreeRTOS) và phân tích kỹ lưỡng các rủi ro phần cứng (tràn bộ nhớ, nhiễu tín hiệu) cùng phương án giảm thiểu.
- Lập kế hoạch hiện thực:** Đã xây dựng lộ trình chi tiết cho giai đoạn Đồ án Tốt nghiệp, bao gồm các mốc thời gian kiểm thử và triển khai cụ thể cho từng thành viên.

7.3 Định hướng thực hiện Đồ án Tốt nghiệp

Giai đoạn tiếp theo sẽ tập trung vào việc hiện thực hóa bản thiết kế thành sản phẩm chạy thực tế:

1. Hiện thực hóa Firmware & Phần cứng:

- Lập trình Firmware ESP32 sử dụng FreeRTOS, tích hợp đọc dữ liệu song song từ đa cảm biến (DHT20, độ ẩm đất) và Camera OV2640.
- Cài đặt và tinh chỉnh thuật toán phát hiện lỗi cảm biến (sử dụng phương pháp thống kê hoặc RFE) để tự động nhận diện các bất thường như trôi số liệu (Drift) hoặc mất tín hiệu.
- Tối ưu hóa giao thức MQTT để giảm thiểu độ trễ truyền tin.

2. Xây dựng Hệ thống Backend & Frontend:

- Phát triển bộ RESTful API và các Worker xử lý dữ liệu IoT chuyên biệt thông qua RabbitMQ.
- Xây dựng Web Dashboard với ReactJS, tập trung vào tính năng trực quan hóa dữ liệu và hệ thống cảnh báo thời gian thực (Real-time Alerting).

3. Kiểm thử và Tối ưu hóa:

- Thực hiện Load Testing với mạng lưới thiết bị mô phỏng quy mô vừa và lớn để đánh giá giới hạn chịu tải của hệ thống.
- Triển khai hệ thống tại mô hình thực nghiệm (Pilot Test) để thu thập dữ liệu thật, từ đó đánh giá độ chính xác của thuật toán phát hiện lỗi và độ ổn định của phần cứng.

4. Các chỉ số kỹ thuật mục tiêu (Target Performance Metrics):

Dể đảm bảo tính định lượng cho việc đánh giá kết quả vào cuối kỳ, nhóm thiết lập các chỉ số KPI mục tiêu như sau:



Tiêu chí	Mô tả	Mục tiêu (Target)
Độ trễ (Latency)	Thời gian từ khi thiết bị gửi dữ liệu đến khi hiển thị trên Dashboard.	< 3 giây (với mạng 4G/WiFi tiêu chuẩn).
Khả năng chịu tải	Số lượng thiết bị gửi dữ liệu đồng thời mà không gây mất gói tin.	50 thiết bị (giả lập) với tần suất 5 giây/gói tin.
Độ chính xác AI	Tỷ lệ phát hiện đúng các lỗi cảm biến cơ bản (mất nguồn, gai dữ liệu).	> 85% trên tập dữ liệu kiểm thử.
Tốc độ phản hồi Web	Thời gian tải trang và hiển thị biểu đồ lịch sử.	< 2 giây (cho truy vấn 7 ngày gần nhất).
Độ ổn định	Thời gian hoạt động liên tục không lỗi (Up-time) trong môi trường thử nghiệm.	99% (trong 48 giờ chạy test liên tục).

7.4 Bài học kinh nghiệm và Góc nhìn phản tư

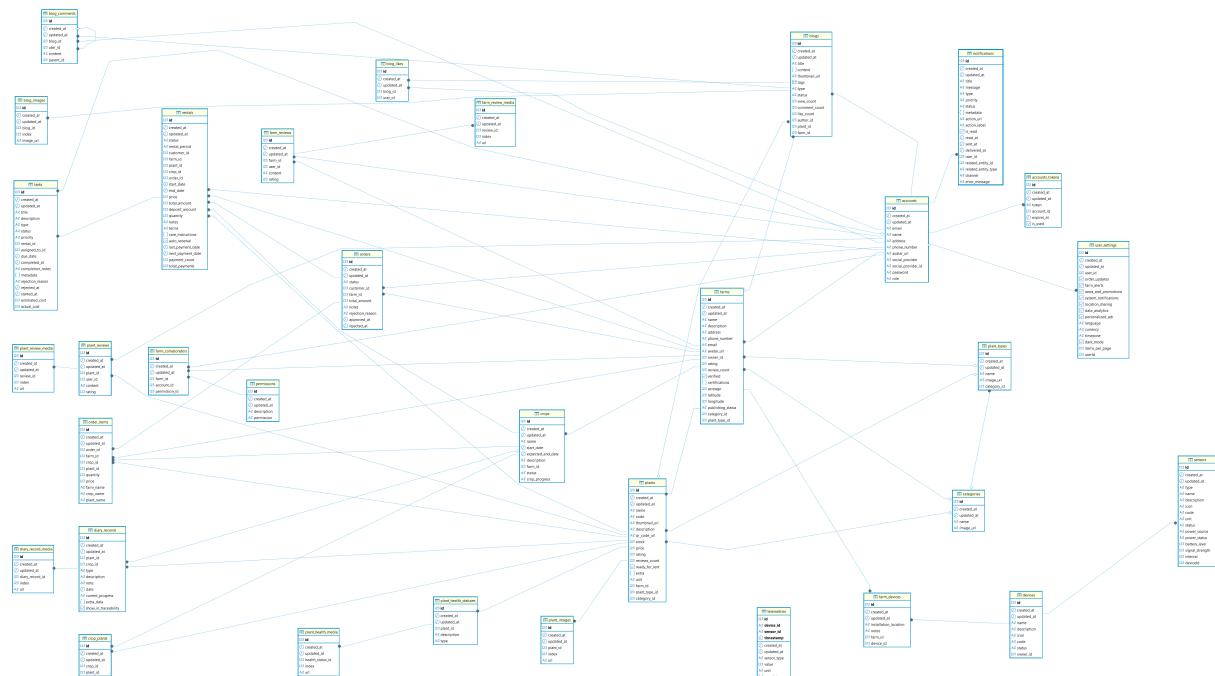
Qua quá trình nghiên cứu, nhóm nhận thức sâu sắc rằng thách thức lớn nhất của dự án IoT không chỉ nằm ở việc lập trình, mà ở việc **thiết kế kiến trúc hệ thống** sao cho linh hoạt và chịu lỗi tốt ngay từ đầu.

Nhóm đã rút ra bài học quan trọng về việc sử dụng cơ chế hàng đợi (Message Queue) để tách rời các thành phần hệ thống (Decoupling), giúp tránh nghẽn cổ chai khi lưu lượng dữ liệu tăng cao. Ngoài ra, việc phân tích rủi ro phần cứng ngay từ giai đoạn thiết kế sẽ giúp giảm thiểu đáng kể thời gian Debug và chi phí phát sinh khi triển khai thực tế.

Dự án này là bước đệm quan trọng, không chỉ mang giá trị học thuật mà còn hướng tới một sản phẩm có tính ứng dụng cao, góp phần giải quyết bài toán chuyển đổi số trong nông nghiệp một cách thiết thực.

A Sơ đồ Cơ sở dữ liệu vật lý (Physical ERD)

Hình dưới đây mô tả chi tiết toàn bộ lược đồ cơ sở dữ liệu của hệ thống, bao gồm các bảng, các trường dữ liệu, kiểu dữ liệu và mối quan hệ giữa chúng.



Hình 21: Sơ đồ thực thể quan hệ chi tiết (Physical Data Model)

B Danh sách các API chính (API Specifications)

Phần này cung cấp tài liệu kỹ thuật chi tiết về các API Endpoint đã được thiết kế cho hệ thống Backend, được trích xuất trực tiếp từ giao diện Swagger UI.

Các hình ảnh dưới đây mô tả rõ ràng phương thức HTTP (GET, POST, PUT, PATCH, DELETE), đường dẫn (URI) và mô tả chức năng ngắn gọn cho từng Endpoint thuộc hai module quan trọng nhất là Quản lý Nông trại (Farms) và Quản lý Thiết bị (Devices).



The screenshot shows the 'Farms' endpoint documentation. It lists various API operations:

- GET /api/v1/farms/{id}/plants
- POST /api/v1/farms/{id}/plants
- PATCH /api/v1/farms/{id}/plants/{plant_id}
- GET /api/v1/farms/{id}/plants/{plant_id}
- GET /api/v1/farms/{id}
- PATCH /api/v1/farms/{id}
- DELETE /api/v1/farms/{id}
- GET /api/v1/farms/{id}/devices
- GET /api/v1/farms
- POST /api/v1/farms
- POST /api/v1/farms/{id}/certifications
- GET /api/v1/farms/{id}/plants/{plant_id}/diaries
- POST /api/v1/farms/{id}/plants/{plant_id}/diaries
- POST /api/v1/farms/{id}/reviews

Hình 22: Đặc tả API Module Quản lý Nông trại (Farms Endpoint) [Nguồn: Swagger UI Hệ thống]

The screenshot shows the 'Devices' endpoint documentation. It lists various API operations:

- PUT /api/v1/notifications/{id}/status
- Devices
 - GET /api/v1/devices
 - POST /api/v1/devices
 - GET /api/v1/devices/{id}
 - PATCH /api/v1/devices/{id}
 - PUT /api/v1/devices/{id}
 - DELETE /api/v1/devices/{id}
 - POST /api/v1/devices/assign-to-farm
 - DELETE /api/v1/devices/{deviceId}/farms/{farmId}
 - GET /api/v1/devices/admin/all
 - GET /api/v1/devices/admin/{id}
 - PATCH /api/v1/devices/admin/{id}
 - PUT /api/v1/devices/admin/{id}
 - DELETE /api/v1/devices/admin/{id}
 - POST /api/v1/devices/admin

Hình 23: Đặc tả API Module Quản lý Thiết bị (Devices Endpoint) [Nguồn: Swagger UI Hệ thống]

Nhìn vào các hình trên, có thể thấy hệ thống đã thiết kế đầy đủ các thao tác CRUD (Tạo, Đọc, Cập nhật, Xóa) cho Nông trại và Thiết bị, cũng như các API nghiệp vụ đặc thù như gán thiết bị vào nông trại ('/assign-to-farm') hay lấy lịch sử dữ liệu cảm biến.

C Danh sách linh kiện phần cứng (Bill of Materials)

Bảng liệt kê các linh kiện được sử dụng để xây dựng Node IoT:



Bảng 14: Bill of Materials (BOM)

STT	Tên linh kiện	Thông số kỹ thuật chính	Số lượng
1	ESP32-S3 WROOM	Dual-core 240MHz, WiFi/BLE	01
2	Cảm biến DHT20	Giao tiếp I2C, đo nhiệt độ/độ ẩm	01
3	Camera OV2640	2 Megapixel, giao tiếp DVP	01
4	Cảm biến độ ẩm đất	Capacitive (Điện dung), Analog	01
5	Relay Module	5VDC - 1 kênh, Opto cách ly	02
6	Module quạt	3.3VDC	01
7	Nguồn Adapter	5VDC	01

D Bảng so sánh các giải pháp tham khảo



Tài liệu

- [1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.
- [2] et al. Bader, A. Time series databases: New ways to store and access data. *Proceedings of the VLDB Endowment*, 2017.
- [3] et al. Boursianis, A. D. Smart farming: Internet of things technologies, challenges and future directions. *Sensors*, 22(10):3735, 2022.
- [4] Báo Chính Phủ. Thủ tướng: Ngành nông nghiệp phải tăng tốc, bứt phá, xuất khẩu 70 tỷ usd trong năm 2025. *Cổng Thông tin điện tử Chính phủ*, 2024.
- [5] Tran Nguyen Minh Duy. *Improving Feature Extraction for Sensor Fault Detection in Low-Power IoT Systems*, 2025.
- [6] et al. Fielding, R. Hypertext transfer protocol (http/1.1): Message syntax and routing. Request for Comments 7230, IETF, 2014.
- [7] FreeRTOS. *FreeRTOS Scheduler and Scheduling Policies*, 2022.
- [8] PostgreSQL Global Development Group. *PostgreSQL 16.0 Documentation*, 2024.
- [9] Timescale Inc. *TimescaleDB Documentation: Hypertables and Chunking*, 2024. Truy cập: 13/12/2025.
- [10] ECMA International. *The JSON Data Interchange Syntax (ECMA-404)*, 2017.
- [11] Microsoft. *TypeScript: Typed JavaScript at Any Scale*, 2024.
- [12] M. Noura, M. Atiquzzaman, and M. Gaedke. Interoperability in iot: Challenges and solutions. In *Proceedings of the International Conference on IoT*, pages 1–6. Springer, 2019.
- [13] OASIS. *MQTT Version 5.0 Committee Specification*, 2019.
- [14] Inc. OmniVision Technologies. *OV2640 Camera Module Datasheet*, 2020.
- [15] Vietnam Report. Báo cáo tăng trưởng và 6 thách thức của nông nghiệp công nghệ cao việt nam, 2024.
- [16] NXP Semiconductors. *I2C Bus Specification and User Manual*, 2021.
- [17] J. Smith and T. Nguyen. Intelligent fault detection in iot-based wsns for precision agriculture using machine learning. *IEEE Internet of Things Journal*, 12(4):1023–1035, 2025.
- [18] et al. Stapleton, P. A review of low-cost precision agriculture: sensors, monitoring and communication systems. *Journal of Agricultural Science*, 12(3):55–70, 2023.
- [19] Espressif Systems. *ESP32 Technical Reference Manual*, 2016.
- [20] Espressif Systems. *ESP32 Series Datasheet*, 2024.
- [21] NestJS Team. *NestJS Documentation: Introduction and Fundamentals*, 2024.