

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 NÔNG TRẠI THÔNG MINH**

GVHD:      Hoàng Lê Hải Thanh  
                Trần Nguyễn Minh Duy  
Sinh viên:    Lê Hoàng Việt    2252903  
                Chế Minh Đức    2210783  
                Nguyễn Phú Quý    2212871



## Mục lục

Danh sách hình vẽ	3
Danh sách bảng	3
Danh sách phân chia công việc	4
<b>1 Giới thiệu</b>	<b>5</b>
1.1 Bối cảnh và bài toán . . . . .	5
1.2 Tính cấp thiết của đề tài . . . . .	5
1.3 Tình hình nghiên cứu và các giải pháp hiện có . . . . .	5
1.4 Mục tiêu và câu hỏi nghiên cứu . . . . .	5
1.5 Phạm vi và giới hạn của đề tài . . . . .	6
<b>2 Cơ sở lý thuyết</b>	<b>6</b>
2.1 Kiến trúc hệ thống IoT . . . . .	6
2.1.1 Mô hình kiến trúc phân lớp (Layered Architecture) . . . . .	6
2.1.2 Tầng Xử lý nghiệp vụ với NestJS . . . . .	7
2.2 Hệ quản trị Cơ sở dữ liệu quan hệ (PostgreSQL) . . . . .	7
2.2.1 Tuân thủ chuẩn ACID và Độ tin cậy cao . . . . .	7
2.2.2 Hỗ trợ dữ liệu bán cấu trúc (JSONB) . . . . .	7
2.2.3 Khả năng mở rộng (Extensibility) . . . . .	7
2.3 Cơ sở dữ liệu chuỗi thời gian (Time - series Database) . . . . .	7
2.3.1 Kiến trúc Hypertables . . . . .	8
2.3.2 Khả năng nén và quản lý vòng đời dữ liệu . . . . .	8
2.3.3 Tính thống nhất trong hệ sinh thái (SQL Compatibility) . . . . .	8
2.4 Công nghệ phần cứng/vi điều khiển . . . . .	8
2.4.1 Vi điều khiển ESP32 . . . . .	8
2.5 Công nghệ cảm biến/ngoài vi . . . . .	8
2.5.1 Giao thức I2C (Inter-Integrated Circuit) . . . . .	8
2.5.2 Giao thức DVPI (Digital Video Port Interface) . . . . .	9
2.5.3 GPIO (General Purpose Input/Output) . . . . .	9
2.5.4 ADC (Analog-to-Digital Converter) . . . . .	9
2.6 Công nghệ hạ tầng mạng . . . . .	9
2.6.1 Giao thức WiFi (IEEE 802.11) . . . . .	9
2.6.2 Giao thức MQTT . . . . .	9
2.6.3 Giao thức HTTP . . . . .	10
2.7 Công nghệ phần mềm/Firmware . . . . .	10
2.7.1 Hệ điều hành FreeRTOS . . . . .	10
2.7.2 Định dạng dữ liệu JSON . . . . .	10
2.7.3 Giao thức UART . . . . .	10
2.8 Thuật toán RFE (Robust Feature Extractor) . . . . .	10
<b>3 Khảo sát các giải pháp đã có</b>	<b>12</b>
3.1 Khảo sát các ứng dụng, sản phẩm hoặc hệ thống liên quan . . . . .	12
3.2 Khảo sát các mô hình, thuật toán, phương pháp . . . . .	12
3.3 Bảng tổng hợp và phân tích khoảng trống . . . . .	12
<b>4 Phân tích yêu cầu</b>	<b>13</b>
4.1 Mục đích . . . . .	13
4.2 Phương pháp thu thập yêu cầu . . . . .	13
4.3 Kết quả thu thập yêu cầu . . . . .	13
4.4 Yêu cầu chức năng . . . . .	13
4.5 Yêu cầu phi chức năng . . . . .	14
4.5.1 Use case: Monitor Farm . . . . .	14
4.6 Các mô hình phân tích UML . . . . .	20
4.6.1 Use case: Monitor System Analytics . . . . .	20



4.6.2 Use case: Troubleshoot System . . . . .	22
4.6.3 Use case: Manage Reports . . . . .	24
<b>5 Giải pháp đề xuất . . . . .</b>	<b>26</b>
5.1 Mục tiêu của giải pháp . . . . .	26
5.2 Thiết kế kiến trúc (Mức khái niệm) . . . . .	26
5.2.1 Thiết kế Cơ sở dữ liệu . . . . .	26
5.3 Thiết kế Kiến trúc Hệ thống . . . . .	27
5.4 Đề xuất hiện thực và lựa chọn công nghệ . . . . .	28
5.4.1 Quản lý lược đồ dữ liệu cảm biến từ xa . . . . .	28
5.4.2 Đề xuất cải tiến cho thuật toán RFE . . . . .	30
5.5 Thiết kế các thành phần chính . . . . .	32
5.5.1 Quy trình Giám sát Dashboard (Analytics Flow): . . . . .	32
5.5.2 Quy trình Xử lý sự cố (Troubleshoot Flow): . . . . .	33
5.5.3 Firmware . . . . .	34
5.5.3.a Activity Diagram . . . . .	34
5.5.3.b Communication Diagram . . . . .	35
5.5.3.c Component Diagram . . . . .	35
5.5.3.d Sequence Diagram . . . . .	36
5.5.3.e Timing Diagram . . . . .	37
5.5.3.f State Machine Diagram . . . . .	38
5.5.3.g Deployment Diagram . . . . .	39
5.5.6 Cách đánh giá giải pháp . . . . .	39
<b>6 Kế hoạch thực hiện . . . . .</b>	<b>40</b>
6.1 Mục tiêu theo từng giai đoạn . . . . .	40
6.1.0.a Firmware . . . . .	40
6.1.0.b Software . . . . .	40
6.2 Lịch trình và mốc thời gian . . . . .	41
6.2.0.a Firmware . . . . .	41
6.2.0.b Software . . . . .	42
6.3 Rủi ro và phương án giảm thiểu . . . . .	43
6.3.0.a Firmware . . . . .	43
6.3.0.b Software . . . . .	43
<b>7 Kết luận . . . . .</b>	<b>44</b>
<b>8 Phụ lục . . . . .</b>	<b>45</b>
<b>Tài liệu tham khảo . . . . .</b>	<b>46</b>



## Danh sách hình vẽ

1	Sơ đồ quản lý farm . . . . .	14
2	Sơ đồ luồng quản lý và tạo báo cáo . . . . .	20
4	Sơ đồ luồng quản lý và tạo báo cáo . . . . .	24
5	Sơ đồ Quan hệ Thực thể (ERD) của toàn bộ hệ thống . . . . .	26
6	Sơ đồ Kiến trúc Phân lớp của Hệ thống . . . . .	27
7	Activity Diagram mô tả luồng giám sát và tương tác trên Dashboard . . . . .	32
8	Activity Diagram quy trình chẩn đoán và khắc phục sự cố hệ thống . . . . .	33
9	Activity Diagram . . . . .	34
10	Communication Diagram . . . . .	35
11	Component Diagram . . . . .	35
12	Sequence Diagram . . . . .	36
13	Timing Diagram . . . . .	37
14	State Machine Diagram . . . . .	38
15	Deployment Diagram . . . . .	39

## Danh sách bảng

1	Bảng phân chia công việc . . . . .	4
2	Bảng phân bố lịch trình thực hiện đồ án 15 tuần . . . . .	41
3	Lịch trình thực hiện phần mềm (Backend & Web) trong 15 tuần . . . . .	42
4	Bảng phân tích rủi ro và phương án giảm thiểu . . . . .	43
5	Phân tích rủi ro phần mềm và phương án xử lý . . . . .	43



## Danh sách phân chia công việc

STT	Họ tên	MSSV	Vai trò	Distribution
1	Lê Hoàng Việt	2252903	Task1 Task2	100%
2	Chế Minh Đức	2210783	Task1 Task2	100%
3	Nguyễn Phú Quý	2212871	Task1 Task2	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 [1].

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 [2].
- 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 Tính cấp thiết của đề tài

Đề tài trở nên 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 tắt 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 [3]. Tuy nhiên, phần lớn các giải pháp hiện tại vẫn còn rời rạc, 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 [4]. 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 [5].

### 1.3 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 [6].
- 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.

### 1.4 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 hệ thống thông nhât có khả năng tích hợp thiết bị từ nhiều nguồn khác nhau.
- Phát triển module Sensor Fault Detection ứng dụng thuật toán để tự động phát hiện các lỗi thường (mất kết nối, dữ liệu sai lệch, trỗi cảm biến).
- Thiết kế giao diện người dùng (Dashboard) trực quan, thân thiện với đối tượng thương lái/chủ vườn, hỗ trợ giám sát đa nông trại và cảnh báo thời gian thực.

### Câu hỏi nghiên cứu:

- Làm thế nào để xây dựng một cơ chế định danh và quản lý thông nhât 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?

## 1.5 Phạm vi và giới hạn của đề tài

### Phạm vi:

- *Đối tượng nghiên cứu:* Các thiết bị IoT cảm biến môi trường phổ biến và các thiết bị chấp hành cơ bản trong mô hình nhà kính.
- *Nền tảng công nghệ:* Hệ thống quản lý tập trung trên 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 điệp nhẹ (Lightweight messaging protocols).
- *Triển khai:* Thủ nghiêm trên mô hình nông trại mẫu (Pilot testing) với dữ liệu mô phỏng và thực tế.

### 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 nghiêm trên một số loại cảm biến đặc thù, có thể cần điều 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 IoT

#### 2.1.1 Mô hình kiến trúc phân lớp (Layered Architecture)

Hệ thống được thiết kế dựa trên mô hình kiến trúc phân lớp (Layered Architecture) kết hợp với hướng sự kiện (Event-driven). Việc phân chia này giúp tách biệt các mối quan tâm (Separation of Concerns), trong đó việc thu thập dữ liệu từ thiết bị và việc xử lý logic nghiệp vụ được độc lập với nhau [7]. Kiến trúc bao gồm các tầng chính:

- **Physical Layer:** Các thiết bị ESP32 thu thập dữ liệu môi trường.
- **Integration Layer:** Đóng vai trò trung gian tiếp nhận dữ liệu qua RabbitMQ và điều phối API qua Gateway.
- **Business Layer:** Nơi xử lý logic chính, được xây dựng trên nền tảng NestJS.
- **Database Layer:** Lưu trữ dữ liệu quan hệ và chuỗi thời gian.



### 2.1.2 Tầng Xử lý nghiệp vụ với NestJS

Dối với tầng Business Layer, nhóm nghiên cứu lựa chọn **NestJS** - một framework mã nguồn mở dành cho việc xây dựng các ứng dụng phía máy chủ (server - side) hiệu quả và dễ mở rộng trên nền tảng Node.js [8].

Các lý do chính cho việc lựa chọn NestJS trong đề tài này bao gồm:

- Hỗ trợ kiến trúc Microservices và Giao tiếp bất đồng bộ:** Đây là yếu tố quan trọng nhất. Khác với các framework truyền thống, NestJS cung cấp cơ chế tích hợp sẵn (out - of - the - box) với các Message Broker như **RabbitMQ**. Điều này cho phép Backend dễ dàng chuyển đổi sang mô hình hướng sự kiện (Event - driven), giúp hệ thống có thể xử lý hàng nghìn thông điệp từ cảm biến gửi về đồng thời mà không bị tắc nghẽn (Non - blocking I/O) [8].
- Kiến trúc Module hóa (Modularity):** NestJS tổ chức mã nguồn thành các Module riêng biệt (như *AuthModule*, *FarmModule*, *DeviceModule*). Cấu trúc này rất phù hợp với bài toán quản lý đa nông trại, giúp nhóm phát triển dễ dàng bảo trì, mở rộng tính năng cho từng phân hệ mà không ảnh hưởng đến toàn bộ hệ thống.
- Sử dụng TypeScript:** Việc sử dụng TypeScript giúp đảm bảo tính chặt chẽ của dữ liệu (Type safety) [9]. Trong các hệ thống IoT, việc định nghĩa chính xác kiểu dữ liệu (Data Payload) từ cảm biến là rất quan trọng để tránh các lỗi logic tiềm ẩn trong quá trình xử lý.

## 2.2 Hệ quản trị Cơ sở dữ liệu quan hệ (PostgreSQL)

Trước khi áp dụng các công nghệ chuyên biệt cho dữ liệu chuỗi thời gian, hệ thống cần một nền tảng vững chắc để quản lý các dữ liệu có cấu trúc (Structured Data) như thông tin người dùng, danh sách nông trại, và quyền truy cập. Nhóm nghiên cứu lựa chọn **PostgreSQL** vì các đặc tính kỹ thuật vượt trội sau [10]:

### 2.2.1 Tuân thủ chuẩn ACID và Độ tin cậy cao

Dối với các phân hệ quản lý (User & Farm Management), tính toàn vẹn dữ liệu là ưu tiên hàng đầu. PostgreSQL tuân thủ nghiêm ngặt chuẩn ACID (Atomicity, Consistency, Isolation, Durability), đảm bảo các giao dịch quan trọng (như tạo mới nông trại, phân quyền người dùng) luôn được thực hiện chính xác và an toàn, ngay cả khi hệ thống gặp sự cố bất ngờ.

### 2.2.2 Hỗ trợ dữ liệu bán cấu trúc (JSONB)

Trong các hệ thống IoT, cấu hình của thiết bị (Device Configurations) thường đa dạng và thay đổi tùy theo loại cảm biến. PostgreSQL cung cấp kiểu dữ liệu **JSONB**, cho phép lưu trữ các thuộc tính động này một cách linh hoạt mà không cần thay đổi cấu trúc bảng (Schema migration). Điều này giúp hệ thống tận dụng được sự linh hoạt của NoSQL (như MongoDB) ngay bên trong một cơ sở dữ liệu quan hệ mạnh mẽ.

### 2.2.3 Khả năng mở rộng (Extensibility)

Khác với các hệ quản trị CSDL khác, PostgreSQL được thiết kế để dễ dàng mở rộng thông qua các *Extensions*. Đây chính là tiền đề kỹ thuật quan trọng để nhóm nghiên cứu có thể tích hợp **TimescaleDB** (một Extension của PostgreSQL) vào hệ thống, biến PostgreSQL thành một cơ sở dữ liệu lai (Hybrid) mạnh mẽ: vừa xử lý tốt dữ liệu quan hệ, vừa tối ưu cho dữ liệu chuỗi thời gian.

## 2.3 Cơ sở dữ liệu chuỗi thời gian (Time - series Database)

Đặc thù của hệ thống giám sát nông nghiệp là việc thu thập dữ liệu môi trường (nhiệt độ, độ ẩm, ánh sáng) diễn ra liên tục theo thời gian thực. Dữ liệu này có tính chất "chỉ thêm vào" (append - only) và khối lượng tăng trưởng rất nhanh theo thời gian [11]. Các hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) truyền thống thường gặp vấn đề về hiệu năng truy vấn (Query performance) khi bảng dữ liệu đạt kích thước hàng triệu bản ghi.



Để giải quyết vấn đề này, nhóm nghiên cứu lựa chọn **TimescaleDB** - một cơ sở dữ liệu chuỗi thời gian mã nguồn mở được xây dựng dựa trên PostgreSQL. Các ưu điểm kỹ thuật nổi bật bao gồm [12]:

### 2.3.1 Kiến trúc Hypertables

TimescaleDB giới thiệu khái niệm **Hypertables** - một lớp trừu tượng hóa (abstraction layer) giúp tự động phân mảnh dữ liệu (partitioning) theo thời gian và không gian.

- Đối với ứng dụng (NestJS), Hypertable hoạt động giống như một bảng SQL bình thường.
- Tuy nhiên, ở tầng vật lý, dữ liệu được chia nhỏ thành các *Chunks*. Khi thực hiện truy vấn dữ liệu lịch sử (ví dụ: vẽ biểu đồ nhiệt độ trong 7 ngày qua), TimescaleDB chỉ cần quét các Chunks liên quan thay vì quét toàn bộ bảng, giúp tăng tốc độ truy vấn lên hàng trăm lần so với PostgreSQL thuần.

### 2.3.2 Khả năng nén và quản lý vòng đời dữ liệu

Trong IoT, dữ liệu cũ (ví dụ: dữ liệu từ năm ngoái) thường ít giá trị hơn dữ liệu mới. TimescaleDB cung cấp cơ chế:

- **Data Retention Policy:** Tự động xóa dữ liệu quá cũ (ví dụ: sau 1 năm) để giải phóng dung lượng lưu trữ mà không cần chạy các cron job thủ công.
- **Native Compression:** Nén dữ liệu lịch sử giúp giảm tới 90% dung lượng đĩa cứng, tiết kiệm chi phí vận hành cho hệ thống.

### 2.3.3 Tính thống nhất trong hệ sinh thái (SQL Compatibility)

Vì TimescaleDB bản chất là một Extension của PostgreSQL, nhóm nghiên cứu có thể sử dụng cùng một ngôn ngữ truy vấn **SQL** và cùng một driver kết nối cho cả dữ liệu nghiệp vụ (User/Farm - PostgreSQL) và dữ liệu cảm biến (Sensor Data - TimescaleDB). Điều này giúp giảm độ phức tạp khi phát triển và bảo trì hệ thống.

## 2.4 Công nghệ phần cứng/vi điều khiển

### 2.4.1 Vi điều khiển ESP32

ESP32 là dòng vi điều khiển (SoC - System on Chip) hiệu năng cao, giá thành thấp được phát triển bởi Espressif Systems, tích hợp sẵn kết nối Wi-Fi 2.4 GHz và Bluetooth chế độ kép (Dual-mode Bluetooth). Được xây dựng dựa trên kiến trúc vi xử lý Xtensa® Dual-Core 32-bit LX6, ESP32 sở hữu khả năng xử lý vượt trội với xung nhịp có thể đạt tới 240 MHz và hiệu suất tính toán lên đến 600 DMIPS. So với thế hệ tiền nhiệm ESP8266, ESP32 không chỉ vượt trội về tốc độ xử lý mà còn được nâng cấp mạnh mẽ về số lượng chân giao tiếp (GPIO), dung lượng bộ nhớ (520 KB SRAM) và các ngoại vi tích hợp như cảm biến chạm (Touch Sensor), cảm biến Hall và bộ điều khiển CAN. Với sự hỗ trợ mạnh mẽ của hệ điều hành thời gian thực FreeRTOS ngay từ tầng phần cứng, ESP32 trở thành nền tảng lý tưởng cho các ứng dụng IoT phức tạp đòi hỏi khả năng đa nhiệm (Multitasking) và xử lý dữ liệu thời gian thực.[13]

## 2.5 Công nghệ cảm biến/ngoài vi

### 2.5.1 Giao thức I2C (Inter-Integrated Circuit)

I2C (Inter-Integrated Circuit) là một giao thức truyền thông nối tiếp đồng bộ, bán song công (Half-duplex) được phát triển bởi Philips Semiconductor (nay là NXP), đóng vai trò là chuẩn giao tiếp tầm ngắn phổ biến nhất trong các thiết kế vi mạch tích hợp. Đặc trưng của I2C là kiến trúc Multi-Master/Multi-Slave, cho phép kết nối nhiều thiết bị ngoại vi (Cảm biến, EEPROM, RTC) với vi điều khiển chỉ thông qua hai đường dây tín hiệu duy nhất: SDA (Serial Data) để truyền tải dữ liệu hai chiều và SCL (Serial Clock) để đồng bộ hóa xung nhịp. Nhờ cơ chế định địa chỉ mềm (Addressing) thay vì sử dụng các chân chọn chip (Chip Select) vật lý như giao thức SPI, I2C giúp tối thiểu hóa số lượng chân GPIO cần thiết và giảm độ phức tạp của mạch in, trở thành giải pháp lý tưởng cho các ứng dụng đo lường và giám sát trong các hệ thống IoT.[14]



### 2.5.2 Giao thức DVPI (Digital Video Port Interface)

DVPI (Digital Video Port Interface) là một giao diện truyền thông song song (Parallel Interface) tiêu chuẩn được sử dụng rộng rãi để kết nối các cảm biến hình ảnh CMOS (như OV2640) với vi điều khiển hoặc bộ xử lý trung tâm. Khác với các giao thức truyền nối tiếp như SPI hay I2C vốn bị giới hạn về băng thông, DVP sử dụng một bus dữ liệu rộng (thường là 8bit hoặc 10bit) để truyền tải toàn bộ một byte dữ liệu điểm ảnh (pixel) trong mỗi chu kỳ xung nhịp. Cơ chế vận hành của DVP dựa vào sự đồng bộ hóa chính xác giữa ba tín hiệu điều khiển cốt lõi: PCLK (Pixel Clock) xác định tốc độ lấy mẫu từng điểm ảnh, VSYNC (Vertical Sync) báo hiệu thời điểm bắt đầu một khung hình mới, và HREF/H SYNC (Horizontal Reference) xác định dữ liệu hợp lệ của từng dòng quét ngang. Nhờ kiến trúc này, DVP cung cấp băng thông đủ lớn để truyền tải dữ liệu hình ảnh thời gian thực với độ trễ thấp, làm nền tảng cho các ứng dụng thị giác máy tính trên thiết bị nhúng.[15]

### 2.5.3 GPIO (General Purpose Input/Output)

GPIO (General Purpose Input/Output) là các chân tín hiệu số đa năng trên vi điều khiển, đóng vai trò là giao diện vật lý tương tác trực tiếp với các mạch điện tử bên ngoài. Không bị giới hạn bởi một chức năng cố định, trạng thái của chân GPIO có thể được lập trình viên cấu hình linh hoạt ở hai chế độ: Output (Đầu ra) để điều khiển mức logic điện áp (High/Low) nhằm kích hoạt các cơ cấu chấp hành, hoặc Input (Đầu vào) để đọc trạng thái logic (0/1) từ các công tắc hay cảm biến số. Trên nền tảng ESP32, kiến trúc GPIO Matrix cho phép ánh xạ linh hoạt các tín hiệu ngoại vi tới các chân vật lý khác nhau, giúp tối ưu hóa quá trình thiết kế mạch in (PCB) và định tuyến tín hiệu.[16]

### 2.5.4 ADC (Analog-to-Digital Converter)

ADC (Analog-to-Digital Converter) là thành phần ngoại vi thiết yếu giúp vi điều khiển (vốn chỉ hoạt động với các tín hiệu số rời rạc) có thể 'hiểu' được các đại lượng vật lý biến thiên liên tục trong môi trường thực như nhiệt độ, ánh sáng hay độ ẩm. Nguyên lý hoạt động của ADC dựa trên quá trình Lấy mẫu (Sampling) và Lượng tử hóa (Quantization): nó đo điện áp đầu vào tại các khoảng thời gian rời rạc và chuyển đổi giá trị điện áp đó thành một chuỗi số nhị phân tương ứng với độ phân giải của bộ chuyển đổi.[16]

## 2.6 Công nghệ hạ tầng mạng

### 2.6.1 Giao thức WiFi (IEEE 802.11)

WiFi (Wireless Fidelity) là một họ các giao thức mạng không dây cục bộ (WLAN) dựa trên bộ tiêu chuẩn IEEE 802.11, cho phép các thiết bị điện tử trao đổi dữ liệu qua sóng vô tuyến tốc độ cao. Trong các ứng dụng IoT, WiFi đóng vai trò là lớp truyền dẫn (Transport Layer) mạnh mẽ, cung cấp khả năng kết nối trực tiếp vào hạ tầng Internet thông qua các thiết bị định tuyến (Router/Access Point) mà không cần Gateway chuyển đổi giao thức phức tạp. ESP32 hỗ trợ chuẩn 802.11 b/g/n hoạt động trên băng tần 2.4 GHz, cung cấp sự cân bằng tối ưu giữa phạm vi phủ sóng (xuyên vật cản tốt hơn 5GHz) và tốc độ truyền tải dữ liệu, đáp ứng tốt các yêu cầu từ gửi gói tin điều khiển nhỏ đến truyền tải hình ảnh dung lượng lớn.[13]

### 2.6.2 Giao thức MQTT

MQTT (Message Queuing Telemetry Transport) là một giao thức truyền thông điệp theo mô hình Xuất bản - Đăng ký (Publish - Subscribe) cực kỳ nhẹ, hoạt động trên nền tảng TCP/IP. Được thiết kế chuyên biệt cho các môi trường có băng thông thấp, độ trễ cao hoặc mạng không ổn định, MQTT đã trở thành tiêu chuẩn vàng trong các hệ thống Internet vạn vật (IoT). Khác với mô hình Request - Response truyền thống của HTTP, kiến trúc MQTT tách biệt hoàn toàn giữa thiết bị gửi (Publisher) và thiết bị nhận (Subscriber) thông qua một thành phần trung gian gọi là Broker. Broker đóng vai trò như một 'bưu điện số', chịu trách nhiệm lọc, định tuyến và phân phối các gói tin dựa trên các chủ đề (Topics) phân cấp. Nhờ cơ chế đóng gói tiêu đề (header) tối giản chỉ 2 byte và khả năng duy trì kết nối liên tục (Keep-alive), MQTT giúp tối ưu hóa năng lượng tiêu thụ cho các thiết bị nhúng như ESP32, đồng thời đảm bảo độ tin cậy cao trong việc truyền tải dữ liệu cảm biến và lệnh điều khiển thời gian thực.[17]



### 2.6.3 Giao thức HTTP

HTTP (Hypertext Transfer Protocol) là giao thức truyền tải siêu văn bản hoạt động ở Lớp Ứng dụng (Application Layer) của mô hình OSI, đóng vai trò là nền tảng cốt lõi của việc trao đổi dữ liệu trên World Wide Web. HTTP vận hành dựa trên mô hình Client - Server theo cơ chế Request - Response (Yêu cầu - Phản hồi): Client (trong trường hợp này là ESP32) gửi một yêu cầu HTTP đến Server, và Server sẽ trả về một mã trạng thái kèm theo dữ liệu tương ứng. Đặc điểm nổi bật của HTTP là tính 'phi trạng thái' (Stateless), nghĩa là mỗi cặp yêu cầu - phản hồi là độc lập và Server không lưu giữ thông tin về Client giữa các lần kết nối (trừ khi sử dụng các cơ chế bổ sung như Cookies hay Token). Trong các hệ thống IoT hiện đại, HTTP thường được triển khai dưới dạng kiến trúc RESTful API, cho phép các thiết bị nhúng tương tác với cơ sở dữ liệu và dịch vụ đám mây thông qua các phương thức chuẩn hóa như GET, POST, PUT và DELETE.[18]

## 2.7 Công nghệ phần mềm/Firmware

### 2.7.1 Hệ điều hành FreeRTOS

FreeRTOS (Real-Time Operating System) là một nhân hệ điều hành thời gian thực mã nguồn mở, được thiết kế chuyên biệt cho các vi điều khiển nhúng để quản lý tài nguyên phần cứng và lập lịch tác vụ. Khác với các hệ điều hành đa dụng (như Windows hay Linux) tập trung vào trải nghiệm người dùng, mục tiêu tối thượng của FreeRTOS là tính xác định (Determinism) và độ trễ thấp, đảm bảo các tác vụ quan trọng phải được thực thi trong một khoảng thời gian quy định nghiêm ngặt. Trên nền tảng ESP32, FreeRTOS được tích hợp sâu vào bộ công cụ phát triển (ESP-IDF/Arduino Core), cung cấp cơ chế lập lịch ưu tiên (Preemptive Scheduling) cho phép phân tách một chương trình lớn thành các tiểu trình độc lập gọi là Tác vụ (Tasks). Mỗi tác vụ sở hữu ngăn xếp (Stack) và mức độ ưu tiên riêng biệt, giúp hệ thống vận hành đa nhiệm (Multitasking) mượt mà ngay cả trên các thiết bị giới hạn về tài nguyên.[19]

### 2.7.2 Định dạng dữ liệu JSON

JSON (JavaScript Object Notation) là một chuẩn định dạng trao đổi dữ liệu văn bản mở (Open Standard), gọn nhẹ và độc lập với ngôn ngữ lập trình, được sử dụng rộng rãi để lưu trữ và truyền tải dữ liệu có cấu trúc. Mặc dù có nguồn gốc từ cú pháp đối tượng của JavaScript, JSON hiện nay được hỗ trợ bởi hầu hết các ngôn ngữ lập trình hiện đại (C/C++, Python, Java, Dart...). Cấu trúc của JSON được xây dựng dựa trên hai cấu trúc dữ liệu phổ biến: tập hợp các cặp 'tên : giá trị' (Key-Value pairs) tạo thành một Đối tượng (Object), và danh sách các giá trị có thứ tự tạo thành một Mảng (Array). Trong các hệ thống IoT, JSON đóng vai trò là 'ngôn ngữ chung' (Lingua Franca), cho phép Firmware nhúng (viết bằng C++) đóng gói dữ liệu cảm biến thành chuỗi văn bản tiêu chuẩn trước khi truyền qua giao thức MQTT hoặc HTTP tới máy chủ.[20]

### 2.7.3 Giao thức UART

UART (Universal Asynchronous Receiver-Transmitter) là giao thức truyền thông nối tiếp không đồng bộ, đóng vai trò là chuẩn giao tiếp cơ bản nhất giữa máy tính và các hệ thống nhúng. Khác với I2C hay SPI vốn hoạt động dựa trên tín hiệu xung nhịp (Clock) đồng bộ, UART truyền tải dữ liệu theo cơ chế bất đồng bộ (Asynchronous). Điều này nghĩa là bên gửi (Transmitter - TX) và bên nhận (Receiver - RX) không chia sẻ chung một đường dây Clock, mà thay vào đó, chúng phải thống nhất trước với nhau về tốc độ truyền tải, gọi là Baud Rate (tốc độ Baud). Dữ liệu được đóng gói thành các khung (Frame) bao gồm bit bắt đầu (Start bit), các bit dữ liệu (Data bits), bit kiểm tra chẵn lẻ tùy chọn (Parity bit) và bit kết thúc (Stop bit). Trên ESP32, UART không chỉ là giao diện lập trình nạp firmware mà còn là kênh giao tiếp quan trọng để xuất nhật ký hoạt động (System Logs) phục vụ quá trình giám sát và gỡ lỗi.[16]

## 2.8 Thuật toán RFE (Robust Feature Extractor)

Trong kỷ nguyên Công nghiệp 4.0, Internet vạn vật (IoT) đóng vai trò trụ cột với các ứng dụng trải rộng từ sản xuất thông minh đến các hệ thống bay không người lái nhằm thu thập dữ liệu thời gian thực. Tuy nhiên, do thường xuyên phải vận hành trong các môi trường khắc nghiệt chịu ảnh hưởng bởi nhiệt độ, độ ẩm hay nhiễu điện từ, dữ liệu cảm biến dễ gặp phải các sai lệch nghiêm trọng như giá trị bất thường, trôi tín hiệu hoặc bị kẹt giá trị. Các giải pháp hiện hữu vẫn tồn tại nhiều rào cản đáng kể: trong khi học



máy cỗ điển gặp khó khăn với các mẫu lỗi phức tạp và phương pháp dựa trên tương quan tỏ ra kém hiệu quả trong môi trường động, thì học sâu (Deep Learning) – dù mạnh mẽ – lại đòi hỏi chi phí huấn luyện lớn, thiếu tính minh bạch (hộp đen) và dễ bỏ sót các lỗi hiếm gặp. Bối cảnh này đặt ra yêu cầu cần thiết về một thuật toán phát hiện lỗi mới có khả năng lấp đầy 'khoảng trống' hiện tại: vừa đảm bảo hiệu suất cao và khả năng diễn giải, vừa tối ưu hóa tài nguyên tính toán để phù hợp với đặc thù năng lượng thấp của các thiết bị IoT.[21]

Thay vì tập trung phát triển các mô hình phân loại phức tạp, nghiên cứu đề xuất phương pháp luận Robust Feature Extractor (RFE), một cách tiếp cận ưu tiên việc xử lý dữ liệu đầu vào thông minh nhằm biến đổi chuỗi thời gian thành không gian đặc trưng đa chiều giàu thông tin. RFE tích hợp các nhóm đặc trưng chiến lược bao gồm: tín hiệu gốc và tốc độ thay đổi để nắm bắt động lực học tức thời; bộ nhớ tam thời (temporal memory) thông qua các giá trị trễ để cung cấp ngữ cảnh ngắn hạn; và xu hướng được làm mịn bằng phương pháp EWMA. Điểm đột phá của RFE nằm ở việc áp dụng thống kê trượt trên tốc độ thay đổi nhằm định lượng mức độ biến động (volatility) của tín hiệu, kết hợp với thống kê cục bộ đa quy mô để phân tích hành vi dữ liệu trên nhiều cửa sổ trượt khác nhau. Đặc biệt, thiết kế của RFE chủ động loại bỏ các đặc trưng dựa trên tương quan, một chiến lược then chốt giúp giảm thiểu chi phí tính toán và ngăn chặn hiện tượng quá khớp (overfitting), đảm bảo tính hiệu quả khi triển khai trên các thiết bị IoT.

Quá trình thực nghiệm được tiến hành trên bộ dữ liệu SeDa thu thập từ cảm biến DHT11 nhằm đánh giá toàn diện hiệu năng của phương pháp Robust Feature Extractor (RFE) thông qua việc đối sánh với ba kỹ thuật cơ sở là Autoencoder (AE), Stationary Wavelet Transform (SWT) và TsAssure trên 7 mô hình học máy tiêu chuẩn (như Random Forest, SVM, XGBoost...). Kết quả định lượng cho thấy bộ đặc trưng RFE mang lại sự vượt trội tuyệt đối về mọi chỉ số hiệu năng, duy trì độ chính xác ổn định ở mức cao từ 90,7% đến 92,9%, áp đảo hoàn toàn so với sự thiếu ổn định của các phương pháp đối chứng như TsAssure (chỉ đạt 85 - 88%) hay Autoencoder (79 - 82%). Không chỉ tối ưu về độ chính xác, RFE còn chứng minh ưu thế vượt bậc về hiệu quả tính toán với thời gian trích xuất đặc trưng chỉ 2,05 giây — nhanh gấp 5 lần so với Autoencoder (10,97 giây) và tiệm cận tốc độ của TsAssure—qua đó khẳng định tính khả thi cao khi triển khai cho các ứng dụng phát hiện lỗi thời gian thực trên các thiết bị IoT giới hạn tài nguyên.

Nghiên cứu nhấn mạnh vai trò tiên quyết của kỹ thuật trích xuất đặc trưng (feature engineering), khẳng định rằng việc khai thác sâu động lực thời gian và biến động cục bộ mang lại hiệu suất vượt trội so với các mô hình 'hộp đen' phức tạp như Autoencoder. Phương pháp RFE được xác định là giải pháp tối ưu đạt được sự cân bằng lý tưởng giữa độ chính xác cao và chi phí tính toán thấp, chứng minh tính khả thi vượt trội khi triển khai trên các hệ thống IoT giới hạn tài nguyên năng lượng. Trên cơ sở đó, các hướng phát triển tương lai được đề xuất tập trung vào việc tinh giản số lượng đặc trưng nhằm giảm thiểu độ trễ dự đoán, đồng thời mở rộng phạm vi kiểm thử phương pháp trên đa dạng các loại cảm biến và kịch bản lỗi thực tế khác nhau.



### 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

Dựa trên phân tích hiện trạng và mô hình nghiệp vụ, các yêu cầu chức năng của hệ thống được chia thành các nhóm sau:

#### Nhóm 1: Tổng quan (Dashboard Overview)

**FR-03** 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** 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** 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** Hệ thống phải cung cấp các nút **Thao tác nhanh** (Quick Actions) cho phép Admin tạo nhanh Người dùng mới, Nông trại mới hoặc Thêm thiết bị mới chỉ với 1 thao tác.

#### Nhóm 2: Quản lý Nông trại và Người dùng

**FR-07 (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 (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 (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.

#### Nhóm 3: Quản lý và Giám sát Thiết bị

**FR-01 (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 (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 (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 (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 (Điều 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.



## 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 nồng 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.

#### 4.5.1 Use case: Monitor Farm

**Hình 1: 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"><li>Admin truy cập màn hình Farm.</li><li>Hệ thống kiểm tra quyền xem của Admin.</li><li>Admin chọn nút “Add Farm”.</li><li>Hệ thống hiển thị Form để nhập thông tin.</li><li>Admin điền thông tin cho farm rồi ấn nút “Save”</li><li>Hệ thống hiển thị cập nhật Farm mới vào danh sách</li></ol>
Quy tắc nghiệp vụ	<ul style="list-style-type: none"><li><b>BR-01 (Authorization):</b> Chỉ tài khoản có vai trò <b>System Admin</b> mới được phép tạo Farm.</li><li><b>BR-02 (Required Fields):</b> Các thông tin bắt buộc của Farm (ví dụ: Tên Farm, Vị trí) không được để trống.</li><li><b>BR-03 (Unique Farm Name):</b> Tên Farm phải là duy nhất và không được trùng lặp với các Farm đã tồn tại trong hệ thống.</li><li><b>BR-04 (Data Validation):</b> Dữ liệu nhập vào phải đúng định dạng và nằm trong phạm vi cho phép theo quy định của hệ thống.</li></ul>
Luồng thay thế / Mở rộng	<ul style="list-style-type: none"><li><b>E-01 (Cancel Create Farm):</b> 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.</li></ul>

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"><li>Admin truy cập menu Farm.</li><li>Hệ thống kiểm tra quyền truy cập của Admin.</li><li>Hệ thống truy vấn dữ liệu Farm từ cơ sở dữ liệu.</li><li>Hệ thống hiển thị danh sách Farm.</li></ol>
Quy tắc nghiệp vụ	<ul style="list-style-type: none"><li><b>BR-01 (Authorization):</b> Chỉ tài khoản có vai trò <b>System Admin</b> mới được phép xem danh sách Farm.</li><li><b>BR-02 (Pagination):</b> Danh sách Farm được phân trang để đảm bảo hiệu năng hiển thị.</li></ul>



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"><li>Admin chọn Farm cần chỉnh sửa.</li><li>Hệ thống hiển thị form chỉnh sửa.</li><li>Admin cập nhật thông tin.</li><li>Admin nhấn “Save”.</li><li>Hệ thống lưu và cập nhật danh sách Farm.</li></ol>
Quy tắc nghiệp vụ	<ul style="list-style-type: none"><li><b>BR-01 (Authorization):</b> Chỉ System Admin được phép chỉnh sửa Farm.</li><li><b>BR-02 (Validation):</b> Dữ liệu chỉnh sửa phải hợp lệ.</li></ul>

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"><li>Admin chọn Farm cần xóa.</li><li>Hệ thống hiển thị hộp thoại xác nhận.</li><li>Admin xác nhận xóa.</li><li>Hệ thống xóa Farm và cập nhật danh sách.</li></ol>
Quy tắc nghiệp vụ	<ul style="list-style-type: none"><li><b>BR-01 (Confirmation):</b> Phải xác nhận trước khi xóa.</li></ul>

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"><li>Admin chọn Import hoặc Export.</li><li>Hệ thống hiển thị tùy chọn file.</li><li>Admin xác nhận thao tác.</li><li>Hệ thống xử lý dữ liệu.</li></ol>
Quy tắc nghiệp vụ	<ul style="list-style-type: none"><li><b>BR-01 (File Format):</b> Chỉ hỗ trợ định dạng cho phép (CSV, Excel).</li></ul>



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"><li>• Admin đã đăng nhập</li><li>• Danh sách Farm đang được hiển thị</li></ul>
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"><li>1. Admin xem danh sách Farm.</li><li>2. Admin nhấn chọn một Farm bất kỳ.</li><li>3. Hệ thống kiểm tra quyền truy cập của Admin.</li><li>4. Hệ thống truy vấn thông tin chi tiết của Farm.</li><li>5. Hệ thống hiển thị màn hình chi tiết Farm.</li></ol>
Quy tắc nghiệp vụ	<ul style="list-style-type: none"><li>• <b>BR-01 (Authorization):</b> Chỉ tài khoản có vai trò <b>System Admin</b> mới được phép xem chi tiết Farm.</li><li>• <b>BR-02 (Data Integrity):</b> Farm phải tồn tại trong hệ thống tại thời điểm truy vấn.</li></ul>
Luồng thay thế / Mở rộng	<ul style="list-style-type: none"><li>• <b>E-01 (Back to List):</b> 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.</li></ul>



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"><li>Admin truy cập module Device.</li><li>Hệ thống kiểm tra quyền truy cập.</li><li>Hệ thống hiển thị danh sách thiết bị.</li></ol>
Quy tắc nghiệp vụ	<ul style="list-style-type: none"><li><b>BR-01 (Authorization):</b> Chỉ System Admin được phép xem danh sách thiết bị.</li></ul>

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"><li>Admin nhấn chọn một thiết bị.</li><li>Hệ thống tải dữ liệu chi tiết thiết bị.</li><li>Hệ thống hiển thị màn hình chi tiết thiết bị.</li></ol>
Luồng thay thế / Mở rộng	<ul style="list-style-type: none"><li><b>E-01 (Back):</b> Admin nhấn nút “Back” → hệ thống quay lại danh sách thiết bị.</li></ul>

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"><li>Admin mở màn hình Device.</li><li>Admin chọn “Add Device”.</li><li>Hệ thống hiển thị form nhập thông tin.</li><li>Admin nhập thông tin và nhấn “Save”.</li><li>Hệ thống lưu thiết bị và cập nhật danh sách.</li></ol>
Quy tắc nghiệp vụ	<ul style="list-style-type: none"><li><b>BR-01:</b> Các trường bắt buộc không được để trống.</li></ul>



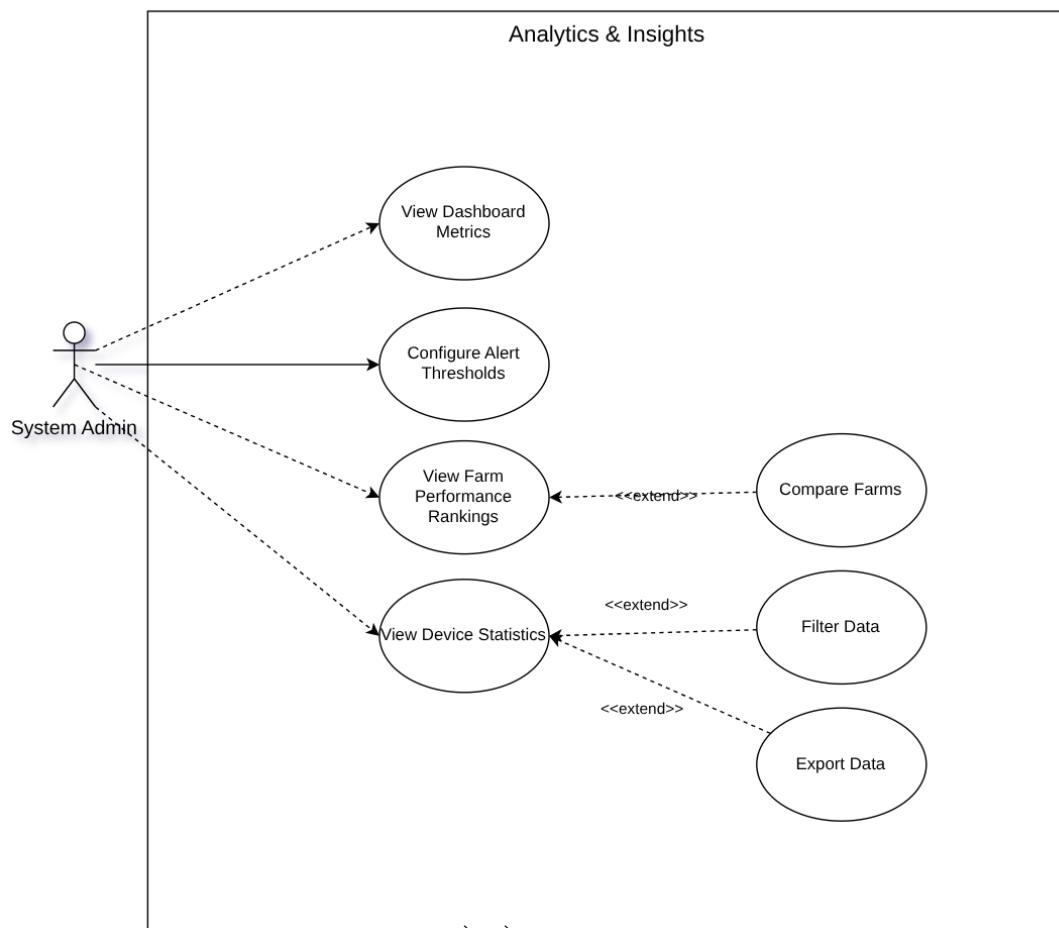
Mã số usecase	UC-04: Edit device
Tên usecase	Chỉnh sửa thiết bị
Mô tả	Admin cập nhật thông tin thiết bị
Actor	System Admin
Tiền điều kiện	Thiết bị đã tồn tại
Hậu điều kiện	Thông tin thiết bị được cập nhật
Trigger	Admin chọn “Edit” tại thiết bị
Luồng chính	<ol style="list-style-type: none"><li>Admin chọn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

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"><li>Admin chọn thiết bị cần xóa.</li><li>Hệ thống hiển thị hộp thoại xác nhận.</li><li>Admin xác nhận xóa.</li><li>Hệ thống xóa thiết bị.</li></ol>
Quy tắc nghiệp vụ	<ul style="list-style-type: none"><li><b>BR-01 (Authorization):</b> Chỉ tài khoản có vai trò <b>System Admin</b> mới được phép xóa thiết bị.</li><li><b>BR-02 (Data Integrity):</b> Thiết bị phải tồn tại trong hệ thống tại thời điểm xóa.</li></ul>
Luồng thay thế / Mở rộng	<ul style="list-style-type: none"><li><b>E-01 (Cancel Delete):</b> Tại bước xác nhận, Admin chọn “Cancel” → Hệ thống hủy thao tác xóa thiết bị.</li><li><b>E-02 (Delete Not Allowed):</b> Nếu thiết bị còn sensor liên kết → Hệ thống hiển thị thông báo không thể xóa thiết bị.</li></ul>

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"> <li>1. Admin chọn chức năng Import hoặc Export.</li> <li>2. Hệ thống xử lý dữ liệu.</li> <li>3. Hệ thống thông báo kết quả.</li> </ol>

## 4.6 Các mô hình phân tích UML

### 4.6.1 Use case: Monitor System Analytics

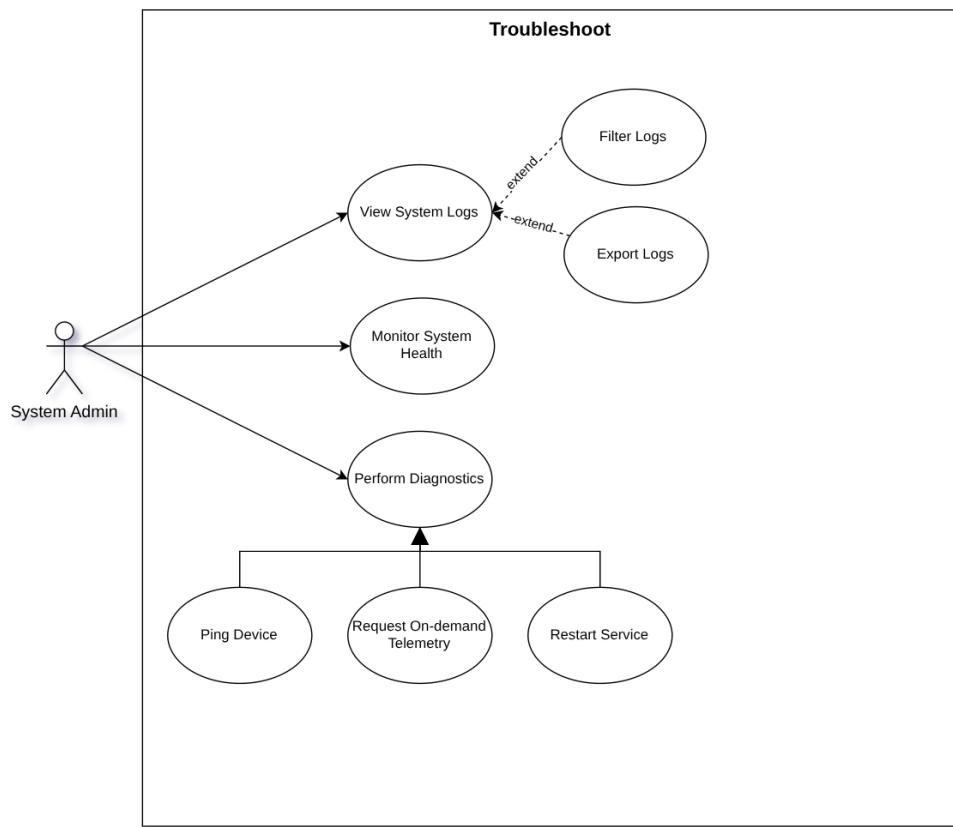


**Hình 2:** Sơ đồ luồng quản lý và tạo báo cáo



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

#### 4.6.2 Use case: Troubleshoot System



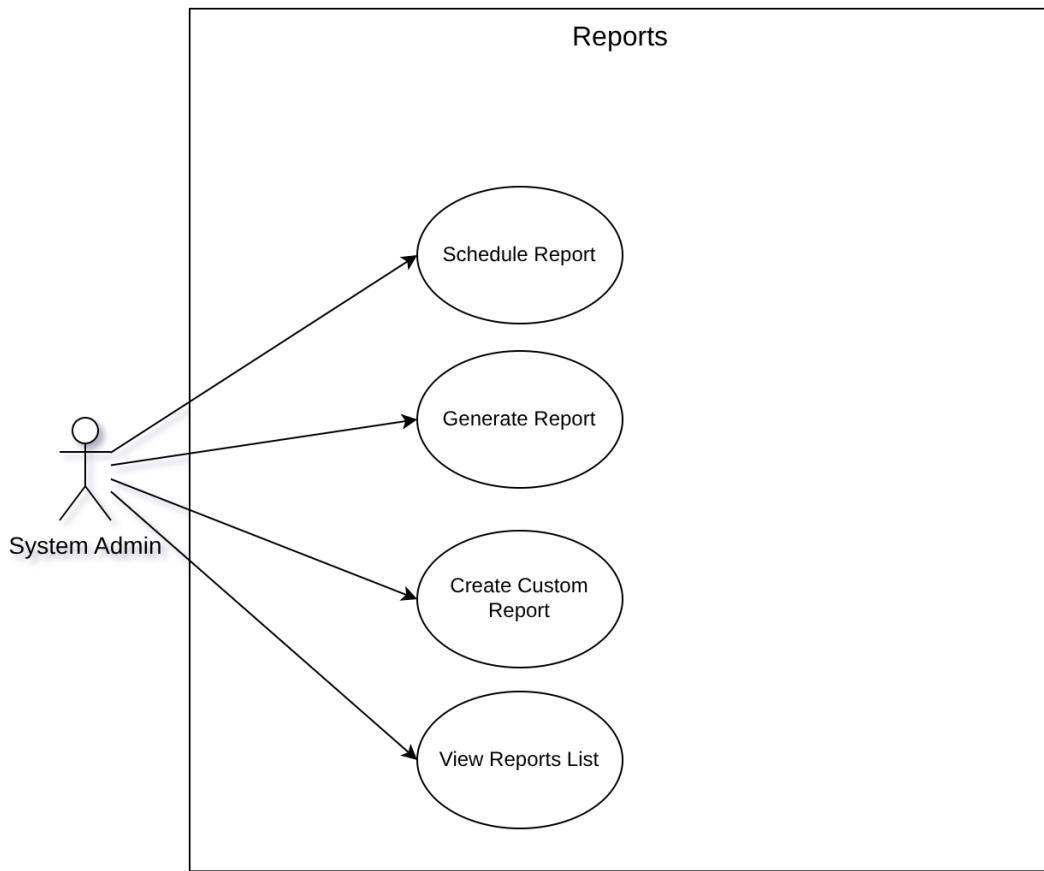
(a) Sơ đồ xử lý sự cố và theo dõi tình trạng hệ thống

Hình mô tả lần lượt quá trình kiểm tra dịch vụ, khắc phục từ xa, và cách truy xuất log, xem lịch sử cảnh báo để đối soát.



Mã số usecase	UC-02: Troubleshoot System
Tên usecase	Kiểm tra và xử lý sự cố
Mô tả	Quy trình xem tình trạng sức khỏe hệ thống, đọc log chi tiết và gửi lệnh khắc phục sự cố từ xa.
Actor	System Admin
Tiền điều kiện	Hệ thống phát hiện sự cố (Alert) hoặc Admin thực hiện kiểm tra định kỳ.
Hậu điều kiện	Nguyên nhân lỗi được xác định hoặc lệnh sửa chữa đã được gửi đi.
Trigger	Admin nhận được thông báo lỗi hoặc truy cập menu “Troubleshoot”.
Luồng chính	<ol style="list-style-type: none"><li>Admin truy cập trang Troubleshoot.</li><li>Hệ thống hiển thị đèn trạng thái (Health Indicators) của Server, API và Mạng.</li><li>Hệ thống truy xuất và hiển thị danh sách 50 dòng System Logs gần nhất.</li><li>Admin phân tích các chỉ số màu đỏ (Lỗi) và nội dung log để xác định nguyên nhân.</li></ol>
Quy tắc nghiệp vụ	<ul style="list-style-type: none"><li><b>BR-05 (Health Coding):</b> Trạng thái hệ thống phải được mã hóa màu: Xanh (Ôn định), Vàng (Cảnh báo tải &gt;80%), Đỏ (Lỗi/Ngừng hoạt động).</li><li><b>BR-06 (Log Privacy):</b> Các thông tin nhạy cảm trong log (như Password, API Key) phải được che dấu (masking) bằng ký tự ‘*****’, trước khi hiển thị.</li><li><b>BR-07 (Action Permission):</b> Chỉ tài khoản Super Admin mới có quyền thực thi các lệnh tác động hệ thống như “Restart Service”.</li></ul>
Luồng thay thế / Mở rộng	<ul style="list-style-type: none"><li><b>E-01 (Perform Diagnostics):</b> Admin chọn thiết bị lỗi → chọn hành động (Ping/Restart Service/Telemetry) → Hệ thống thực thi lệnh và trả về kết quả (Success/Timeout).</li><li><b>E-02 (Export Logs):</b> Admin bấm “Export Logs” → Hệ thống tổng hợp log lỗi thành file CSV/JSON và kích hoạt tải xuống.</li><li><b>E-03 (Filter Logs):</b> Admin lọc theo mức độ “Error” → Hệ thống ẩn các log thông thường, chỉ hiện log lỗi.</li></ul>

#### 4.6.3 Use case: Manage Reports



**Hình 4:** 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"><li>Admin mở trang Reports.</li><li>Hệ thống truy vấn cơ sở dữ liệu báo cáo.</li><li>Hệ thống hiển thị danh sách Recent Reports (bao gồm: Tên, Ngày tạo, Người tạo, Loại báo cáo).</li><li>Admin xem thông tin hoặc tải về các báo cáo cũ.</li></ol>
Quy tắc nghiệp vụ	<ul style="list-style-type: none"><li><b>BR-08 (Retention Policy):</b> 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).</li><li><b>BR-09 (File Format):</b> 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).</li><li><b>BR-10 (Schedule Limit):</b> Mỗi tài khoản Admin chỉ được thiết lập tối đa 5 lịch báo cáo tự động để đảm bảo hiệu năng hệ thống.</li></ul>
Luồng thay thế / Mở rộng	<ul style="list-style-type: none"><li><b>E-01 (Generate):</b> Admin bấm “Generate New” → Hệ thống thu thập dữ liệu hiện tại và tạo file báo cáo ngay lập tức.</li><li><b>E-02 (Schedule):</b> Admin bấm “Schedule” → Hệ thống hiển thị form chọn tần suất (hàng ngày/tuần) → Lưu lịch chạy tự động (Cron job).</li><li><b>E-03 (Custom Report):</b> Admin chọn các trường dữ liệu tùy chỉnh → Bấm “Create” → Hệ thống tạo báo cáo theo mẫu riêng.</li></ul>

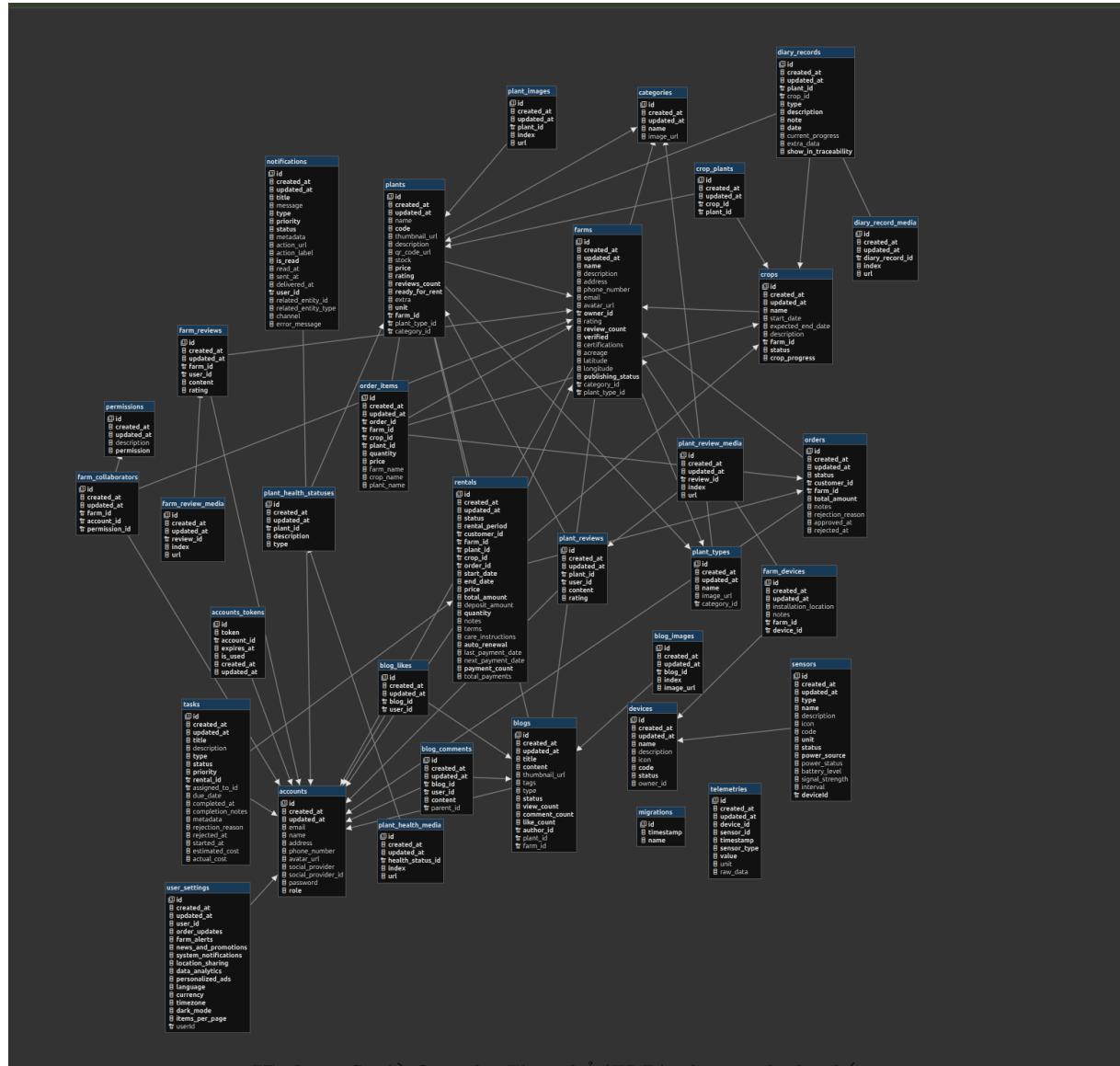
## 5 Giải pháp đề xuất

### 5.1 Mục tiêu của giải pháp

### 5.2 Thiết kế kiến trúc (Mức khái niệm)

#### 5.2.1 Thiết kế Cơ sở dữ liệu

Mô hình Quan hệ Thực thể (ERD) dưới đây minh họa cấu trúc lưu trữ dữ liệu của hệ thống, được thiết kế để đảm bảo tính toàn vẹn dữ liệu và khả năng mở rộng cho các nghiệp vụ giám sát nông nghiệp thông minh.



Hình 5: Sơ đồ Quan hệ Thực thể (ERD) của toàn bộ hệ thống

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

**5.2.1.1 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 xuống giống đến khi thu hoạch, bao gồm cả việc theo dõi sức khỏe cây trồng qua bảng **plant\_health\_statuses**.

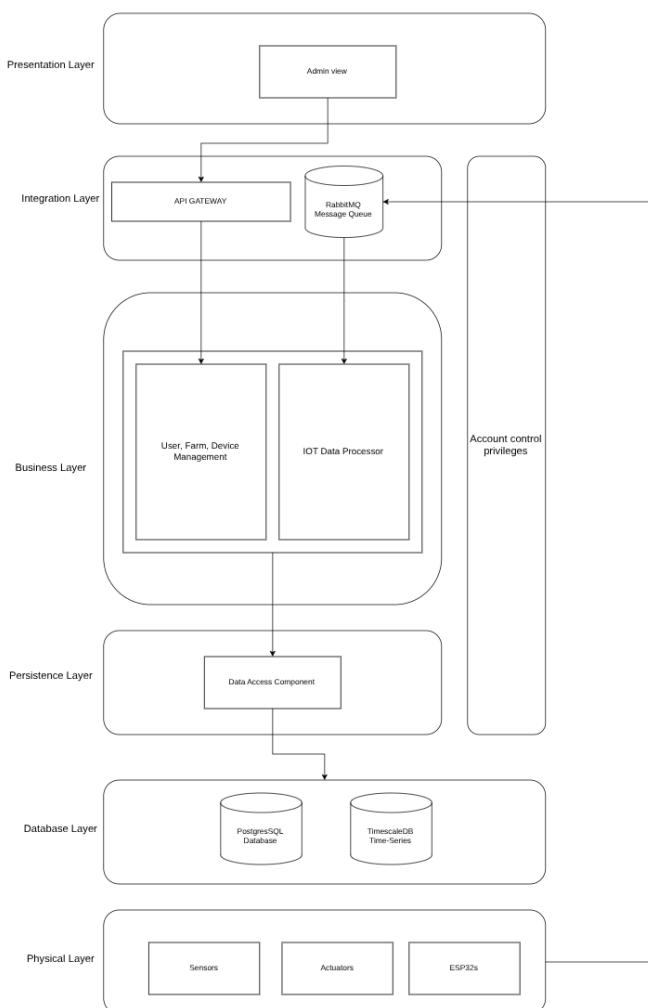
**5.2.1.2 Phân hệ IoT và Thu thập dữ liệu:** Các thiết bị phần cứng được quản lý qua bảng **devices** và **sensors**. Dữ liệu quan trọng nhất của hệ thống là **telemtries**, nơi lưu trữ hàng triệu bản ghi dữ liệu cảm biến (nhiệt độ, độ ẩm, ánh sáng) được gửi về liên tục. Quan hệ giữa **devices** và **farms**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

**5.2.1.3 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 RBAC (Role-Based Access Control), 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.

### 5.3 Thiết kế Kiến trúc Hệ thống

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 6: Sơ đồ Kiến trúc Phân lớp của Hệ thống

Hệ thống được tổ chức thành 5 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 xuống 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 truffle 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 (Telemtries) 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.

**5.3.0.1 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.

## 5.4 Đề xuất hiện thực và lựa chọn công nghệ

### 5.4.1 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.json”, “env\_data\_v1.1.json”, “camera\_event\_v1.0.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.

#### 5.4.2 Đề xuất cải tiến cho thuật toán RFE

### • Cốt lõi thuật toán RFE:

- Phương pháp RFE trong bài báo chủ yếu tập trung vào dữ liệu của một cảm biến đơn lẻ và tránh đặc trưng dựa trên tương quan để giảm chi phí tính toán.
- Tuy nhiên, trong một nhà kính, các cảm biến không hoạt động độc lập. Nhiệt độ ở mọi điểm phải tương quan với nhau.

### • Đề xuất cải tiến: Đặc trưng tương quan không gian.

- Nếu một cảm biến báo nhiệt độ 50 độ C trong khi 10 cảm biến xung quanh nó báo 25 độ C thì cảm biến đó chắc chắn bị lỗi.
- Ý tưởng:  $\Delta \text{valueER} = \text{value\_A} - \text{avr}(\text{all\_other\_sensors\_value})$  (So một cảm biến với giá trị trung bình của tất cả các cảm biến).
- Khó khăn: khó để phát hiện ngưỡng nào quyết định cảm biến bị lỗi hay cảm biến bình thường

### • Khả năng tích hợp vào dự án:

- RFE nguyên bản:
  - \* Ưu điểm:
    - Hiệu quả tính toán rất cao: Thuật toán được thiết kế cho các hệ thống IoT năng lượng thấp. Các phép toán rất nhanh và nhẹ. Hoàn toàn phù hợp để chạy trên một gateway tại nhà kính mà không cần phần cứng mạnh.
    - Phát hiện lỗi cực bộ tốt: RFE rất phù hợp trong việc phát hiện các lỗi truyền thống của một cảm biến khi nó hoạt động sai so với chính nó như:
      1. Giá trị bị kẹt (Stuck): Cảm biến luôn báo 25 độ C. RFE sẽ phát hiện ra vì đặc trưng tốc độ thay đổi và độ biến động sẽ bằng 0.
      2. Lỗi đột biến (Spike): Giá trị nhảy vọt bất thường. RFE sẽ phát hiện qua tốc độ thay đổi.
      3. Lỗi trôi (Drift): Giá trị từ từ sai lệch. RFE sẽ phát hiện qua các đặc trưng "xu hướng" (EWMA).



\* Nhược điểm:

- RFE nguyên bản chủ động tránh các đặc trưng dựa trên tương quan để giảm chi phí. Do đó, nó không thể trả lời câu hỏi: "Cảm biến này đang báo giá trị có hợp lý so với các cảm biến xung quanh nó không?".
- Nếu một cảm biến nhiệt độ bị lỗi và báo giá trị  $20^{\circ}\text{C}$  (một giá trị hợp lệ) một cách ổn định, trong khi cả nhà kính đang là  $35^{\circ}\text{C}$ , RFE nguyên bản sẽ không phát hiện được lỗi này. Nó chỉ thấy một tín hiệu ổn định và cho là "bình thường".

\* Kết luận:

- Khả năng tích hợp cao.
- Hiệu quả về tài nguyên.
- Giải quyết phần lớn các lỗi phần cứng cơ bản của từng cảm biến riêng lẻ.

- RFE cải tiến với đặc trưng tương quan không gian:

\* Ưu điểm:

- Khắc phục điểm yếu lớn nhất: Bằng cách thêm đặc trưng "Tương quan không gian" ( $\Delta = \text{Giá trị cảm biến} - \text{Trung bình của mạng lưới}$ ), hệ thống giờ đây đã có "nhận thức về bối cảnh". Nó có thể phát hiện "lỗi logic" (cảm biến báo  $20^{\circ}\text{C}$  trong khi cả nhà kính là  $35^{\circ}\text{C}$ ).
- Tăng độ chính xác: Việc bổ sung bối cảnh không gian cung cấp cho mô hình một bức tranh hoàn chỉnh hơn. Điều này sẽ làm tăng đáng kể độ chính xác phát hiện lỗi.
- Phù hợp với nông nghiệp: Môi trường nhà kính có tính tương quan cao. Phiên bản cải tiến khai thác được cả hai yếu tố này.

\* Nhược điểm:

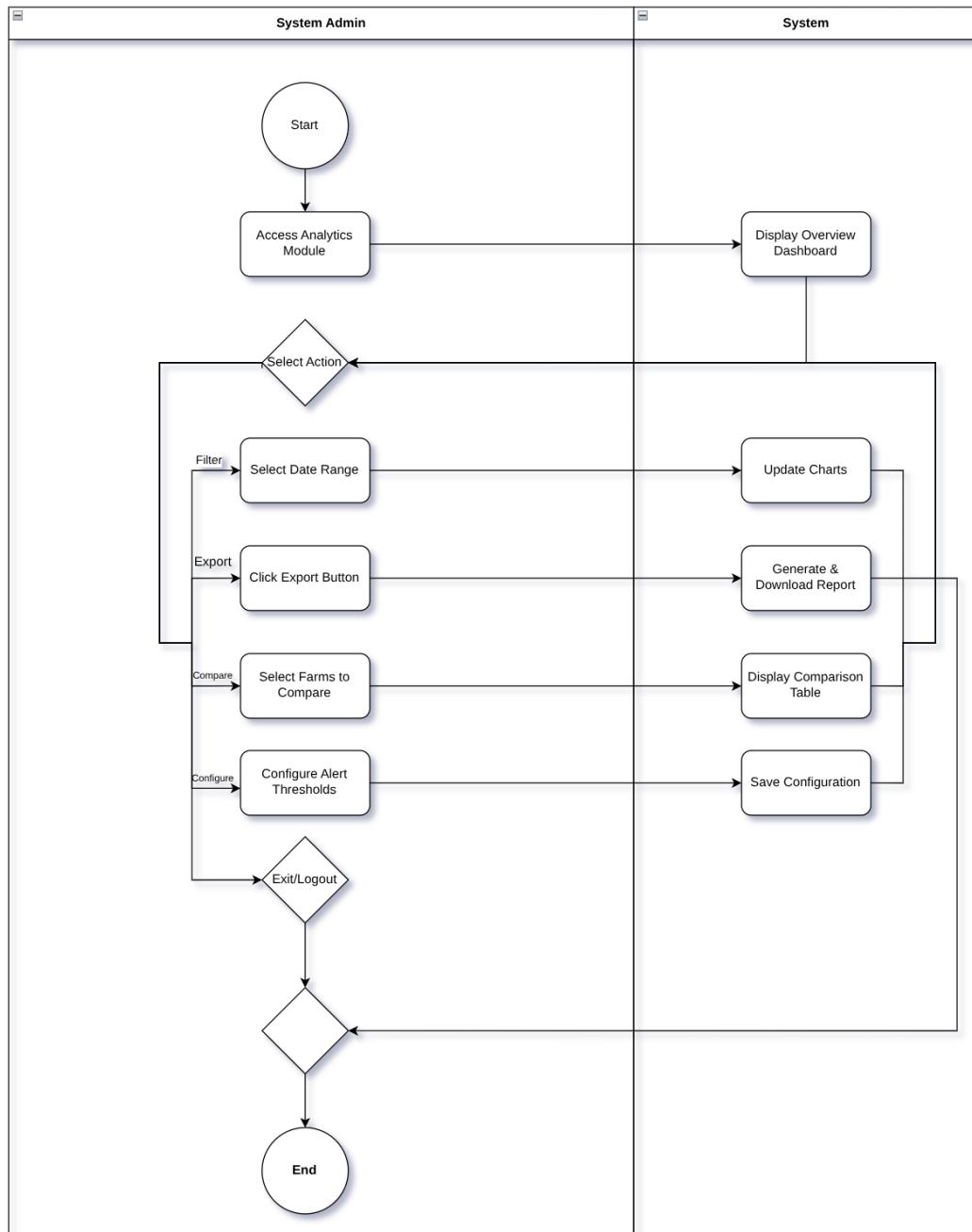
- Tăng chi phí tính toán: Để tính đặc trưng "tương quan không gian", hệ thống phải thu thập dữ liệu từ nhiều cảm biến rồi mới thực hiện phép tính. Điều này nặng hơn một chút so với RFE gốc.
- Độ phức tạp của luồng dữ liệu tăng lên: Cần một bước gom dữ liệu trước khi chạy kỹ thuật đặc trưng, thay vì xử lý song song từng cảm biến.

\* Kết luận:

- Khả năng tích hợp cao.
- Tăng nhẹ về chi phí tính toán.
- Độ chính xác và khả năng phát hiện lỗi logic vượt trội.

## 5.5 Thiết kế các thành phần chính

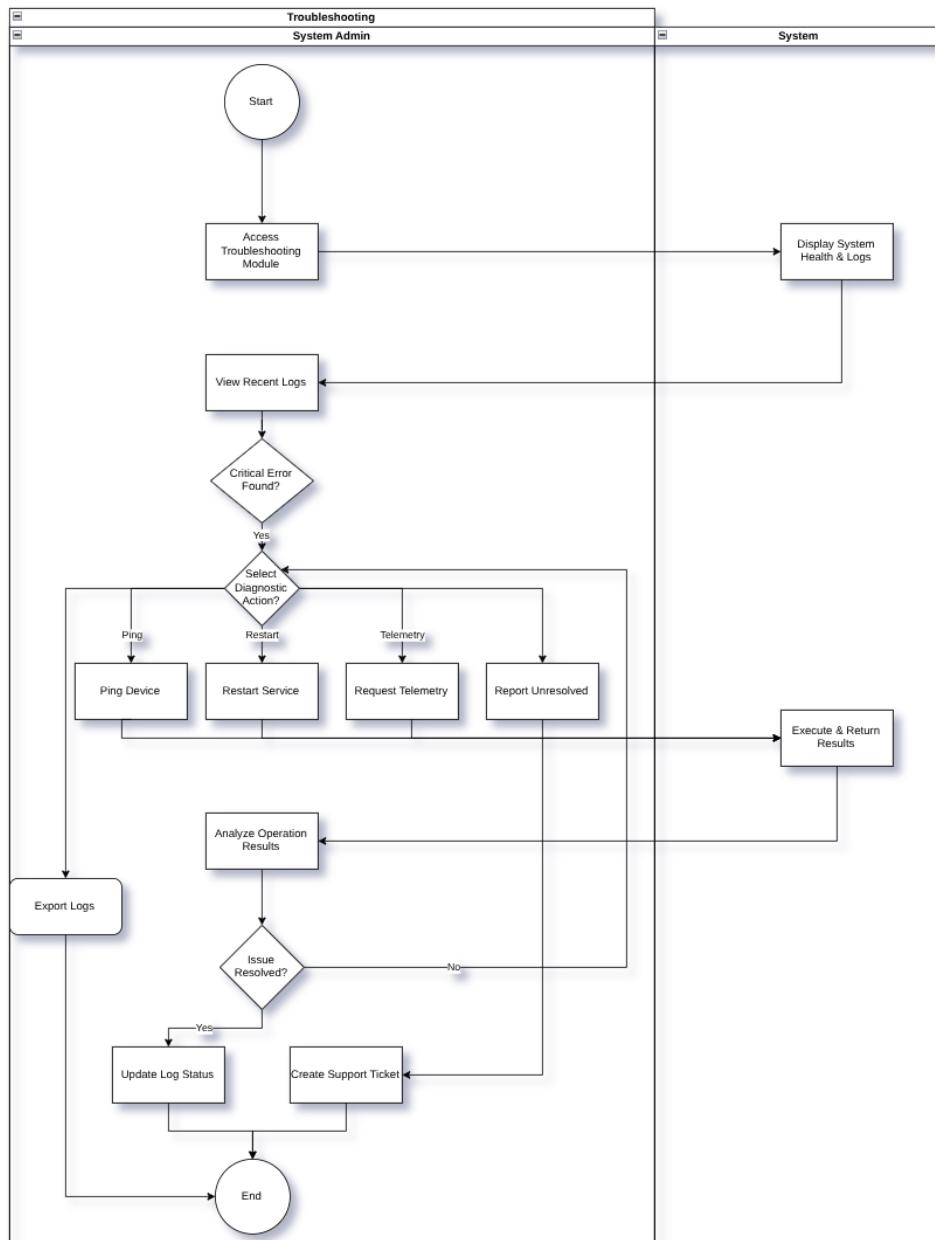
### 5.5.1 Quy trình Giám sát Dashboard (Analytics Flow):



Hình 7: Activity Diagram mô tả luồng giám sát và tương tác trên Dashboard

Sơ đồ thể hiện tính tương tác liên tục: Admin có thể thực hiện nhiều tác vụ (Lọc, Xuất, Cấu hình) và quay lại màn hình chính mà không bị ngắt quãng phiên làm việc.

### 5.5.2 Quy trình Xử lý sự cố (Troubleshoot Flow):

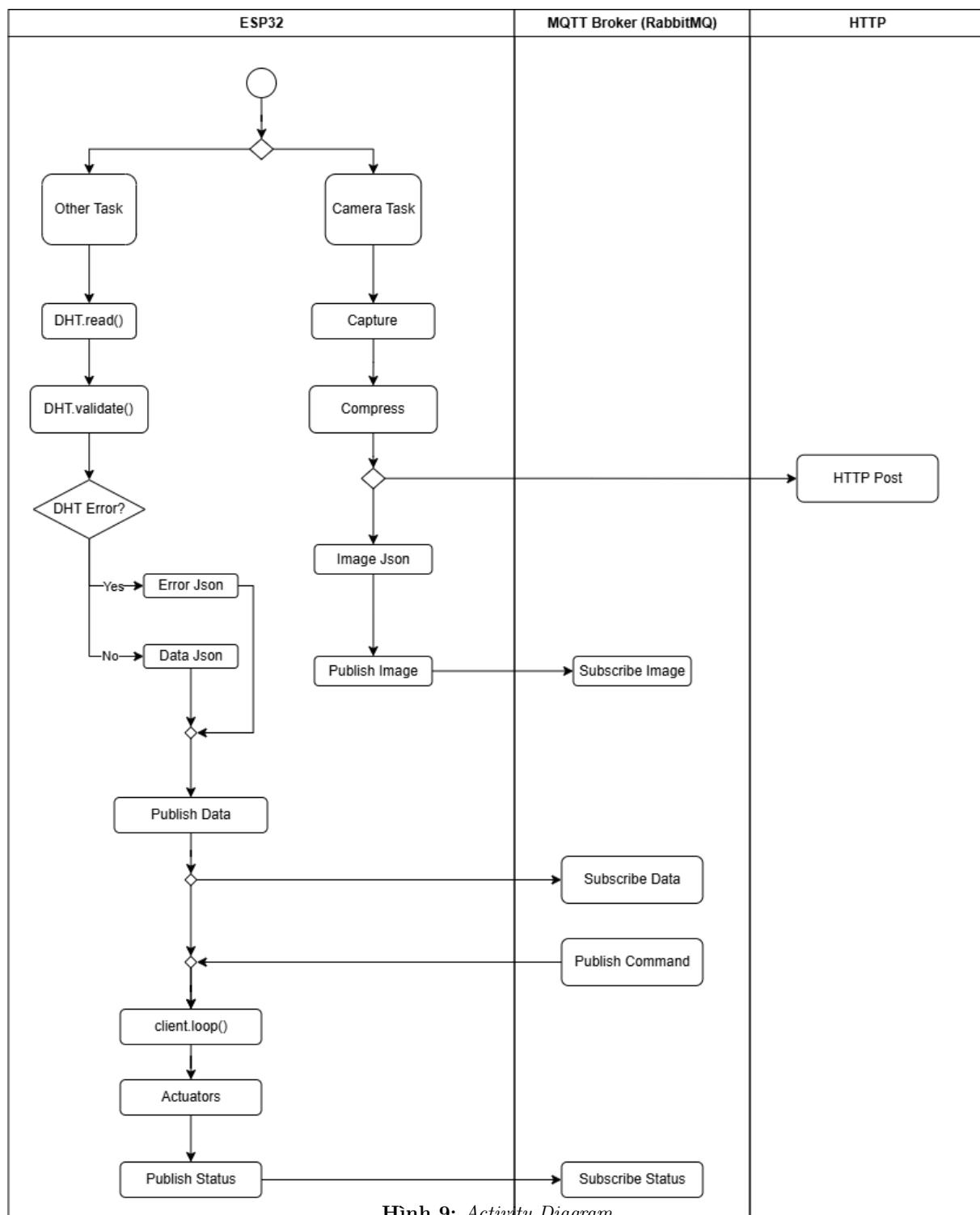


**Hình 8:** Activity Diagram quy trình chẩn đoán và khắc phục sự cố hệ thống

Quy trình đảm bảo mọi sự cố đều có điểm kết thúc rõ ràng: hoặc là được giải quyết (Resolved), hoặc là được chuyển tiếp thành vé hỗ trợ (Support Ticket) để đội ngũ kỹ thuật can thiệp sâu hơn.

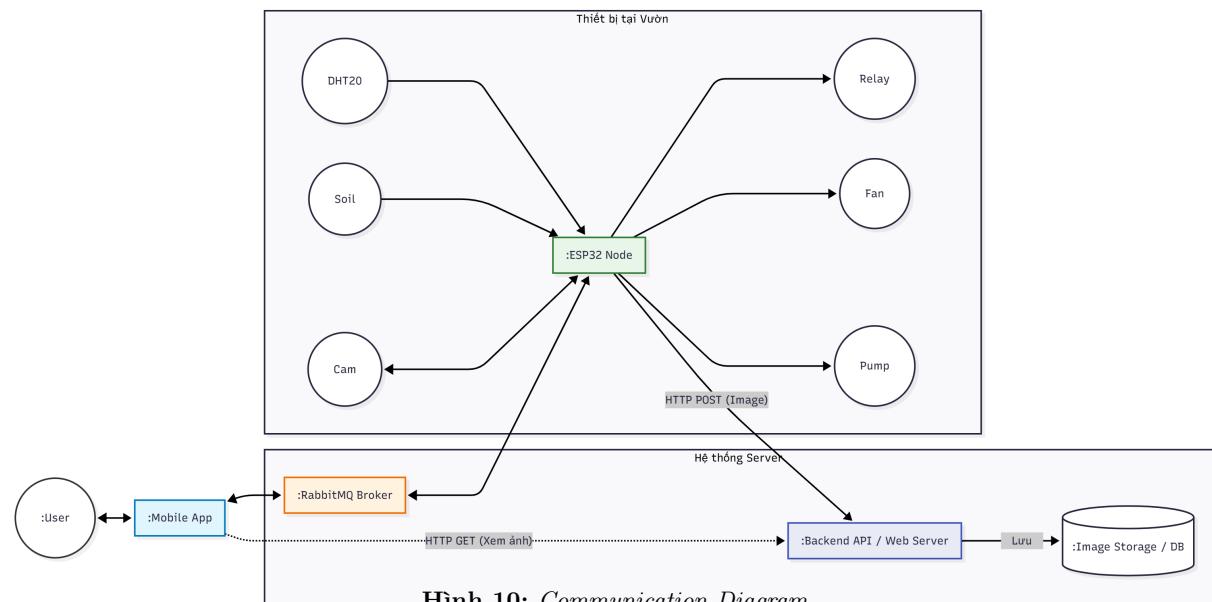
### 5.5.3 Firmware

#### 5.5.3.a Activity Diagram

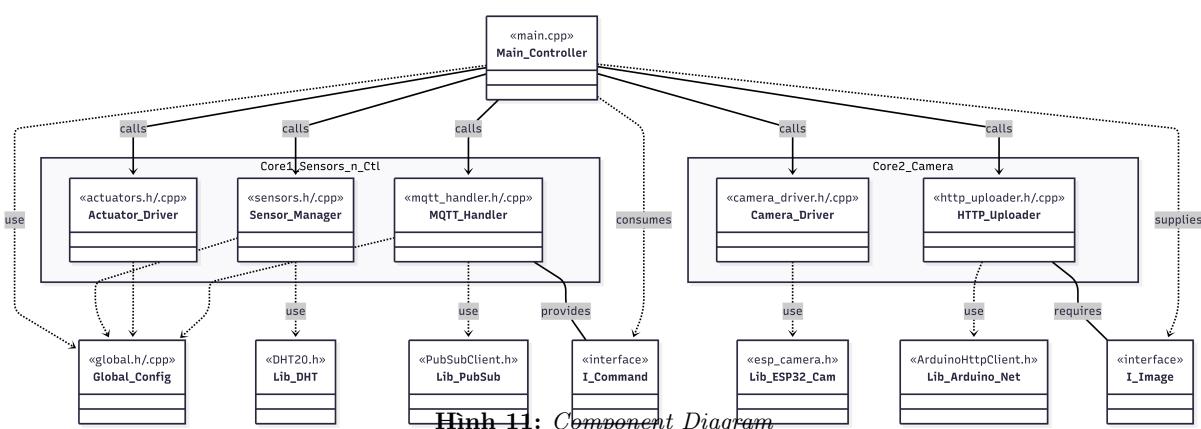


Hình 9: Activity Diagram

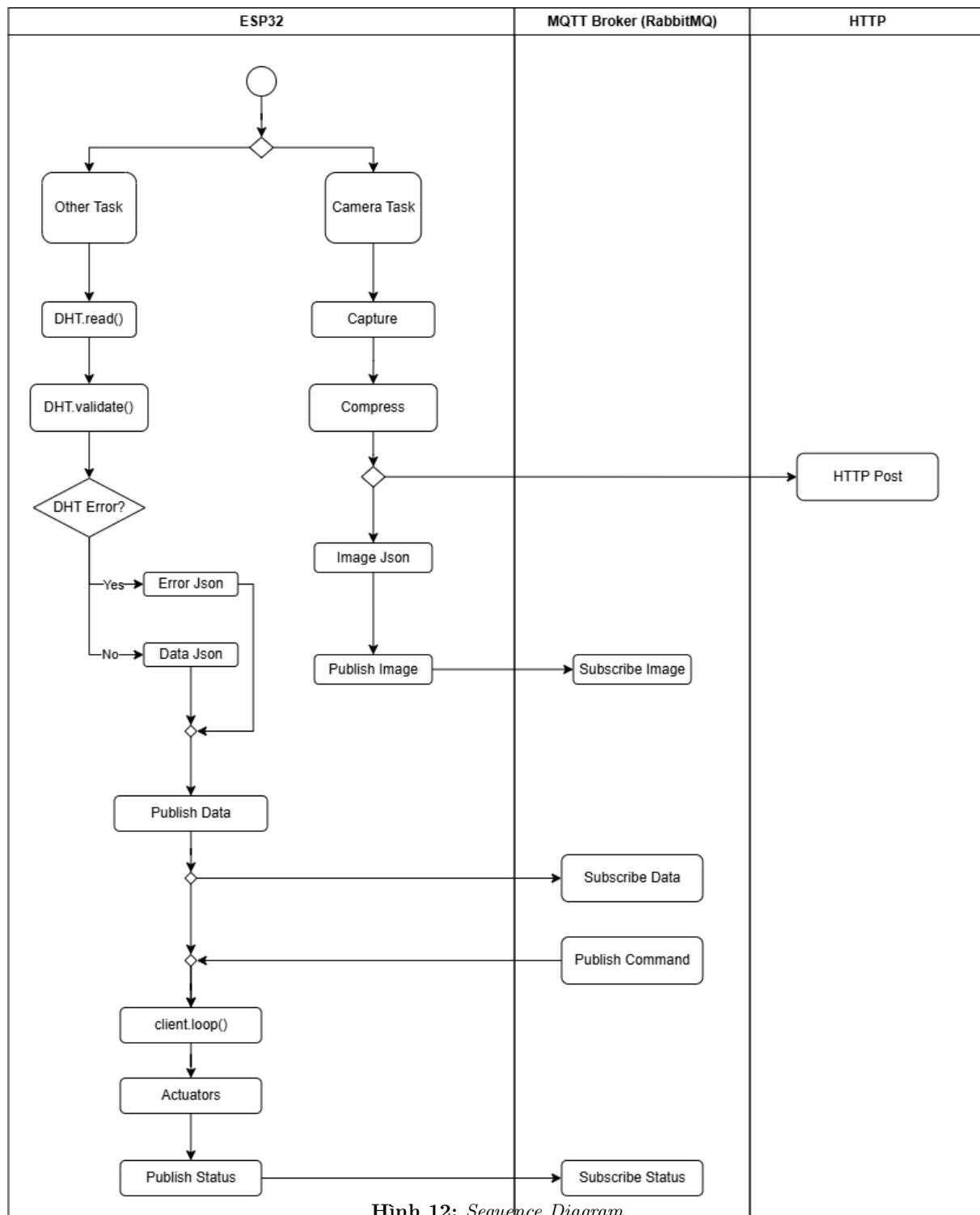
### 5.5.3.b Communication Diagram



### 5.5.3.c Component Diagram



### 5.5.3.d Sequence Diagram



Hình 12: Sequence Diagram

### 5.5.3.e Timing Diagram

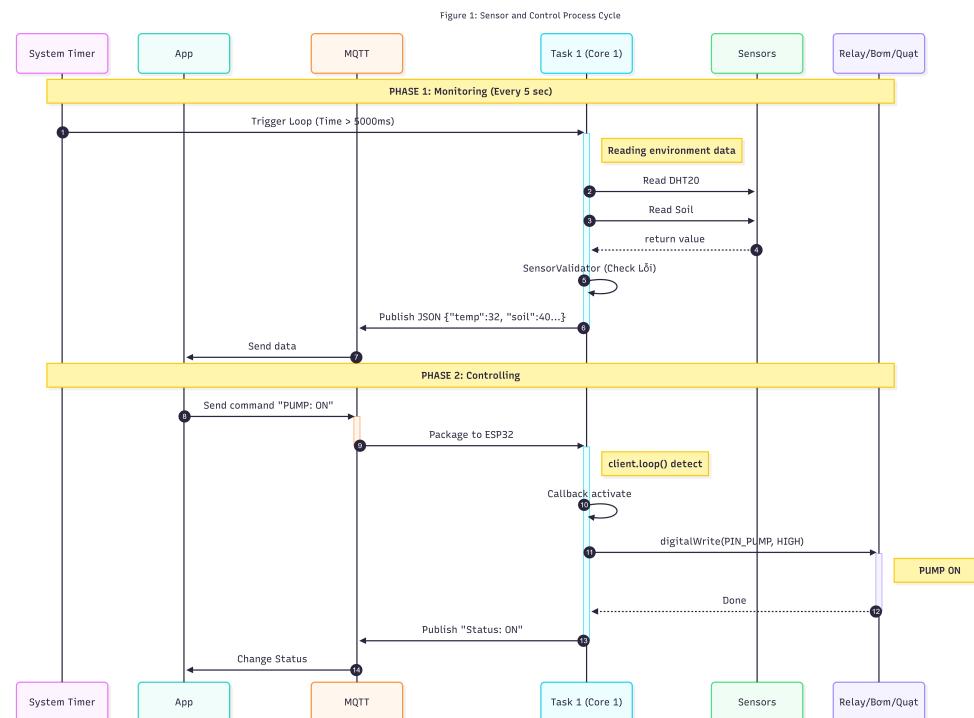
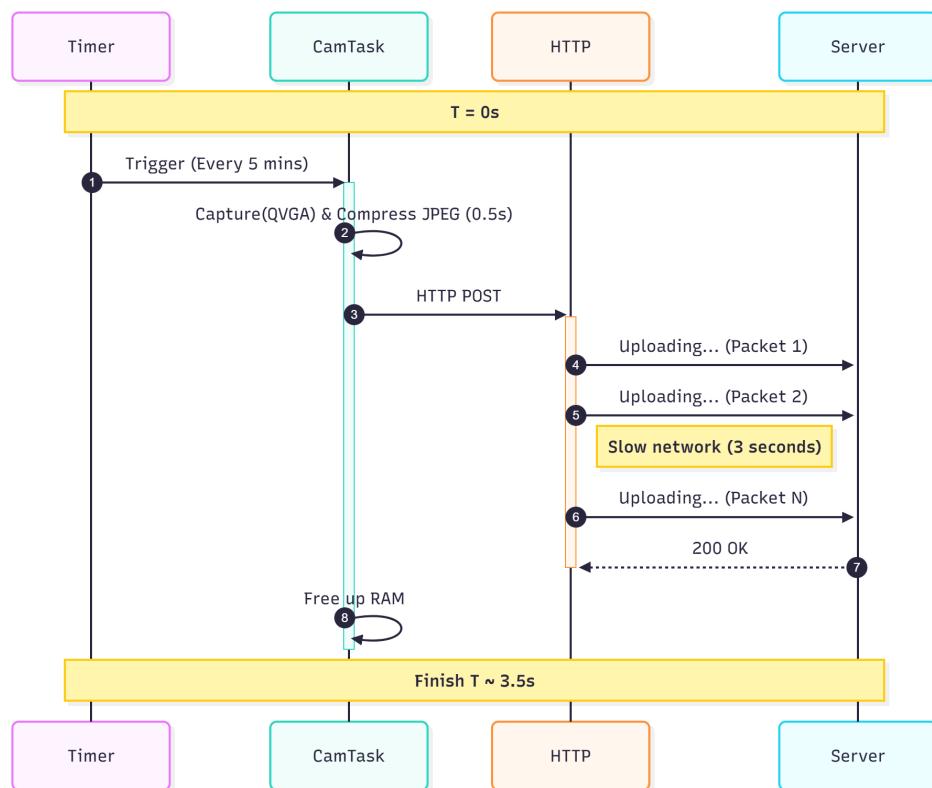
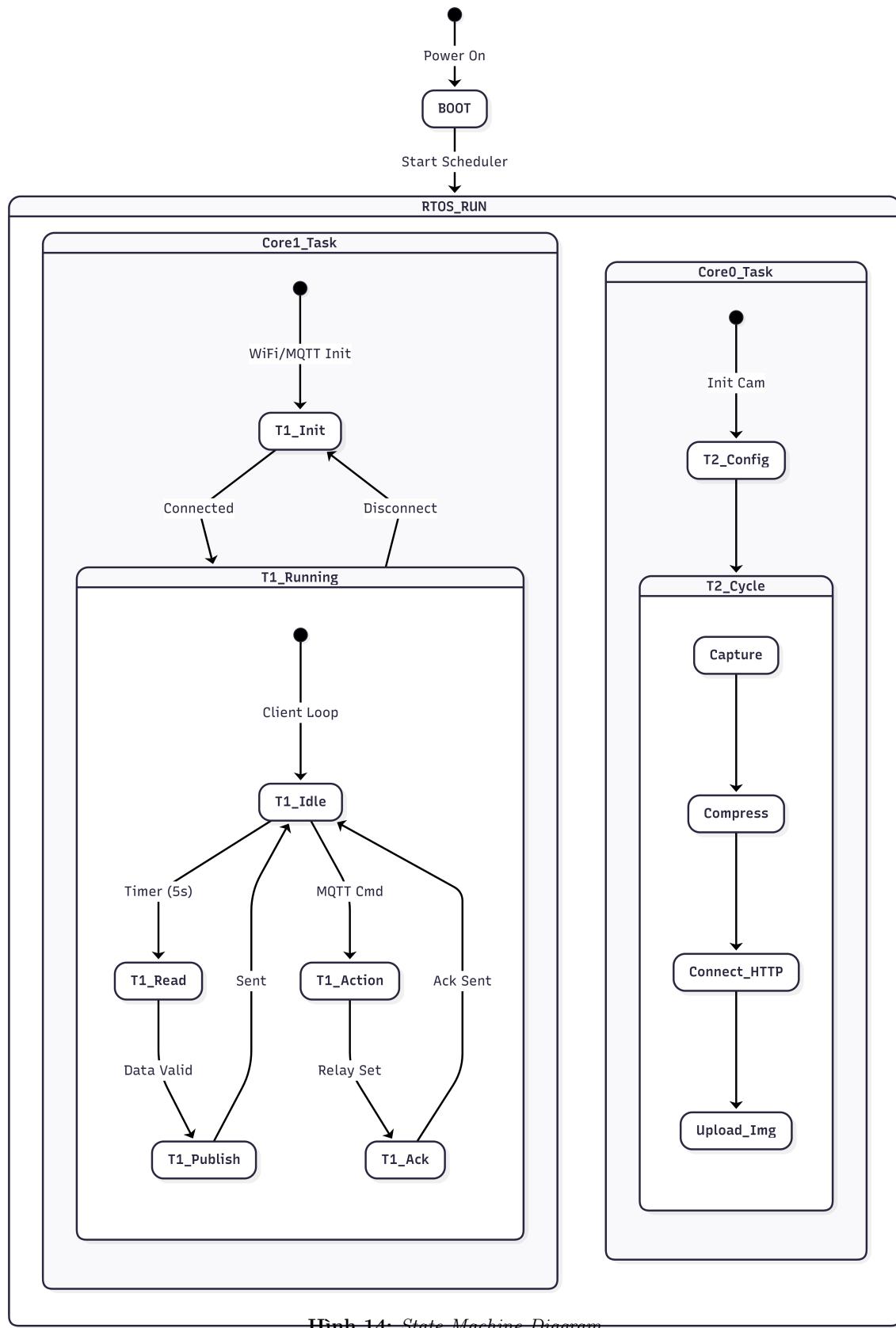


Figure 2: Camera Process Cycle



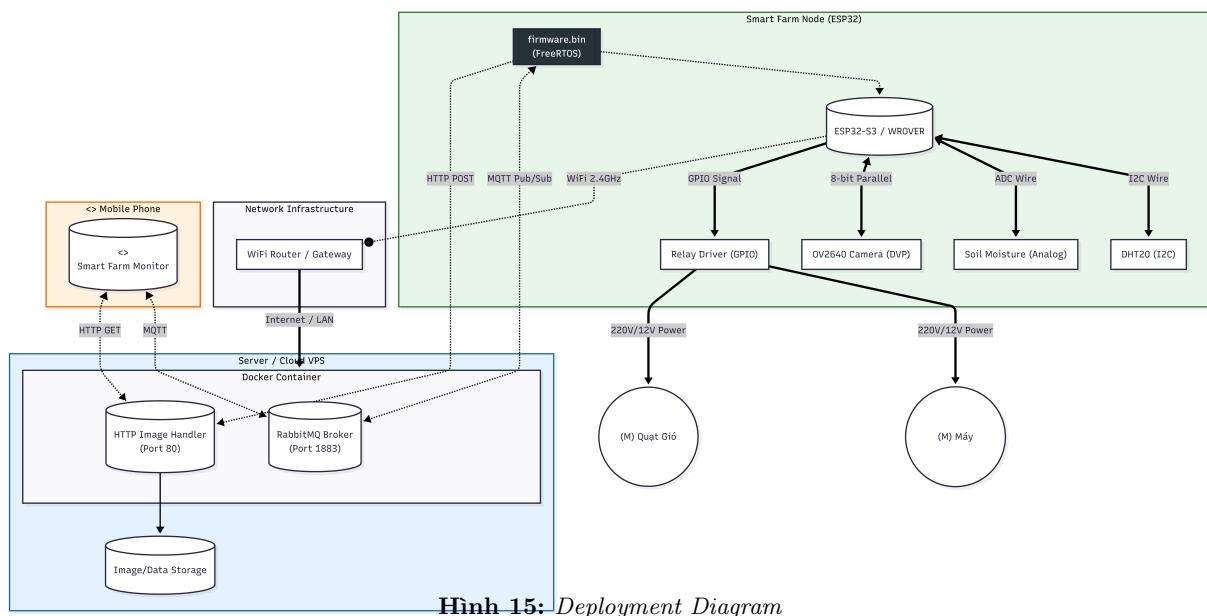
Hình 13: Timing Diagram

### 5.5.3.f State Machine Diagram



Hình 14: State Machine Diagram

### 5.5.3.g Deployment Diagram



### 5.6 Cách đánh giá giải pháp



## 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.
- **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.
  - 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 2: 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"><li>Nghiên cứu Datasheet ESP32, OV2640.</li><li>Tìm hiểu thư viện ArduinoJson, PubSubClient.</li><li>Đặt mua linh kiện phần cứng.</li></ul>	<ul style="list-style-type: none"><li>Chương 1 (Cơ sở lý thuyết).</li><li>Danh sách linh kiện đầy đủ.</li></ul>
3	<ul style="list-style-type: none"><li>Thiết kế sơ đồ khối và sơ đồ nguyên lý (Schematic).</li><li>Kiểm tra hoạt động riêng lẻ của module cảm biến, Relay.</li></ul>	<ul style="list-style-type: none"><li>Bản vẽ sơ đồ nguyên lý.</li><li>Code kiểm thử đơn giản.</li></ul>
4	<ul style="list-style-type: none"><li>Lắp ráp phần cứng hoàn chỉnh (trên mạch in hoặc đục lỗ).</li><li>Kiểm tra nguồn, đo đặc chống nhiễu.</li></ul>	<ul style="list-style-type: none"><li>Mô hình phần cứng thô.</li><li>Chương 2 (Thiết kế phần cứng).</li></ul>
5-6	<ul style="list-style-type: none"><li><b>Cấu hình FreeRTOS:</b> Tạo Task, Queue, Semaphore.</li><li>Lập trình đọc cảm biến (DHT20, Đất) và điều khiển Relay.</li></ul>	<ul style="list-style-type: none"><li>Code khung sườn (Skeleton code).</li><li>Dữ liệu hiển thị trên Serial Monitor.</li></ul>
7-8	<ul style="list-style-type: none"><li><b>Xử lý Camera (Trọng tâm):</b> Cấu hình giao thức DVP, lấy dữ liệu ảnh JPEG.</li><li>Lập trình Web Server để stream ảnh qua HTTP.</li></ul>	<ul style="list-style-type: none"><li>Hình ảnh hiển thị trên trình duyệt.</li><li>Frame rate đạt mức ổn định (&gt;10fps).</li></ul>
9	<ul style="list-style-type: none"><li><b>Kết nối IoT:</b> Lập trình giao thức MQTT.</li><li>Đóng gói dữ liệu dạng JSON.</li></ul>	<ul style="list-style-type: none"><li>Gửi thành công JSON lên Broker.</li><li>Nhận lệnh điều khiển từ xa.</li></ul>
10	<ul style="list-style-type: none"><li>Tích hợp hệ thống: Ghép module Camera + Cảm biến + MQTT.</li><li>Xử lý xung đột tài nguyên giữa 2 nhân (Core 0/1).</li></ul>	<ul style="list-style-type: none"><li>Firmware hoàn chỉnh v1.0.</li><li>Chương 3 (Thiết kế phần mềm).</li></ul>
11	<ul style="list-style-type: none"><li>Xây dựng Dashboard điều khiển (Web app hoặc Mobile app đơn giản).</li></ul>	<ul style="list-style-type: none"><li>Giao diện người dùng (UI) hoạt động.</li></ul>
12	<ul style="list-style-type: none"><li>Chạy thử nghiệm hệ thống liên tục (Stress test).</li><li>Tinh chỉnh người dùng cảm biến, tối ưu bộ nhớ.</li></ul>	<ul style="list-style-type: none"><li>Số liệu thực nghiệm.</li><li>Bảng đánh giá độ ổn định.</li></ul>
13	<ul style="list-style-type: none"><li>Tổng hợp số liệu, vẽ biểu đồ kết quả.</li><li>Viết chương Kết quả và Thảo luận.</li></ul>	<ul style="list-style-type: none"><li>Chương 4 (Kết quả thực nghiệm).</li></ul>
14	<ul style="list-style-type: none"><li>Hoàn thiện toàn bộ báo cáo thuyết minh.</li><li>Chỉnh sửa định dạng, trích dẫn tài liệu tham khảo.</li></ul>	<ul style="list-style-type: none"><li>Bản thảo báo cáo hoàn chỉnh.</li></ul>
15	<ul style="list-style-type: none"><li>Soạn thảo Slide thuyết trình.</li><li>Quay video demo và luyện tập bảo vệ.</li></ul>	<ul style="list-style-type: none"><li>Slide Powerpoint.</li><li>Video Demo sản phẩm.</li></ul>



### 6.2.0.b Software

Bảng 3: Lịch trình thực hiện phần mềm (Backend & Web) trong 15 tuần

Tuần	Công việc chi tiết	Kết quả bàn giao
1-2	<b>Khởi tạo nền tảng:</b> - 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	<b>Quản lý người dùng (Auth):</b> - 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	<b>Quản lý Farm &amp; Device:</b> - 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	<b>Tích hợp IoT (Core):</b> - 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 lỗi định dạng.
9	<b>Cảnh báo thời gian thực:</b> - 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	<b>Frontend - Cơ bản:</b> - Dựng khung Web Admin (React/Vue). - Giao diện Login, Quản lý Farm/Device.	- Giao diện quản lý cơ bản hoạt động.
11-12	<b>Frontend - Nâng cao:</b> - 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	<b>Tích hợp hệ thống:</b> - 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	<b>Tối ưu hóa:</b> - Indexing và Materialized Views cho DB. - Cấu hình nén dữ liệu TimescaleDB.	- Báo cáo hiệu năng hệ thống (Response time).
15	<b>Đóng gói &amp; Báo cáo:</b> - 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 4: Bảng phân tích rủi ro và phương án giảm thiểu

STT	Rủi ro (Risk)	Mức độ	Phương án giảm thiểu
1	<b>Hỏng hóc phần cứng:</b> Cháy ESP32 hoặc Camera do đấu sai nguồn/ngắn mạch.	Cao	<ul style="list-style-type: none"><li>Mua dự phòng 01 bộ linh kiện.</li><li>Kiểm tra kỹ mạch bằng VOM trước khi cấp nguồn.</li></ul>
2	<b>Tràn bộ nhớ (Stack Overflow):</b> Do xử lý ảnh JPEG lớn trên FreeRTOS.	Cao	<ul style="list-style-type: none"><li>Sử dụng hàm: <code>uxTaskGetStackHighWaterMark</code> để giám sát RAM.</li><li>Cân nhắc sử dụng ESP32 có PSRAM.</li></ul>
3	<b>Nhiều tín hiệu:</b> Camera bị sọc hoặc cảm biến đất báo giá trị ảo.	Trung bình	<ul style="list-style-type: none"><li>Sử dụng tụ lọc nguồn 100uF và 100nF.</li><li>Đi dây tín hiệu ngắn gọn, tách biệt với dây động cơ.</li></ul>
4	<b>Độ trễ truyền ảnh cao:</b> Video bị giật, lag khi mạng WiFi yếu.	Trung bình	<ul style="list-style-type: none"><li>Giảm độ phân giải ảnh (VGA/QVGA) khi mạng kém.</li><li>Tách luồng gửi ảnh sang Core 0 độc lập.</li></ul>
5	<b>Chậm tiến độ:</b> Do vướng mắc thuật toán phức tạp.	Thấp	<ul style="list-style-type: none"><li>Ưu tiên xử lý phần Camera trước (phần khó nhất).</li><li>Tham khảo cộng đồng Open Source ESP32.</li></ul>

#### 6.3.0.b Software

Bảng 5: Phân tích rủi ro phần mềm và phương án xử lý

STT	Rủi ro (Risk)	Mức độ	Phương án giảm thiểu
1	<b>Nghẽn cổ chai Database:</b> Khi hàng trăm thiết bị gửi dữ liệu cùng lúc, việc Insert từng dòng gây chậm.	Cao	<ul style="list-style-type: none"><li>Sử dụng kỹ thuật Batch Insert (gom 100-500 bản ghi/lần).</li><li>Tận dụng kiến trúc Hypertables của TimescaleDB.</li></ul>
2	<b>Mất dữ liệu tại Broker:</b> RabbitMQ bị đầy hàng đợi (Queue overflow) nếu Backend xử lý chậm.	Trung bình	<ul style="list-style-type: none"><li>Cấu hình cơ chế Message Acknowledgment (chỉ xóa khi đã xử lý).</li><li>Tăng số lượng Worker (Consumer) trong NestJS.</li></ul>
3	<b>Dữ liệu rác (Invalid Schema):</b> Thiết bị gửi sai định dạng JSON làm lỗi parser.	Cao	<ul style="list-style-type: none"><li>Triển khai lớp Schema Validation (DTO) chặt chẽ tại đầu vào.</li><li>Ghi log gói tin lỗi ra bảng riêng để debug.</li></ul>
4	<b>Truy vấn Dashboard chậm:</b> Load biểu đồ lịch sử 1 năm mất quá nhiều thời gian (>5s).	Trung bình	<ul style="list-style-type: none"><li>Sử dụng Materialized Views để tính sẵn giá trị trung bình theo giờ.</li><li>Cache kết quả truy vấn vào Redis.</li></ul>
5	<b>Lỗi hỏng bảo mật API:</b> Người dùng truy cập vào dữ liệu nông trại của người khác.	Cao	<ul style="list-style-type: none"><li>Áp dụng chặt chẽ Guards và Interceptors.</li><li>Kiểm tra quyền sở hữu (Ownership check) trong mọi API.</li></ul>



## 7 Kết luận



## 8 Phụ lục



## References

- [1] A. D. Boursianis et al., ?Smart farming: Internet of Things technologies, challenges and future directions,? *Sensors*, vol. 22, no. 10, p. 3735, 2022.
- [2] M. Noura, M. Atiquzzaman, and M. Gaedke, ?Interoperability in IoT: Challenges and solutions,? In *Proceedings of the International Conference on IoT*, Springer, 2019, pp. 1–6.
- [3] 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. address: <https://baochinhphu.vn/>
- [4] Vietnam Report, *Báo cáo tăng trưởng và 6 thách thức của nông nghiệp công nghệ cao Việt Nam*, 2024.
- [5] J. Smith and T. Nguyen, ?Intelligent Fault Detection in IoT-Based WSNs for Precision Agriculture Using Machine Learning,? *IEEE Internet of Things Journal*, vol. 12, no. 4, pp. 1023–1035, 2025.
- [6] P. Stapleton et al., ?A review of low-cost precision agriculture: sensors, monitoring and communication systems,? *Journal of Agricultural Science*, vol. 12, no. 3, pp. 55–70, 2023.
- [7] 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*, vol. 17, no. 4, pp. 2347–2376, 2015.
- [8] NestJS Team, *NestJS Documentation: Introduction and Fundamentals*, 2024. address: <https://docs.nestjs.com/>
- [9] Microsoft, *TypeScript: Typed JavaScript at Any Scale*, 2024. address: <https://www.typescriptlang.org/>
- [10] The PostgreSQL Global Development Group, *PostgreSQL 16.0 Documentation*, 2024. address: <https://www.postgresql.org/docs/current/>
- [11] A. Bader et al., ?Time Series Databases: New Ways to Store and Access Data,? *Proceedings of the VLDB Endowment*, 2017.
- [12] Timescale Inc., *TimescaleDB Documentation: Hypertables and Chunking*, Truy cập: 13/12/2025, 2024. address: <https://docs.timescale.com/>
- [13] Espressif Systems, *ESP32 Series Datasheet*, 2024. address: [https://documentation.espressif.com/esp32\\_datasheet\\_en.pdf](https://documentation.espressif.com/esp32_datasheet_en.pdf)
- [14] N. Semiconductors, *I2C Bus Specification and User Manual*, 2021. address: <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>
- [15] I. OmniVision Technologies, *OV2640 Camera Module Datasheet*, 2020. address: [https://www.uctronics.com/download/cam\\_module/OV2640DS.pdf](https://www.uctronics.com/download/cam_module/OV2640DS.pdf)
- [16] E. Systems, *ESP32 Technical Reference Manual*, 2016. address: [https://documentation.espressif.com/esp32\\_technical\\_reference\\_manual\\_en.pdf](https://documentation.espressif.com/esp32_technical_reference_manual_en.pdf)
- [17] OASIS Open, *MQTT Version 5.0 Committee Specification*, 2019. address: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>
- [18] R. Fielding et al., ?Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing,? IETF, Request for Comments 7230, 2014.
- [19] FreeRTOS, *FreeRTOS Scheduler and Scheduling Policies*, 2022. address: <https://www.freertos.org/>
- [20] E. International, *The JSON Data Interchange Syntax (ECMA-404)*, 2017. address: [https://www.ecma-international.org/wp-content/uploads/ECMA-404\\_2nd\\_edition\\_december\\_2017.pdf](https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf)
- [21] T. N. M. Duy, *Improving Feature Extraction for Sensor Fault Detection in Low-Power IoT Systems*, 2025. address: <https://www.researchgate.net/publication/393103517>