

TRƯỜNG ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN ĐIỀU KHIỂN TỰ ĐỘNG

NGUYỄN MINH KHÁNH – 1511514

LUẬN VĂN TỐT NGHIỆP

NỀN TẢNG WEB THIẾT KẾ SCADA VÀ GATEWAY TỰ CẤU HÌNH

DESIGNABLE WEB-BASED SCADA AND SELF-CONFIGURED GATEWAY

KỸ SƯ NGÀNH KỸ THUẬT ĐIỀU KHIỂN & TỰ ĐỘNG HÓA

GIẢNG VIÊN HƯỚNG DẪN

TRƯƠNG ĐÌNH CHÂU

TP. HỒ CHÍ MINH, 2019

TRƯỜNG ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN: ĐIỀU KHIỂN TỰ ĐỘNG

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập - Tự do - Hạnh phúc

TP. HCM, ngày....tháng.....năm.....

NHẬN XÉT LUẬN VĂN TỐT NGHIỆP
CỦA CÁN BỘ HƯỚNG DẪN

Tên luận văn:

NỀN TẢNG WEB THIẾT KẾ SCADA VÀ GATEWAY TỰ CẤU HÌNH /
DESIGNABLE WEB-BASED SCADA AND SELF-CONFIGURED GATEWAY

Nhóm Sinh viên thực hiện:

Nguyễn Minh Khánh

Cán bộ hướng dẫn:

1511514 Trương Đình Châu

Đánh giá Luận văn

1. Về cuốn báo cáo:

Số trang _____ Số chương _____

Số bảng số liệu _____ Số hình vẽ _____

Số tài liệu tham khảo _____ Sản phẩm _____

Một số nhận xét về hình thức cuốn báo cáo:

2. Về nội dung luận văn:

3. Về tính ứng dụng:

4. Về thái độ làm việc của sinh viên:

Đánh giá chung:

Điểm từng sinh viên:

Nguyễn Minh Khánh:...../10

Cán bộ hướng dẫn

(Ký tên và ghi rõ họ tên)

TRƯỜNG ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN: ĐIỀU KHIỂN TỰ ĐỘNG

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập - Tự do - Hạnh phúc

TP. HCM, ngày....tháng.....năm.....

NHẬN XÉT LUẬN VĂN TỐT NGHIỆP

CỦA CÁN BỘ PHẢN BIỆN

Tên luận văn:

**NỀN TẢNG WEB THIẾT KẾ SCADA VÀ GATEWAY TỰ CẤU HÌNH /
DESIGNABLE WEB-BASED SCADA AND SELF-CONFIGURED GATEWAY**

Nhóm Sinh viên thực hiện:

Nguyễn Minh Khánh

Cán bộ phản biện:

1511514

Đánh giá Luận văn

5. Về cuốn báo cáo:

Số trang _____ Số chương _____

Số bảng số liệu _____ Số hình vẽ _____

Số tài liệu tham khảo _____ Sản phẩm _____

Một số nhận xét về hình thức cuốn báo cáo:

6. Về nội dung luận văn:

7. Về tính ứng dụng:

8. Về thái độ làm việc của sinh viên:

Đánh giá chung:

Điểm từng sinh viên:

Nguyễn Minh Khánh:...../10

Người nhận xét

(Ký tên và ghi rõ họ tên)

TRƯỜNG ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN: ĐIỀU KHIỂN TỰ ĐỘNG

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập - Tự do - Hạnh phúc

TP. HCM, ngày....tháng.....năm.....

ĐỀ CƯƠNG CHI TIẾT

TÊN LUẬN VĂN: NỀN TẢNG WEB THIẾT KẾ SCADA VÀ GATEWAY TỰ CẤU HÌNH	
Cán bộ hướng dẫn: Trương Đình Châu	
Thời gian thực hiện: Từ ngày 01/02/2019 đến ngày 01/06/2019	
Sinh viên thực hiện: Nguyễn Minh Khánh – 1511514	
Nội dung đề tài: Xây dựng ứng dụng SCADA trên nền web, kết nối với nhà máy công nghiệp bằng gateway có khả năng tự cấu hình theo dữ liệu trên web.	
Kế hoạch thực hiện: Xây dựng kiến trúc hệ thống, định nghĩa các giao thức truyền thông, lựa chọn công nghệ. Xây dựng ứng dụng SCADA trên nền web. Xây dựng chương trình cho gateway. Ứng dụng SCADA và gateway vào thực tế.	
Xác nhận của Cán bộ hướng dẫn (Ký tên và ghi rõ họ tên)	TP. HCM, ngày....thángnăm..... Sinh viên (Ký tên và ghi rõ họ tên)

DANH SÁCH HỘI ĐỒNG BẢO VỆ LUẬN VĂN

Hội đồng chấm luận văn tốt nghiệp, thành lập theo Quyết định số ngày
..... của Hiệu trưởng Trường Đại học Bách khoa TP.HCM.

1. – Chủ tịch.
2. – Thư ký.
3. – Ủy viên.
4. – Ủy viên.
5. – Ủy viên.

LỜI CẢM ƠN

Trong thời gian làm luận văn tốt nghiệp, em đã nhận được nhiều sự giúp đỡ, đóng góp ý kiến và hướng dẫn nhiệt tình từ thầy cô, gia đình và bạn bè.

Em xin chân thành cảm ơn các thầy cô trong trường Đại học Bách Khoa nói chung và bộ môn Tự động nói riêng đã cung cấp các kiến thức bổ ích, giúp em có cơ sở lý thuyết vững vàng và tạo điều kiện giúp đỡ em trong suốt quá trình học tập. Đặc biệt, em xin cảm ơn thầy Trương Đình Châu đã luôn đồng hành cùng em trong suốt quá trình thực hiện luận văn này. Các đóng góp quý báu của thầy giúp em kịp thời khắc phục lỗi và hoàn thiện đề tài.

Cuối cùng, em xin chân thành cảm ơn gia đình và bạn bè đã luôn tạo điều kiện, quan tâm, giúp đỡ, động viên em trong suốt quá trình học tập cũng như hoàn thành luận văn tốt nghiệp.

Tp Hồ Chí Minh, ngày 29 tháng 5 năm 2019

Sinh viên thực hiện

Nguyễn Minh Khánh

MỤC LỤC

Chương 1. TÍNH CẤP THIẾT, MỤC TIÊU VÀ NHIỆM VỤ ĐỀ TÀI	4
1.1. Tính cấp thiết của đề tài	4
1.2. Mục tiêu, nhiệm vụ của đề tài	4
Chương 2. TỔNG QUAN	6
2.1. Phân tích các đề tài liên quan	6
2.2. Industrial Internet of Things	6
2.3. Các giao thức kết nối nhà máy với server	7
Chương 3. CẤU TRÚC HỆ THỐNG	8
3.1. Cấu trúc tổng quát	8
3.2. Cấu trúc gateway	8
3.3. Cấu trúc server.....	9
Chương 4. XÂY DỰNG SERVER	11
4.1. Cấu hình MQTT broker.....	11
4.1.1. Tổng quan về MQTT.....	11
4.1.1.1. Giới thiệu giao thức MQTT	11
4.1.1.2. Các thuật ngữ quan trọng	12
4.1.2. Cấu hình broker	14
4.1.2.1. Cài đặt Mosquitto MQTT broker	15
4.1.2.2. Cấu hình OpenSSL.....	15
4.1.2.3. Cấu hình Mosquitto broker.....	16
4.1.3. Khởi động broker	17
4.1.4. Danh sách các topic MQTT trong luận văn.....	17

4.2. Cấu hình database	18
4.2.1. Giới thiệu MongoDB	18
4.2.2. Cài đặt MongoDB	20
4.2.3. Cấu trúc dữ liệu.....	20
4.3. Xây dựng web-based SCADA.....	22
4.3.1. Giới thiệu	22
4.3.1.1. HTTP.....	22
4.3.1.2. Websocket và socket.io	23
4.3.1.3. HTML, CSS và Javascript.....	24
4.3.1.4. NodeJS	25
4.3.2. Các chức năng chính của web-based SCADA	26
4.3.3. Chức năng đăng kí, đăng nhập.....	26
4.3.3.1. Đăng kí	27
4.3.3.2. Đăng nhập.....	30
4.3.4. Quản lí gateway	33
4.3.4.1. Tạo gateway mới	34
4.3.4.2. Xoá gateway	38
4.3.4.3. Yêu cầu danh sách cấu hình	39
4.3.4.4. Trạng thái gateway	41
4.3.5. Chức năng thiết kế giao diện	42
4.3.5.1. Giới thiệu	42
4.3.5.2. Nhóm basic	44
4.3.5.3. Nhóm display	50
4.3.5.4. Nhóm control.....	57

4.3.5.5. Nhóm advance	63
4.3.6. Chức năng vận hành thử	68
4.3.7. Chức năng publish	77
Chương 5. XÂY DỰNG CHƯƠNG TRÌNH CHO GATEWAY	81
5.1. Giới thiệu	81
5.2. Các module chức năng.....	83
5.2.1. Module S7	83
5.2.2. Module OPC UA	85
5.2.3. Module alarm	88
5.3. Lập trình chương trình cho gateway.....	90
Chương 6. KẾT QUẢ THỬ NGHIỆM	99
6.1. S7-connection	99
6.2. OPC UA	103
Chương 7. KẾT LUẬN	108
TÀI LIỆU THAM KHẢO	109

DANH MỤC HÌNH ẢNH

Hình 2.1. Mô hình Publish – Subscribe và mô hình Client – Server.....	7
Hình 3.1. Cấu trúc tổng quát hệ thống	8
Hình 3.2. Cấu trúc gateway.....	9
Hình 3.3. Cấu trúc server	10
Hình 4.1. Mô hình hoạt động của MQTT	11
Hình 4.2. Cấu trúc dữ liệu MongoDB	21
Hình 4.3. Giao diện đăng kí.....	28
Hình 4.4. Lưu đồ xử lí đăng kí	29
Hình 4.5. Giao diện đăng nhập	30
Hình 4.6. Lưu đồ kiểm tra đăng nhập	31
Hình 4.7. Giao diện quản lí gateway	34
Hình 4.8. Giao diện tạo gateway mới	35
Hình 4.9. Thêm PLC vào gateway.....	36
Hình 4.10. Thêm danh sách biến cho từng PLC.....	36
Hình 4.11. Xoá gateway	38
Hình 4.12. Children node – Danh sách PLC	40
Hình 4.13. Node details – Danh sách biến	41
Hình 4.14. Trạng thái gateway	41
Hình 4.15. Giao diện thiết kế SCADA	43
Hình 4.16. Polyline.....	45
Hình 4.17. Modal thuộc tính của polyline.....	46
Hình 4.18. Polygon.....	48

Hình 4.19. Line, oval, circle, rectangle, round rectangle.....	48
Hình 4.20. Nhóm display.....	50
Hình 4.21. Thuộc tính của Image	51
Hình 4.22. Thuộc tính của text.....	52
Hình 4.23. Thuộc tính của Display value	53
Hình 4.24. Thuộc tính của Symbol set.....	54
Hình 4.25. Thuộc tính của progress bar	55
Hình 4.26. Nhóm control	57
Hình 4.27. Thuộc tính của Button	57
Hình 4.28. Thuộc tính của Input	58
Hình 4.29. Thuộc tính của switch và checkbox	60
Hình 4.30. Thuộc tính của slider	61
Hình 4.31. Chart và gauge.....	63
Hình 4.32. Thuộc tính của chart.....	64
Hình 4.33. Các thuộc tính của gauge	66
Hình 4.34. Kích hoạt chức năng Run (toolbox bị vô hiệu)	68
Hình 4.35. Màn hình alarm	73
Hình 4.36. Màn hình history	75
Hình 4.37. Giao diện publish.....	77
Hình 4.38. Cấu trúc thư mục người dùng	79
Hình 5.1. Raspberry Pi 3.....	82
Hình 5.2. Simatic IoT2040	83
Hình 5.3. Quá trình kích hoạt alarm.....	88

Hình 5.4. Máy trạng thái alarm.....	90
Hình 5.5. Máy trạng thái gateway.....	91
Hình 5.6. Lưu đồ trạng thái Ready	93
Hình 5.7. Lưu đồ trạng thái Init.....	95
Hình 5.8. Lưu đồ trạng thái Run.....	96
Hình 6.1. Mô hình điều khiển mức chất lỏng.....	99
Hình 6.2. Sơ đồ hệ thống tank liquid control	100
Hình 6.3. Mô hình điều khiển mức chất lỏng - mapping.....	100
Hình 6.4. Giao diện quản lí gateway	101
Hình 6.5. Cấu hình gateway	101
Hình 6.6. Mô hình điều khiển mức nước – chế độ Auto.....	102
Hình 6.7. Mô hình điều khiển mức nước – Chế độ Manual	102
Hình 6.8. Mô hình điều khiển mức nước - Alarm	103
Hình 6.9. Mô hình điều khiển mức nước – Dashboard.....	103
Hình 6.10. Mô hình phân loại sản phẩm theo màu.....	104
Hình 6.11. Mô hình phân loại sản phẩm theo màu.....	105
Hình 6.12. Mô hình phân loại sản phẩm theo màu – mapping.....	105
Hình 6.13. Trạng thái Stop	106
Hình 6.14. Trạng thái Run	106
Hình 6.15. Cảnh báo khi làm việc quá thời gian quy định	107
Hình 6.16. Màn hình cảnh báo.....	107

DANH MỤC BẢNG BIỂU

Bảng 4.1. Danh sách topic MQTT18

Bảng 4.2. So sánh MongoDB và RDBMS19

TÓM TẮT LUẬN VĂN

Luận văn này thực hiện phần mềm thiết kế hệ thống giám sát, điều khiển và thu thập dữ liệu (SCADA) trên nền web. Hệ thống SCADA này có nhiều ưu điểm so với các hệ thống truyền thống, trong đó đáng chú ý là: tính tiện lợi, linh hoạt và giá thành triển khai. Tuy nhiên, hạn chế của các giao thức trên nền web như hiệu năng, tính bảo mật và độ trễ làm cho cách tiếp cận này không được ứng dụng rộng rãi trong công nghiệp. Luận văn tập trung nghiên cứu các công nghệ web tiên tiến để giải quyết những vấn đề nêu trên. Các đề tài trong lĩnh vực này khá ít và chưa đủ sức cạnh tranh với những phần mềm SCADA truyền thống. Sản phẩm của luận văn là một phần mềm SCADA trên nền web, cho phép thiết kế và giám sát nhà máy, đi kèm với gateway tự cấu hình để chuẩn hoá giao thức truyền thông. Sản phẩm này được triển khai vào các hệ thống điều khiển cụ thể nhằm kiểm tra và đánh giá tính khả thi của đề tài.

ABSTRACT

This thesis presents a web-based Supervisory Control And Data Acquisition (SCADA). This SCADA system has more advantages over the other traditional softwares, notably in terms of the mobility, flexibility and deployment cost. However, the limitations of web protocol such as performance, security and delay prevented it from becoming popular in the field of industrial automation. New advanced web technologies are applied to overcome the mentioned issues. There are a few topics about web-based SCADA, but they are not as good as the traditional SCADA softwares. This thesis will result a new web-based SCADA platform to design and monitor the factory plants with a self-configured gateway to standardise the communication protocol. This platform will be applied to the specified control systems to examine and evaluate the feasibility of this thesis.

MỞ ĐẦU

Hệ thống SCADA nói chung và phần mềm SCADA nói riêng là một thành phần quan trọng trong quá trình phát triển nền công nghiệp hiện đại, nhất là với tốc độ hiện đại hoá ngày càng nhanh chóng như hiện nay. Do các hệ thống SCADA đã xuất hiện từ khá lâu nên trên thế giới có rất nhiều phần mềm SCADA của các hãng nổi tiếng, được sử dụng rộng rãi và tính ổn định cao như phần mềm Vijeo Citect của Schneider Electric (Pháp), WinCC của Siemens (Đức), InTouch của WonderWare (Hoa Kỳ), RSView của Rockwell Automation (Hoa Kỳ),...

Các phần mềm SCADA nói trên đều có hiệu năng và độ tin cậy cao, tuy nhiên vẫn tồn tại một số nhược điểm:

- Giá thành quá cao so với các ứng dụng vừa và nhỏ ở Việt Nam.
- Mã nguồn đóng, sử dụng cơ chế mã hoá riêng nên người dùng khó tùy biến.
- Chỉ hỗ trợ hệ điều hành Window, không hỗ trợ các hệ điều hành nhân Unix như Ubuntu, Mac OS.
- Đa số ứng dụng chỉ chạy trong mạng nội bộ (local network), một số có khả năng xuất ra web nhưng muốn truy cập từ xa phải mở port cho router, điều này tiềm ẩn nhiều nguy cơ bảo mật.
- Khó khăn trong việc quản lý số lượng lớn nhà máy ở các vị trí địa lý khác nhau.

Bên cạnh đó, cuộc cách mạng 4.0 đang diễn ra trong môi trường công nghiệp đặt ra yêu cầu mới về thiết bị, phần cứng và phần mềm. Một trở ngại lớn trong việc áp dụng Industrial Internet of Things ở Việt Nam là các thiết bị đa phần sử dụng công nghệ cũ, không được thiết kế để kết nối Internet, mà việc nâng cấp thiết bị tốn rất nhiều thời gian và tiền bạc, đồng thời chứa đựng nhiều rủi ro cho nhà máy. Hệ thống càng lớn thì càng khó thay đổi vì phải dừng hoạt động trong thời gian dài.

Luận văn này hướng đến xây dựng một nền tảng SCADA hiện đại, khắc phục nhược điểm của những phần mềm SCADA hiện có, góp phần gắn kết nền công nghiệp truyền thống với cách mạng công nghiệp 4.0.

Chương 1. TÍNH CẤP THIẾT, MỤC TIÊU VÀ NHIỆM VỤ ĐỀ TÀI

1.1. Tính cấp thiết của đề tài

Thứ nhất, hầu hết các phần mềm SCADA hiện tại chỉ hỗ trợ hệ điều hành Window, không có phiên bản cho các hệ điều hành nhân Unix (ví dụ Ubuntu, MacOS), và cũng không thể truy cập từ xa thông qua các cách thức thông thường. Tạo ra một nền tảng SCADA không phụ thuộc vào hệ điều hành giúp việc thiết kế, quản lý và giám sát nhà máy trở nên dễ dàng hơn, đồng thời đáp ứng yêu cầu của cuộc cách mạng công nghiệp 4.0 là truy cập từ bất kì đâu.

Thứ hai, luận văn liên quan đến SCADA tuy được nhiều sinh viên, học viên thực hiện, nhưng hầu hết dành cho một ứng dụng cụ thể. Các đề tài này phần lớn áp dụng một phần mềm SCADA có sẵn trên thị trường, được cài trực tiếp trên máy tính của người dùng để giao tiếp với hệ thống. Một số ít đề tài làm về SCADA trên nền web, tuy nhiên chỉ dừng lại ở mức thiết kế kiến trúc tổng quát, người dùng cuối chưa thể sử dụng được ngay vì cần phải chỉnh sửa code theo từng yêu cầu cụ thể. Do đó, cần thiết để tạo ra một phần mềm SCADA trên nền web cho phép người dùng tự thiết kế ứng dụng của mình, không cần các thao tác chỉnh sửa code phức tạp.

Thứ ba, trên thực tế các nhà máy thường có nhiều chi nhánh, người quản lý tổng phải chờ kết quả báo cáo của các chi nhánh gửi về mới đánh giá được tình hình sản xuất chung. Do vậy, cần phải có một nền tảng giúp kết nối các nhà máy với nhau để người quản lý có thể nắm bắt kịp thời tình hình tại mỗi nhà máy.

Thứ tư, việc can thiệp sâu vào hệ thống điều khiển và giám sát của từng nhà máy chứa đựng nhiều rủi ro và không mang lại lợi ích kinh tế. Kết nối nhà máy với Internet nhưng vẫn giữ nguyên hệ thống cũ, không ảnh hưởng đến quá trình sản xuất là một yêu cầu cấp thiết.

1.2. Mục tiêu, nhiệm vụ của đề tài

Mục tiêu của luận văn là xây dựng một nền tảng SCADA có thể chạy trên nhiều hệ điều hành khác nhau. Nền tảng SCADA này phải có các chức năng thiết kế, vận hành, cảnh

báo và lưu trữ dữ liệu. SCADA trên nền web là một lựa chọn thích hợp vì tất cả các hệ điều hành đều hỗ trợ trình duyệt web. Ngoài ra, để truyền dữ liệu giữa nhà máy với server, cần xây dựng thêm một gateway. Gateway hỗ trợ các giao thức phổ biến trong công nghiệp (S7-connection, Modbus, OPC UA,...).

Nhiệm vụ cần thực hiện trong đề tài này bao gồm:

- Tìm hiểu và lựa chọn các giao thức sử dụng trong môi trường công nghiệp.
- Tìm hiểu các kỹ thuật lập trình web và lựa chọn kỹ thuật thích hợp với yêu cầu đề tài.
- Xây dựng cấu trúc hệ thống.
- Xây dựng web-based SCADA.
- Xây dựng chương trình cho gateway.
- Ứng dụng SCADA và gateway vào thực tế.

Chương 2. TỔNG QUAN

2.1. Phân tích các đề tài liên quan

Các đề tài về cấu trúc SCADA tổng quát thường rất hiếm, đặc biệt là SCADA trên nền web. Hiện tại, chỉ có một số luận văn cao học về SCADA tự thiết kế, nhưng dành cho nền tảng di động là Android và IOS, còn trên nền web chỉ dừng lại ở mức thiết kế kiến trúc tổng quát, người dùng không thể tự thiết kế giao diện.

Các phần mềm SCADA trên thị trường thường cho phép người dùng xuất bản các thiết kế từ ứng dụng lên web server. Tuy nhiên, để truy cập trang web này từ xa, cần phải mở port cho router gateway, việc này tiềm ẩn nhiều nguy cơ bảo mật. Hơn nữa, các ứng dụng này chỉ hoạt động trong mạng nội bộ, không thể kết nối các nhà máy ở các địa điểm khác. Ngoài ra, hầu hết phần mềm SCADA chỉ hỗ trợ hệ điều hành Window, gây phát sinh chi phí bản quyền.

Một giải pháp gần đây được quan tâm là sử dụng các máy tính nhúng, vừa làm gateway để kết nối với PLC và các thiết bị đo lường thông minh khác, vừa làm web server cho phép người dùng thiết kế và giám sát nhà máy. Đây là giải pháp mới, giảm chi phí triển khai, không phụ thuộc vào hệ điều hành. Tuy nhiên, cũng giống như giải pháp trên, muốn truy cập giám sát thiết bị này từ bên ngoài mạng nội bộ, cần mở port cho router, làm giảm tính bảo mật của hệ thống. Các sản phẩm này hầu hết đến từ những công ty nước ngoài, chưa thấy sản phẩm trong nước đủ sức cạnh tranh trong lĩnh vực này.

Web-based SCADA là một xu hướng mới trong thế giới SCADA. Trên thế giới số lượng công ty trong lĩnh vực này không nhiều. Đây là mảnh đất màu mỡ để các công ty khởi nghiệp có thể thực hiện ý tưởng của mình. Đề tài luận văn này mong muốn tạo ra một nền tảng cho thế hệ web-based SCADA mới, mở đường cho các nghiên cứu sau này.

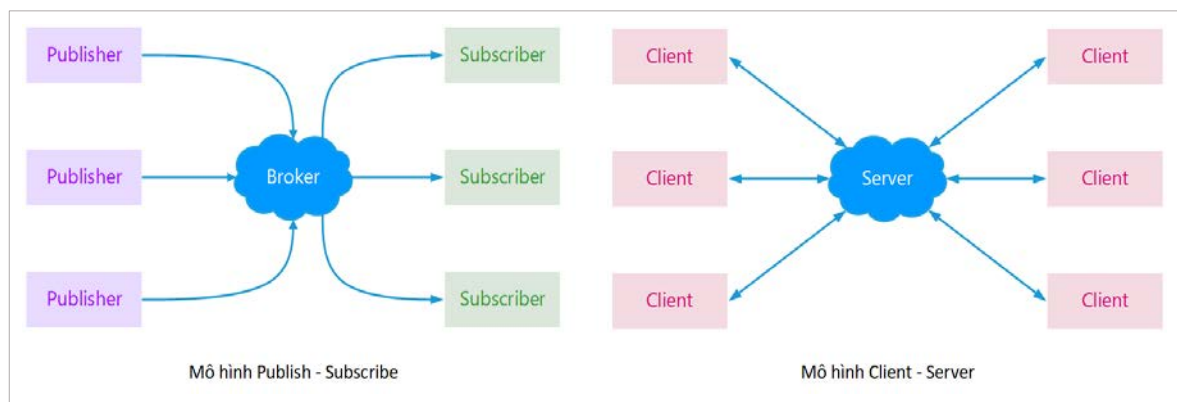
2.2. Industrial Internet of Things

Internet of Things trong công nghiệp, Industrial IoT – IIoT hay còn được gọi là Industrial Internet, hướng đến việc kết nối các hệ thống tự động hoá trong công nghiệp để dễ dàng quản lý, cũng như nâng cao hiệu suất. Một hướng khác của IIoT là thu thập dữ

liệu từ các thiết bị, từ đó đưa ra dự đoán, dự báo về tình trạng làm việc, cũng như phân tích đưa ra các kế hoạch sản xuất dựa trên dữ liệu thu thập được. IIoT giúp tiết kiệm thời gian và tiền bạc, dễ dàng quản lý và mở rộng hệ thống. Hai vấn đề cần quan tâm trong IIoT là khả năng tương tác giữa các thiết bị và tính bảo mật. Các thiết bị trong công nghiệp do nhiều nhà sản xuất cung cấp, mỗi nhà sản xuất sử dụng giao thức truyền thông riêng. Việc kết nối các thiết bị với nhau đòi hỏi một giao thức chuẩn được nhiều nhà sản xuất áp dụng. Các dữ liệu công nghiệp yêu cầu tính bảo mật cao, nếu hệ thống bảo mật không tốt sẽ có nguy cơ bị tấn công, gây tổn thất cho nhà máy.

2.3. Các giao thức kết nối nhà máy với server

Có hai mô hình phổ biến được sử dụng để gửi dữ liệu đến server: client – server và publish – subscribe. Mô hình client – server là loại giao tiếp điểm – điểm, có khả năng bảo mật cao nhưng khó mở rộng. Mô hình publish – subscribe là loại giao tiếp mà mọi publisher (client) sẽ gửi dữ liệu về một trạm trung tâm (gọi là broker), broker sẽ phân phát dữ liệu đến các subscriber (client khác) theo yêu cầu. Các giao thức thường được dùng trong IIoT là OPC UA, HTTP, MQTT, AMQP ... Với công nghệ hiện tại, giao thức MQTT phù hợp hơn vì đây là giao thức sử dụng băng thông thấp, có độ tin cậy cao và khả năng hoạt động trong điều kiện đường truyền không ổn định. AMQP là thế hệ tiếp theo của giao thức MQTT, cho phép trao đổi dữ liệu phức tạp hơn, độ tin cậy cao hơn nhưng lại yêu cầu cao về băng thông và tài nguyên, điều này không phù hợp với môi trường công nghiệp.

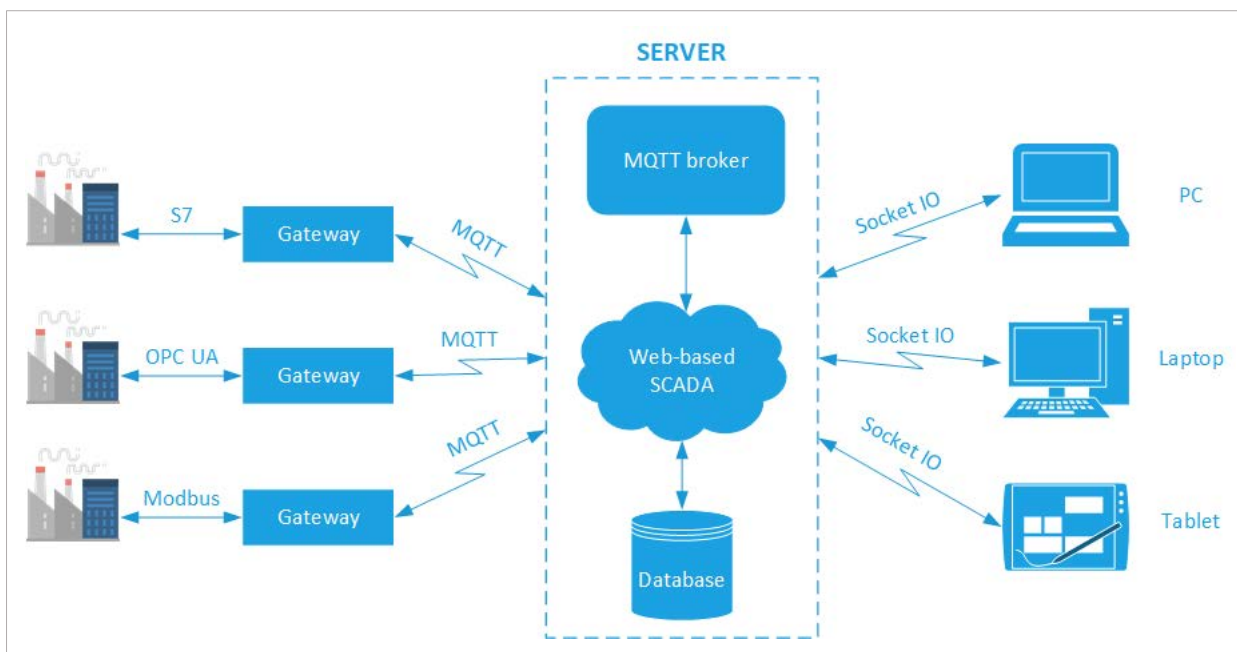


Hình 2.1. Mô hình Publish – Subscribe và mô hình Client – Server

Chương 3. CẤU TRÚC HỆ THỐNG

3.1. Cấu trúc tổng quát

Cấu trúc tổng quát của luận văn có 2 phần: Gateway và server, được thể hiện trong hình 3.1.



Hình 3.1. Cấu trúc tổng quát hệ thống

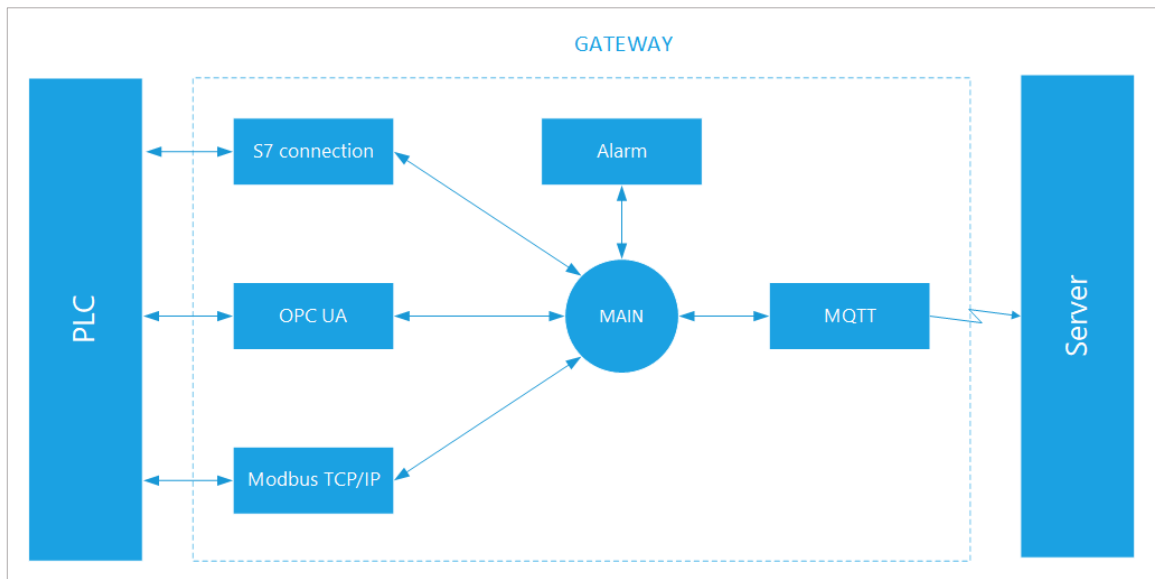
Gateway là thiết bị trung gian, thực hiện nhiệm vụ đọc dữ liệu từ PLC và gửi lên server, đồng thời nhận lệnh từ server gửi về và yêu cầu PLC thực hiện.

Server được xây dựng gồm web-based SCADA, database và MQTT broker. Web-based SCADA cung cấp giao diện thiết kế, quản lý và vận hành các nhà máy. Database kết nối để lưu lịch sử và các cảnh báo hệ thống. MQTT broker là trung gian giao tiếp giữa client và server.

Người dùng sử dụng trình duyệt web để truy cập vào web-based SCADA, thực hiện cấu hình gateway, thiết kế giao diện hay giám sát các nhà máy.

3.2. Cấu trúc gateway

Cấu trúc của gateway được thể hiện qua hình 3.2.

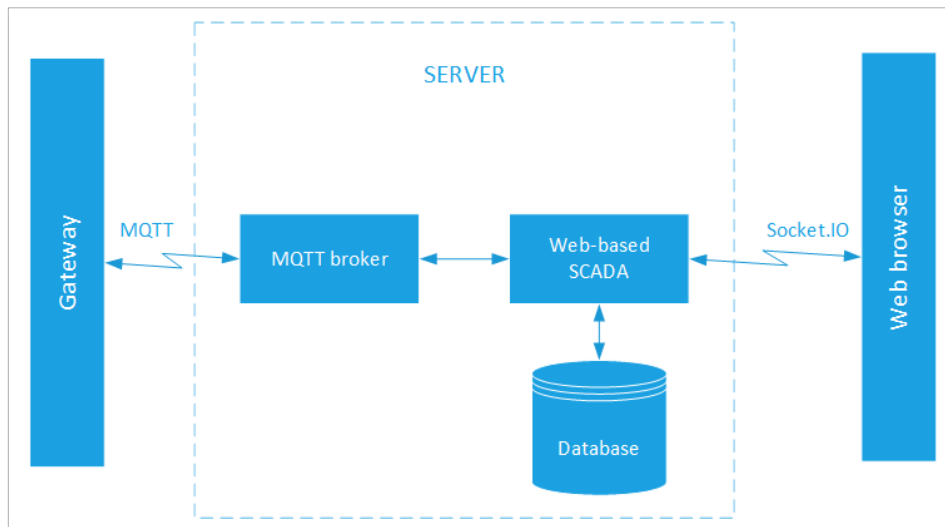


Hình 3.2. Cấu trúc gateway

Gateway có nhiệm vụ kết nối PLC và giao tiếp với server. Mỗi gateway có một mã số duy nhất do nhà sản xuất (ở đây là người lập trình) qui định. Mã số này giúp server nhận dạng và quản lí các gateway khác nhau. Để kết nối với nhà máy, gateway sử dụng các module S7, OPC UA và Modbus TCP/IP để đọc và ghi dữ liệu. Module S7 để giao tiếp với các thiết bị Siemens theo giao thức S7-connection, module OPC UA để giao tiếp với các thiết bị (PLC và thiết bị đo thông minh) theo giao thức OPC UA, module Modbus TCP/IP để giao tiếp với thiết bị (PLC và thiết bị đo thông minh) theo giao thức Modbus TCP/IP. Để giao tiếp với server theo giao thức MQTT, gateway sử dụng module MQTT. Việc cấu hình cho gateway được người dùng thực hiện trực tiếp trên trình duyệt thông qua giao diện web do server cung cấp. Gateway nhận các lệnh mà server gửi về (bao gồm lệnh điều khiển, lệnh cấu hình và lệnh reset), đồng thời gửi giá trị thu thập được cho server xử lí. Ngoài ra, module alarm dùng để kiểm tra, quản lí và thông báo các cảnh báo cho hệ thống SCADA. Các mức cảnh báo do người dùng qui định và được thực hiện thông qua giao diện web.

3.3. Cấu trúc server

Server gồm 3 thành phần chính: MQTT broker, web-based SCADA và database. Cấu trúc tổng quát của server được thể hiện trong hình 3.3.



Hình 3.3. Cấu trúc server

MQTT broker là cầu nối giữa gateway với server. Các gateway gửi dữ liệu theo từng chủ đề đến MQTT broker, broker nhận được và phân phát về server để xử lý. Server phân biệt từng gateway thông mã số định danh của chúng. Sau khi server xử lý xong, sẽ gửi dữ liệu về broker theo đúng mã số đã nhận được, broker nhận và phân phát về gateway.

Web-based SCADA là nơi cung cấp các giao diện quản lý gateway, giao diện thiết kế và giao diện giám sát nhà máy cho người dùng thông qua trình duyệt web. Giao diện quản lý gateway giúp người dùng biết được trạng thái và vị trí của từng gateway. Tại đây, người dùng có thể truy cập vào từng gateway cụ thể để thiết kế hoặc giám sát nhà máy. Giao diện thiết kế được lập trình theo hướng đối tượng, cho phép người dùng tự thiết kế giao diện của riêng mình dựa vào các công cụ có sẵn. Sau khi thiết kế xong, có thể chạy thử để kiểm tra kết quả trước khi xuất bản ra trang web chính thức. Giao diện giám sát nhà máy là kết quả của quá trình thiết kế ở bước trước. Giao diện này cung cấp các cửa sổ để giám sát nhà máy, kiểm soát cảnh báo, xem lịch sử và xuất báo cáo theo yêu cầu người dùng.

Database là một thành phần của server, dùng để lưu trữ dữ liệu lịch sử hoặc cảnh báo. Mọi thao tác với database đều do server quản lý, người dùng không được quyền can thiệp.

Chương 4. XÂY DỰNG SERVER

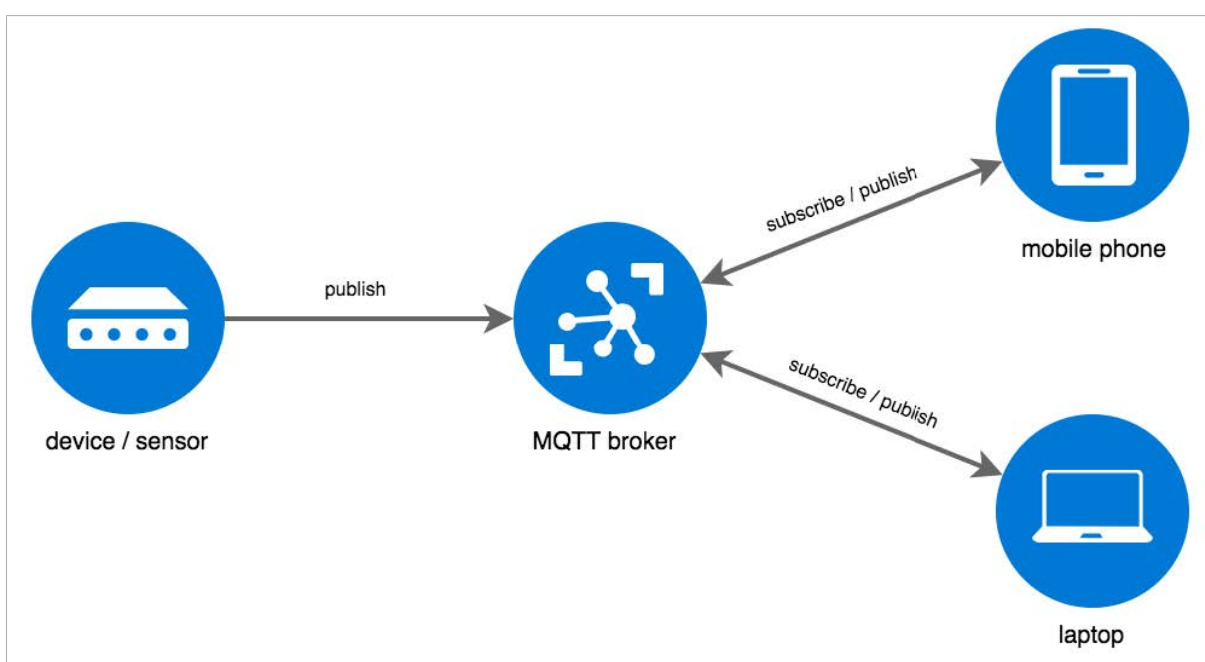
4.1. Cấu hình MQTT broker

4.1.1. Tổng quan về MQTT

4.1.1.1. Giới thiệu giao thức MQTT

MQTT (Message Queuing Telemetry Transport) là giao thức lớp ứng dụng chạy trên nền TCP/IP; là một giao thức đơn giản, nhẹ, dễ thực hiện và sử dụng mô hình publish – subscribe. Những đặc tính này làm cho MQTT phù hợp với các ứng dụng M2M (machine to machine) và IoT (Internet of Things) với các ràng buộc về bộ nhớ và băng thông.

Trong mô hình publish – subscribe, một điểm trung tâm gọi là broker, các điểm khác kết nối với broker được gọi là client. Mọi giao tiếp trong mạng đều phải thông qua broker, các client không giao tiếp trực tiếp với nhau. Các thiết bị gửi dữ liệu gọi là publisher, thiết bị nhận dữ liệu gọi là subscriber. Một thiết bị có thể vừa là publisher, vừa là subscriber. Broker nhận tin nhắn của các publisher theo các chủ đề (topic) cụ thể, sau đó phân phát đến các subscriber đã đăng kí chủ đề đó.



Hình 4.1. Mô hình hoạt động của MQTT

4.1.1.2. Các thuật ngữ quan trọng

○ Các thuật ngữ liên quan đến kết nối

Khi client muốn thiết lập kết nối, nó gửi gói tin CONNECT đến broker. Gói tin này chứa các thông số để thiết lập và duy trì kết nối, quyết định đến hành vi của broker khi client mất kết nối đột ngột. Những thông số cơ bản bao gồm:

- Client Identifier: dùng để xác định client, tên này là duy nhất với mỗi client. Nếu bỏ qua broker sẽ báo lỗi không thể kết nối.
- Clean session: thông báo với broker phiên làm việc mới là clean hay persistent. Nếu cờ này mang giá trị “true” (hay 1) thì broker không lưu lại bất kì thông tin gì về client và xóa hết tất cả thông tin về phiên làm việc trước đó của client. Nếu cờ này mang giá trị “false” (hay 0) thì broker sẽ phục hồi phiên làm việc trước (nếu có) của client, bao gồm các topic được đăng kí và tin nhắn đã bị client bỏ lỡ khi offline (chỉ lưu tin nhắn cuối cùng nếu có).
- Username và password: dùng để xác thực client với broker, được gửi đi dưới dạng text.
- Last will: định nghĩa will topic và will message, được dùng khi client mất kết nối đột ngột, broker sẽ thay mặt client đó gửi thông báo đến các client khác.
- Keep alive: là khoảng thời gian nếu giữa client và broker không có trao đổi dữ liệu thì client sẽ gửi gói tin PINGREQ tới broker, broker trả lời bằng gói PINGRESP. Cơ chế này dùng để kiểm tra và duy trì kết nối giữa client với broker.

○ Các thuật ngữ liên quan đến publish, subscribe

Sau khi tạo kết nối thành công, client gửi dữ liệu đến broker bằng gói tin PUBLISH. Gói tin bao gồm dữ liệu lớp ứng dụng, topic và các thông tin quan trọng khác như cờ retain, Quality of Service (QoS).

Khi client đăng kí nhận thông tin từ topic, nó gửi gói tin SUBSCRIBE đến broker. Gói tin này bao gồm danh sách các topic mong muốn và mức QoS tương ứng. QoS độc lập với

từng topic và client.

Tin nhắn (payload) trong gói PUBLISH hoặc SUBSCRIBE có thể ở bất cứ dạng gì: mã nhị phân, chuỗi thông thường, XML, JSON hay CSV. Broker nhận và phân phát tin nhắn giữa các client mà không quan tâm đến nội dung, client sẽ xử lý nội dung đó tùy theo nhu cầu. Do đó, có thể sử dụng các tin nhắn được mã hoá, các client khác nếu không biết cơ chế mã hoá sẽ không thể giải mã được.

Một số thuật ngữ cần lưu ý:

- Retain: Cờ này báo cho broker biết có lưu lại tin nhắn cuối cùng của topic được publish hay không. Nếu giá trị là “true” (hay 1) thì khi một client khác đăng kí topic đó, ngay lập tức broker sẽ gửi lại tin nhắn cuối cùng được đánh dấu là retain.
- Quality of Service (QoS) là thoả thuận giữa client và broker về tính đảm bảo của dữ liệu. QoS độc lập với từng client và topic. Có 3 mức QoS trong MQTT. Mức 0 (nhiều nhất một lần) là mức độ đơn giản nhất, client chỉ gửi tin nhắn đi một lần mà không chờ xác nhận từ thiết bị nhận, do đó nếu gói tin bị mất hay thành công thì thiết bị nhận chỉ nhận được nhiều nhất một lần. Với mức 1 (ít nhất một lần), khi gói tin PUBLISH được gửi đi, client sẽ chờ xác nhận bằng gói PUBACK; sau mỗi thời gian timeout, nếu vẫn chưa nhận được gói PUBACK thì sẽ gửi lại gói PUBLISH. Với QoS từ mức 1 trở lên, mỗi gói tin sẽ được đánh dấu bằng một Package Identifier, dựa vào đó bên gửi và bên nhận có thể phân biệt các gói tin với nhau. QoS mức 2 (chỉ một lần) là mức độ tin cậy cao nhất. Bên nhận chỉ nhận mỗi gói tin PUBLISH một lần duy nhất. Để làm được điều này, ít nhất 4 gói tin được trao đổi giữa bên gửi và bên nhận. Đầu tiên bên gửi sẽ gửi gói PUBLISH, khi nhận được, bên nhận sẽ lưu lại package identifier của gói này để tránh việc xử lý một gói tin nhiều lần. Tiếp theo, bên nhận gửi gói PUBREC để xác nhận. Bên gửi sau khi nhận được sẽ gỡ bỏ gói PUBLISH đã gửi thành công, đồng thời xác nhận bằng gói PUBREL. Nếu nhận được, bên nhận sẽ xoá packet identifier đã lưu và kết thúc bằng

gói PUBCOMP. Nếu vượt quá khoảng thời gian quy định mà không nhận được gói tin thì sẽ gửi lại gói tin đó.

- **Các thuật ngữ liên quan đến bảo mật**

Client identifier, username và password là cách thức bảo mật đơn giản nhất trong giao thức MQTT. Ở phương thức bảo mật này, ba thông tin trên sẽ được client gửi đến broker trong gói CONNECT. Broker xác thực client và chỉ client được cấp quyền mới được truy cập vào các tài nguyên được cho phép. Để làm được điều này, broker phải được cấu hình từ trước khi bắt đầu kết nối. Các thông tin gửi đi đều ở dạng văn bản, con người có thể đọc được.

Để tăng cường tính bảo mật, có thể sử dụng thêm giao thức mã hoá TLS thay cho quá trình xác thực thông thường. Khi sử dụng TLS, các thông tin này được mã hoá và xác thực bằng chứng chỉ số X.509. Đây là giải pháp có giá trị cao về bảo mật, tuy nhiên lại phức tạp và yêu cầu phần cứng phải tính toán nhiều, do đó có yêu cầu cao về tốc độ xử lý và bộ nhớ của thiết bị.

4.1.2. Cấu hình broker

Có nhiều cloud MQTT broker được cung cấp miễn phí trên Internet như cloudmqtt.com hay rabbitmq.com, tuy nhiên các loại broker này có nhiều hạn chế như giới hạn số lượng client, hạn chế số lượng nhắn trao tin mỗi ngày, hạn chế băng thông và phương pháp bảo mật. Với ứng dụng cần trao đổi dữ liệu lớn như SCADA, việc tự tạo và cấu hình MQTT broker riêng là điều cần thiết để đảm bảo băng thông và bảo mật. Có nhiều phần mềm cho phép tạo MQTT broker, trong phạm vi luận văn này ta chọn phần mềm Mosquitto MQTT vì một số lý do sau:

- Mã nguồn mở, cung cấp hoàn toàn miễn phí, có thể viết lại mã nguồn để đáp ứng nhu cầu.
- Cho phép số lượng lớn client kết nối đồng thời (trên 100.000 client).
- Không giới hạn số lượng tin nhắn trao đổi trong ngày.
- Cài đặt đơn giản, không chiếm nhiều tài nguyên hệ thống.

- Có thể kết hợp với các plugin của bên thứ ba để tạo ra nhiều phương thức bảo mật.

Các ứng dụng trong công nghiệp yêu cầu độ bảo mật cao, với SCADA thì điều này lại càng quan trọng. Nếu hệ thống SCADA bị tấn công sẽ ảnh hưởng nghiêm trọng đến quy trình sản xuất, gây thiệt hại cho công ty. Chính vì vậy, MQTT broker cần phải được xây dựng với độ bảo mật cao nhất có thể. Đối với giao thức MQTT, bảo mật bằng TLS/SSL 2 chiều (có xác thực client) hiện đang là phương pháp bảo mật an toàn nhất. Tuy nhiên, nếu xét theo phương diện kinh tế, với mỗi gateway sản xuất ra cần phải tạo riêng một chứng chỉ để xác thực với broker thì sẽ tốn nhiều thời gian và công sức, làm giảm hiệu quả kinh tế. Ngoài ra, phương pháp bảo mật TLS/SSL 1 chiều cũng đã cho độ tin cậy rất cao, các hãng công nghệ lớn như Microsoft, Amazon khi xây dựng MQTT broker cho riêng mình cũng chỉ sử dụng phương pháp này. Do đó, luận văn này sẽ xây dựng MQTT broker với phương pháp bảo mật TLS/SSL 1 chiều. Trong phương pháp này, broker cung cấp một chứng chỉ, các client khác khi muốn kết nối phải đưa ra chứng chỉ này, broker kiểm tra và quyết định cho client kết nối hay không. Đây là một giải pháp tiện lợi, có độ tin cậy cao.

4.1.2.1. Cài đặt Mosquitto MQTT broker

Truy cập trang web <https://mosquitto.org/download/>, lựa chọn phiên bản phù hợp với hệ điều hành rồi tiến hành cài đặt theo các hướng dẫn.

Để có thể cấu hình broker bảo mật TLS/SSL, cần cài đặt thêm phần mềm OpenSSL tại địa chỉ <https://slproweb.com/products/Win32OpenSSL.html>.

4.1.2.2. Cấu hình OpenSSL

Mở command window, truy cập vào thư mục **\bin** trong đường dẫn cài OpenSSL, thường là: **C:\Program Files\OpenSSL-Win64\bin**.

Trước hết, cần tạo một chứng chỉ và mã xác thực gốc, các chứng chỉ sau này đều được xây dựng từ chứng chỉ và mã xác thực này.

```
openssl req -new -x509 -days [duration] -keyout [outkey] -out [outCA]
```

Trong đó:

- Duration là thời gian hợp lệ của chứng chỉ, đơn vị là ngày. Ví dụ 365 tương đương 365 ngày.
- Outkey là đường dẫn để xuất file .key.
- OutCA là đường dẫn để xuất file .crt.

Tiếp theo, tạo mã xác thực và đăng kí chứng chỉ cho server bằng mã xác thực đó.

```
openssl genrsa -out [serverKey] [keyLength]  
openssl req -out [serverCertificate] -key [serverKey] -new
```

Trong đó:

- ServerKey là đường dẫn để xuất file .key cho server.
- KeyLength là độ dài mã xác thực, tùy vào yêu cầu của hệ thống.
- ServerCertificate là đường dẫn để xuất file .crt cho server.

Cuối cùng, gửi chứng chỉ của server cho chứng chỉ gốc để đăng kí và xác nhận.

```
openssl x509 -req -in [serverCertificate] -CA [outCA] -Cakey [outKey] -  
CAcreateserial -out [finalServerCertificate] -days [period]
```

Trong đó:

- FinalServerCertificate là đường dẫn để xuất file .crt, đây là file chứng chỉ mà server cung cấp cho các client khi cần kết nối.
- Period là thời gian chứng chỉ hợp lệ, đơn vị là ngày.

4.1.2.3. Cấu hình Mosquitto broker

Sau khi đã chuẩn bị các chứng chỉ và mã xác thực cần thiết, cũng trong command window, đi đến thư mục cài đặt mosquitto, thường là: **C:\Program Files\mosquitto**.

Chỉnh sửa file mosquitto.conf theo nhu cầu sử dụng. Đây là file cấu hình mặc định của mosquitto broker. Trong luận văn này, ta chỉnh sửa nội dung file như sau:

```
port 8883
cafile [path to outCA]
certfile [path to finalServerCertificate]
keyfile [path to serverKey]
require_certificate false
tls_version [version]
```

Trong đó:

- Port là cổng sẽ chạy broker, mặc định là 1883 cho broker không bảo mật bằng TLS/SSL, 8883 cho broker bảo mật bằng TLS/SSL.
- Path to outCA: Đường dẫn đến file chứng chỉ gốc.
- Path to finalServerCertificate: Đường dẫn đến file chứng chỉ server đã được đăng kí bằng chứng chỉ gốc.
- Path to serverKey: Đường dẫn đến file mã xác thực của server.
- Require_certificate: “false” nếu xác thực một chiều, “true” nếu xác thực hai chiều.
- Version: Phiên bản TLS muốn sử dụng (ví dụ: tlsv1.1)

4.1.3. Khởi động broker

Tại đường dẫn thư mục cài mosquitto, chạy lệnh sau để khởi động broker:

```
mosquitto -c [path to config file] -v
```

Trong đó: “*path to config file*” là đường dẫn đến file mosquitto.conf đã cấu hình ở mục 4.1.2.3.

Có thể kiểm tra hoạt động của broker bằng một phần mềm MQTT client (ví dụ như MQTT.fx).

4.1.4. Danh sách các topic MQTT trong luận văn

Danh sách các topic MQTT trong luận văn được trình bày cụ thể ở bảng 4.1, với device ID là mã định danh của từng gateway.

STT	Topic	Ghi chú
1	/device ID/config	Gateway subscribe để nhận file cấu hình mới từ server
2	/device ID/status	Will topic dùng để thông báo trạng thái của gateway. Khi gateway online sẽ publish lên topic này, khi offline broker sẽ thay mặt gateway publish.
3	/device ID/tags	Gateway publish để gửi giá trị các biến thu thập được về server
4	/device ID/write	Gateway subscribe để nhận yêu cầu thay đổi giá trị biến từ server
5	/device ID/alarm	Gateway publish để gửi các cảnh báo cho server
6	/device ID/resAlarm	Gateway subscribe để nhận xác nhận cảnh báo từ server
7	/+/tags	Server subscribe để nhận giá trị biến từ tất cả các gateway
8	/+/status	Server subscribe để nhận trạng thái từ tất cả các gateway
9	/+/alarm	Server subscribe để nhận cảnh báo từ tất cả các gateway

Bảng 4.1. Danh sách topic MQTT

4.2. Cấu hình database

4.2.1. Giới thiệu MongoDB

MongoDB là một hệ quản trị cơ sở dữ liệu mã nguồn mở thuộc họ NoSQL, được phát triển bởi MongoDB Inc. Nó được thiết kế theo hướng đối tượng, các bảng trong MongoDB được cấu trúc rất linh hoạt, cho phép dữ liệu lưu trữ trên bảng không cần tuân theo một cấu trúc nhất định nào cả. Dữ liệu trong MongoDB được lưu trữ dưới dạng mã nhị phân gọi là BSON (Binary JSON). Thông tin liên quan được lưu trữ cùng nhau để giảm thời gian truy vấn. MongoDB thường được sử dụng trong các dự án lập trình dùng NodeJS vì định dạng JSON (Javascript Object Notation) hỗ trợ tất cả các kiểu dữ liệu của ngôn ngữ lập trình Javascript (JS).

Một trong những lợi ích hàng đầu mà MongoDB cung cấp là việc sử dụng các lược đồ động, giúp loại bỏ nhu cầu xác định trước cấu trúc dữ liệu (ví dụ như định nghĩa các

cột, các kiểu dữ liệu). Mô hình này giúp tăng tính cơ động khi biểu diễn các mối quan hệ phân cấp, lưu trữ mảng và khả năng thay đổi cấu trúc mà không ảnh hưởng đến dữ liệu sẵn có.

MongoDB kế thừa một số tính năng của các hệ quản trị cơ sở dữ liệu quan hệ (RDBMS - ví dụ như MySQL, Microsoft SQL Server), đồng thời cũng thay đổi một số khái niệm để phù hợp hơn với mô hình dữ liệu. Bảng 4.2 so sánh một số thuật ngữ cơ bản của hai hệ quản trị cơ sở dữ liệu này.

MongoDB	RDBMS	Ý nghĩa
ACID Transactions	ACID Transactions	Khả năng đảm bảo tính toàn vẹn, hợp lệ, an toàn và bền vững của dữ liệu
Collection	Table	Bảng dữ liệu, bên trong chứa nhiều “cột” và “hàng”
Document	Row	Một “hàng” trong bảng dữ liệu
Field	Column	Một “cột” trong bảng dữ liệu
Secondary Index	Secondary Index	Chỉ mục để nâng cao khả năng tìm kiếm
Embedded documents, \$lookup & \$graphLookup	JOINS	Kết hợp các bảng có cùng thuộc tính
Aggregation Pipeline	GROUP_BY	Gom nhóm nhiều dữ liệu khác nhau để xử lý và trả về kết quả đơn lẻ

Bảng 4.2. So sánh MongoDB và RDBMS

Các tổ chức thuộc mọi quy mô đang áp dụng MongoDB vì nó cho phép xây dựng các ứng dụng nhanh hơn, xử lý dữ liệu đa dạng và quản lý ứng dụng hiệu quả hơn. So với RDBMS, MongoDB vượt trội hơn ở các điểm sau:

- Tốc độ truy vấn (find, update, insert, delete) nhanh hơn hẳn. Thử nghiệm thực tế cho thấy, tốc độ insert của MongoDB nhanh gấp 100 lần so với hệ quản trị cơ sở dữ liệu MySQL (thuộc họ RDBMS). MongoDB có hiệu năng cao như vậy là do dữ liệu được lưu với dạng JSON, khi insert nhiều đối tượng thì nó sẽ insert một mảng JSON giống như insert một đối tượng. Hơn nữa, dữ liệu trong MongoDB không có sự ràng buộc lẫn nhau như trong RDBMS, do đó nó không cần mất thời gian kiểm tra tính thoả mãn của dữ liệu. Một nguyên nhân nữa là mọi dữ liệu trong MongoDB đều được đánh chỉ mục (index) nên tốc độ truy vấn rất nhanh.

- Khả năng mở rộng nhanh chóng mà các RDBMS khác không có được. Khi triển khai hệ thống, MongoDB có thể dễ dàng chỉnh sửa và nâng cấp, trong khi các RDBMS khác lại đòi hỏi tùy chỉnh đáng kể.

Bên cạnh các ưu điểm đó, MongoDB cũng có những nhược điểm mà khi thiết kế hệ thống cần lưu ý:

- Không có các tính chất ràng buộc như trong RDBMS nên khi thao tác dữ liệu phải hết sức cẩn thận.
- Hao tốn tài nguyên hệ thống nhiều hơn RDBMS. Tuy nhiên các máy tính hiện tại đã mạnh lên rất nhiều, do đó vấn đề này cũng không còn đáng lo ngại.

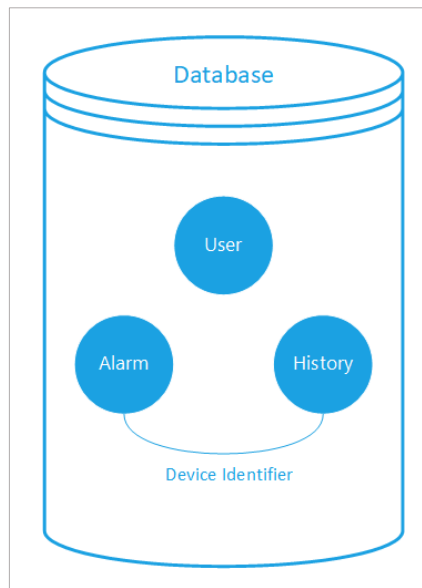
MongoDB được sử dụng rộng rãi trong các hệ thống thời gian thực (realtime), yêu cầu độ phản hồi nhanh, các hệ thống big data với số lượng truy vấn lớn, các hệ thống có tần suất write/ insert lớn hoặc dùng làm search engine. Đối với các ứng dụng SCADA, với đặc thù về dữ liệu và realtime, MongoDB phù hợp hơn so với các RDBMS khác đang có trên thị trường.

4.2.2. Cài đặt MongoDB

MongoDB được cung cấp hoàn toàn miễn phí cho nhiều nền tảng hệ điều hành khác nhau (Window, Linux, MacOS). Người dùng truy cập trang web <https://www.mongodb.com/download-center/community>, lựa chọn phiên bản phù hợp và tiến hành cài đặt theo hướng dẫn của chương trình.

4.2.3. Cấu trúc dữ liệu

Luận văn sử dụng một database MongoDB duy nhất với nhiều collections, được chia làm 3 nhóm: user, alarm và history.



Hình 4.2. Cấu trúc dữ liệu MongoDB

Nhóm user: Gồm 1 collection duy nhất là “user”, chứa thông tin về các người dùng. Thông tin này có được khi người đăng kí dịch vụ và được dùng để xác minh khi đăng nhập. Các field của collection này là:

- firstName: Họ người dùng.
- lastName: Tên người dùng.
- email: Email đăng kí dịch vụ.
- password: Mật khẩu tài khoản, đã được mã hoá trước khi lưu.
- createTime: Thời gian tạo tài khoản.

Nhóm alarm: Gồm nhiều collection, mỗi collection ứng với một gateway và được đặt tên theo cú pháp: **“Device_” + Device ID + “_alarm”**. Nếu gateway có device ID là 12345 thì trong database có collection là “Device_12345_alarm”. Nhóm này dùng để ghi nhận các cảnh báo xảy ra trong quá trình vận hành, giúp người vận hành kiểm tra lại khi cần thiết. Việc chia collection theo từng gateway mặc dù khiến database trông phức tạp hơn, nhưng làm cho mỗi gateway trở nên độc lập khi xử lí. Hơn nữa, việc lưu cảnh báo của nhiều gateway vào cùng một collection làm tăng kích thước collection (MongoDB giới hạn dung lượng mỗi collection là 16MB) và giảm hiệu suất truy vấn. Các field của nhóm collection này bao gồm:

- source: Tên biến gây ra cảnh báo.
- value: Giá trị gây ra cảnh báo.
- message: Thông tin về cảnh báo.
- type: Loại cảnh báo.
- state: Trạng thái của cảnh báo (UNACK hay ACKED).
- timestamp: Thời điểm xảy ra cảnh báo.

Nhóm history: Tương tự như nhóm alarm, nhóm này cũng gồm nhiều collection, mỗi collection ứng với một gateway và được đặt tên theo cú pháp: ***“Device_” + Device ID + “_history”***. Nhóm này ghi nhận các giá trị của những biến đã được cấu hình trước đó, làm căn cứ để xuất báo cáo sau này. Các field của nhóm collection này bao gồm:

- tag: Tên biến.
- type: Kiểu dữ liệu.
- address: Địa chỉ biến.
- value: Giá trị của biến.
- timestamp: Thời điểm ghi nhận.

4.3. Xây dựng web-based SCADA

Web-based SCADA là cầu nối giữa người dùng và hệ thống SCADA, là nơi cung cấp mọi chức năng và tiện ích để xây dựng giao diện và giám sát nhà máy. Đây là nội dung trọng tâm của luận văn.

4.3.1. Giới thiệu

4.3.1.1. HTTP

HTTP (HyperText Transfer Protocol) là giao thức lớp ứng dụng phổ biến nhất được dùng trong World Wide Web. HTTP sử dụng mô hình server – client, trong đó server thường được gọi là web server, phản hồi lại dữ liệu khi client yêu cầu (request). HTTP vận hành trên nền TCP/IP.

HTTP định nghĩa 6 phương thức để chỉ hành động mà server phải thực hiện. Mỗi

request mà client gửi đi có một phương thức chứa trong phần header. Các phương thức thường dùng là GET, POST, PUT và DELETE.

- GET được dùng để lấy hay yêu cầu một tài nguyên, các tham số cần thiết được đặt trong URL gửi đi. Khi truy cập vào một trang web, phương thức GET được sử dụng, các request bằng phương thức này sẽ lưu lại trên trình duyệt. Vì vậy, với các dữ liệu nhạy cảm, không được dùng phương thức GET.
- POST được dùng để tạo một tài nguyên trên server, thông tin của tài nguyên cần tạo được gửi kèm trong phần body của request. Trình duyệt không lưu lại POST request nên POST thường được dùng để gửi dữ liệu nhạy cảm (thông tin cá nhân, username, password,...).
- PUT được dùng để cập nhật tài nguyên trên server, dữ liệu cập nhật được gửi kèm trong phần body của request.
- DELETE dùng để xóa một tài nguyên trên server.

Trong giao thức HTTP, mã trạng thái là một mã gồm ba chữ số được gửi trong response cho biết kết quả của request. Có 5 nhóm mã trạng thái:

- 1xx: Đã nhận được request, đang xử lý.
- 2xx: Thành công.
- 3xx: Tài nguyên không còn tồn tại hoặc được chuyển tới URL khác.
- 4xx: Lỗi của client, ví dụ như gửi thiếu tham số, thiếu thông tin xác thực, tài nguyên bị cấm hoặc sai phương thức.
- 5xx: Lỗi của server, server không thể xử lý request.

4.3.1.2. *Websocket và socket.io*

Đối với HTTP, client gửi request đến server bằng AJAX. Nhược điểm của phương pháp này là với mỗi request cần phải thiết lập lại một kết nối và kích thước header rất lớn. Bên cạnh đó, chỉ có client có thể bắt đầu một kết nối, server không thể chủ động gửi dữ liệu đến client. Giao thức Websocket ra đời để khắc phục những nhược điểm này. Đây là giao thức truyền thông cho phép giao tiếp hai chiều giữa client và server chỉ trên

một đường kết nối giữa chúng (full-duplex single socket connection). Websocket đơn giản hoá việc giao tiếp hai hướng trên nền web. Websocket tạo kết nối bằng cách gửi HTTP request nhưng với phần header đặc biệt, giúp duy trì kết nối, có thể truyền và nhận dữ liệu như một TCP socket. Phần header sau khi đã kết nối có kích thước 2 byte, ít hơn rất nhiều so với 40 byte của HTTP truyền thống.

Websocket là sự lựa chọn hoàn hảo cho giao tiếp giữa client và web-based SCADA vì một số lí do sau:

- Hiệu suất cao: các giao tiếp thời gian thực trở nên hiệu quả hơn, tiết kiệm băng thông, giảm gánh nặng xử lí cho hệ thống.
- Đơn giản hoá quá trình thiết lập kết nối và xử lí dữ liệu ở cả client và server.
- Websocket là giao thức đã được chuẩn hoá, được hỗ trợ bởi đa số các trình duyệt hiện đại, đồng thời có nhiều thư viện hỗ trợ.

Socket.io là một thư viện sử dụng công nghệ Websocket, hỗ trợ giao tiếp hai chiều giữa client và server. Nó được xây dựng nhằm mục đích tạo ra các ứng dụng thời gian thực trên nền web, giúp các bên có thể kết nối với nhau, truyền dữ liệu ngay lập tức thông qua server trung gian. Socket.io thường được dùng trong các ứng dụng chat, game online hay cập nhật kết quả của một trận đấu đang diễn ra,...

4.3.1.3. HTML, CSS và Javascript

HTML (HyperText Markup Language) là ngôn ngữ đánh dấu siêu văn bản, được dùng để tạo ra một trang web. HTML không phải là một ngôn ngữ lập trình, đồng nghĩa với việc nó không thể tạo ra các chức năng động. Ban đầu, HTML được phát triển với mục đích xác định cấu trúc của các tài liệu, ví dụ như tiêu đề, các đoạn văn, các danh sách,... và tạo sự thuận lợi cho việc chia sẻ thông tin khoa học giữa các nhà nghiên cứu. Ngày nay, HTML được sử dụng rộng rãi để định dạng cấu trúc và bố cục một trang web với số lượng tag ngày càng phong phú.

CSS (Cascading Style Sheet Language) là ngôn ngữ tạo phong cách cho trang web. Nó dùng để định dạng những thành phần được tạo ra bằng HTML. CSS có thể định dạng

nhiều trang web cùng lúc, giúp tiết kiệm công sức của người lập trình. Về lý thuyết, CSS chỉ tác động đến phần hiển thị của trang web, không ảnh hưởng đến nội dung, có thể được bỏ qua. Tuy nhiên, một trang web không có CSS sẽ trở nên rất đơn điệu và nhàm chán!

Javascript là một ngôn ngữ lập trình hướng đối tượng, đa nền tảng và hỗ trợ kịch bản. Javascript có thể chạy trên nhiều nền tảng công nghệ khác nhau như web, ứng dụng desktop, game và máy chủ. Với lợi thế là một ngôn ngữ kịch bản, nó có thể chạy ngay mà không cần biên dịch. Trong lĩnh vực lập trình web, Javascript cùng HTML và CSS là bộ ba ngôn ngữ mà bất kì nhà phát triển web nào cũng phải học. Nếu việc xây dựng một trang web giống như thiết kế một robot, thì HTML chính là khung sườn của robot, CSS là trang trí hình dạng của robot và Javascript giúp tạo nên các chuyển động, các hiệu ứng cho robot. Ngày nay, một xu hướng đang nổi là xây dựng các ứng dụng web cho máy chủ bằng các framework của Javascript như NodeJS, AngularJS,... Với cùng một cấu hình máy chủ, khi số lượng người dùng truy cập quá lớn thì PHP, Java, Python hay .NET sẽ không thể xử lý được, nhưng với các framework của Javascript thì mọi chuyện sẽ hoàn toàn khác. Chính vì vậy mà Javascript đang ngày càng trở nên phổ biến trong thế giới lập trình.

4.3.1.4. NodeJS

Một trong những lợi thế lớn nhất của Javascript giúp nó phù hợp với các ứng dụng thời gian thực là “Asynchronous Event – Driven Non – Blocking I/O”. I/O là quá trình giao tiếp giữa một hệ thống tin học với môi trường bên ngoài, ví dụ như giữa CPU với ngoại vi, đọc ghi file. Trong các ngôn ngữ khác, ví dụ như C, khi đọc dữ liệu một file, chương trình bị chặn cho đến khi có kết quả trả về. Đối với Javascript, khi thực hiện một dòng lệnh I/O, chương trình không bị chặn (non – blocking) mà chạy lệnh kế tiếp. Khi quá trình đọc I/O trả về kết quả, chương trình quay lại nhận kết quả và xử lý (event – driven). Như vậy, các I/O có thể chạy song song, thời gian thực hiện của các quá trình khác nhau, nên lệnh gọi trước có thể thực hiện và trả kết quả sau, gọi là bất đồng bộ (asynchronous). Một đặc điểm nữa của Javascript là một process chỉ thực thi một

thread, vì vậy chỉ dùng một lõi của CPU. Để tận dụng CPU có nhiều lõi, có thể cho nhiều process chạy song song. Web server và gateway là những ứng dụng thời gian thực, có nhiều module thành phần nên Javascript là sự lựa chọn phù hợp.

NodeJS là một nền tảng cho phía server được xây dựng dựa trên Google Engine V8, nhằm giúp lập trình viên thực thi backend của một hệ thống web trực tiếp bằng Javascript. Kế thừa các đặc điểm của Javascript như hướng sự kiện (event – driven), không chặn I/O (non – blocking I/O) nhằm cung cấp một nền tảng gọn nhẹ và hiệu quả, NodeJS thích hợp với các web-app thời gian thực. Do được Google tối ưu hoá nên tốc độ thực thi của NodeJS rất ấn tượng, vượt xa các ngôn ngữ thông dịch khác như Ruby, Python.

Với việc lựa chọn ngôn ngữ Javascript làm nền tảng dẫn đến còn nhiều thư viện, tiện ích thiết thực cho việc lập trình web chưa đủ để thoả mãn yêu cầu hiện tại. Tuy nhiên, với cộng đồng người dùng lớn, nhiều hệ sinh thái mã nguồn mở ra đời, là nơi để mọi người chia sẻ những module cần thiết trong quá trình xây dựng web. Hiện tại, NPM (Node Package Manager) là hệ sinh thái module NodeJS lớn nhất thế giới.

4.3.2. Các chức năng chính của web-based SCADA

Web-based SCADA phải có đầy đủ các chức năng sau:

- Đăng nhập, đăng kí.
- Quản lí các gateway trực quan.
- Thêm mới hoặc sửa gateway dễ dàng.
- Thiết kế giao diện HMI cơ bản: hình ảnh, nút nhấn, input, chart, gauge, alarm, history,...
- Vận hành thử khi thiết kế.
- Xuất bản giao diện ra trang web chính thức.

4.3.3. Chức năng đăng kí, đăng nhập

Đăng kí, đăng nhập là hai chức năng cơ bản mà bất kì một website nào cũng phải có. Để thực hiện được hai chức năng này, ta cần cài thêm các thư viện hỗ trợ:

- Cookie-parser: đọc cookie trong request mà client gửi đến server, thường được dùng để đọc thông tin đăng nhập hoặc đăng kí.
- Express-session: công cụ để định danh người dùng hiện tại, thông tin được lưu lại để khi người dùng chuyển trang không cần phải đăng nhập lại.
- Connect-flash: là một vùng nhớ đặc biệt trong session, dùng để lưu trữ tin nhắn. Những tin nhắn được gửi bằng flash sẽ tự động xoá đi sau khi xuất hiện cho người dùng.
- Express-validator: công cụ kiểm tra và xác thực dữ liệu mà client gửi lên server.
- Bcryptjs: mã hoá mật khẩu dạng băm. Kỹ thuật băm là từ một chuỗi đầu vào có độ dài bất kì, đi qua hàm băm sẽ cho đầu ra là một chuỗi có độ dài cố định, gọi là giá trị băm. Độ dài và nội dung của giá trị băm phụ thuộc vào thuật toán băm. Không thể có nhiều đầu vào cùng cho đầu ra giống nhau. Từ giá trị băm, không thể tính ngược lại giá trị đầu vào. Do đó, phương pháp này thường được dùng để bảo vệ mật khẩu người dùng, nếu cơ sở dữ liệu bị lộ cũng không thể giải mã được mật khẩu.
- Mongoose: giao tiếp với cơ sở dữ liệu MongoDB, dùng để lưu trữ, kiểm tra dữ liệu người dùng.
- Passport: xác thực yêu cầu đăng nhập của client, hỗ trợ toàn diện các phương pháp đăng nhập như username – password, email, Facebook, Twitter,... Trong luận văn này, tài khoản người dùng được quản lí thông qua email.
- Passport-local: kết hợp với passport để thực hiện chức năng đăng nhập. Do mọi thông tin về người dùng đều nằm trên máy chủ (localhost), không sử dụng chức năng đăng nhập bằng Gmail, Facebook hay Twitter nên cần sử dụng thêm module này.

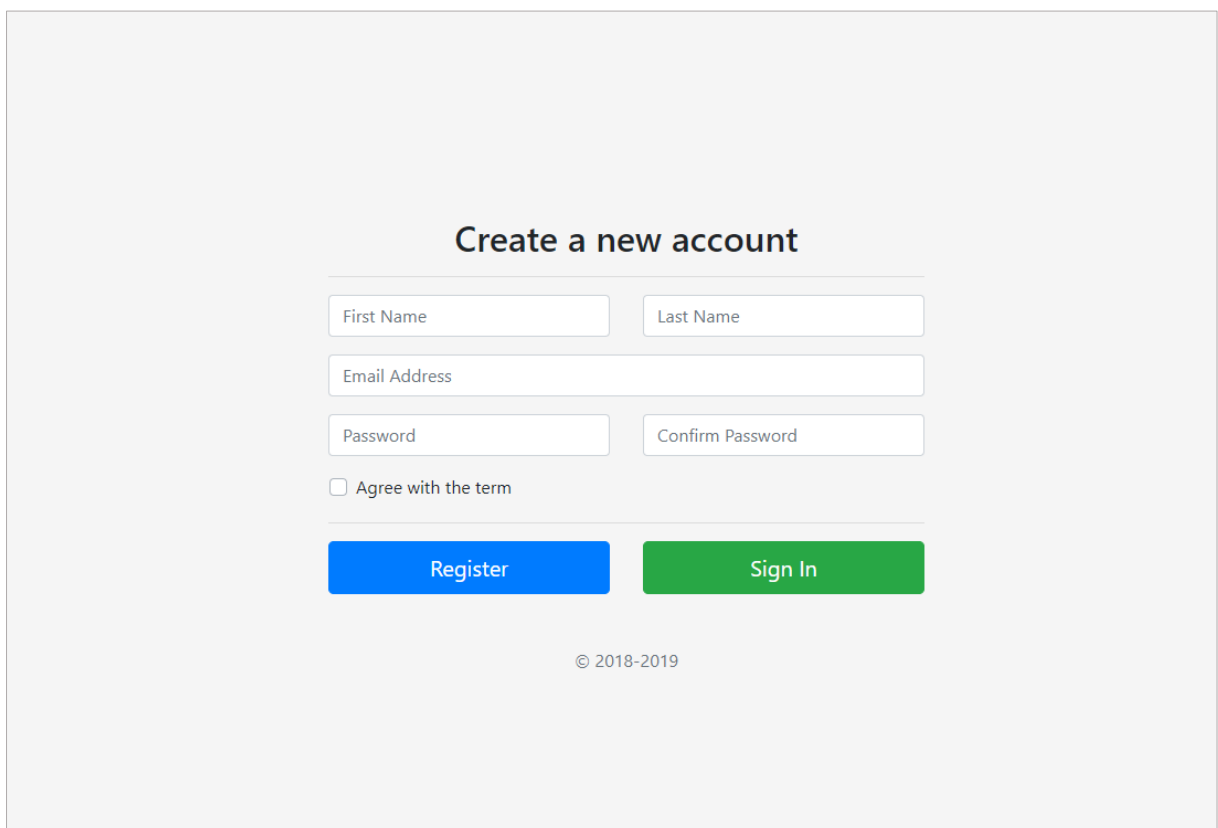
4.3.3.1. Đăng kí

Giao diện đăng kí được thiết kế như hình 4.3, chỉ có những session chưa đăng nhập mới có thể truy cập vào trang web này. Hàm checkLogin thực hiện việc kiểm tra session hiện tại, nếu đã đăng nhập thì sẽ chuyển đến trang web khác.

```
router.get('/register', checkLogin, function(req, res, next) {
  res.render('register', {errors : null});
});

function checkLogin(req, res, next) {
  if (!req.isAuthenticated()) next();
  else res.redirect('/device');
}
```

Để đăng kí thành công, người dùng cần cung cấp đầy đủ họ tên, email, mật khẩu và xác nhận mật khẩu, đồng thời phải đồng ý với điều khoản của ứng dụng. Nếu tất cả các trường được thoả mãn, khi nhấn Register dữ liệu sẽ được gửi lên server để xử lí. Nếu đã có tài khoản, có thể nhấn nút Sign In để chuyển về màn hình đăng nhập.



Create a new account

☐ Agree with the term

© 2018-2019

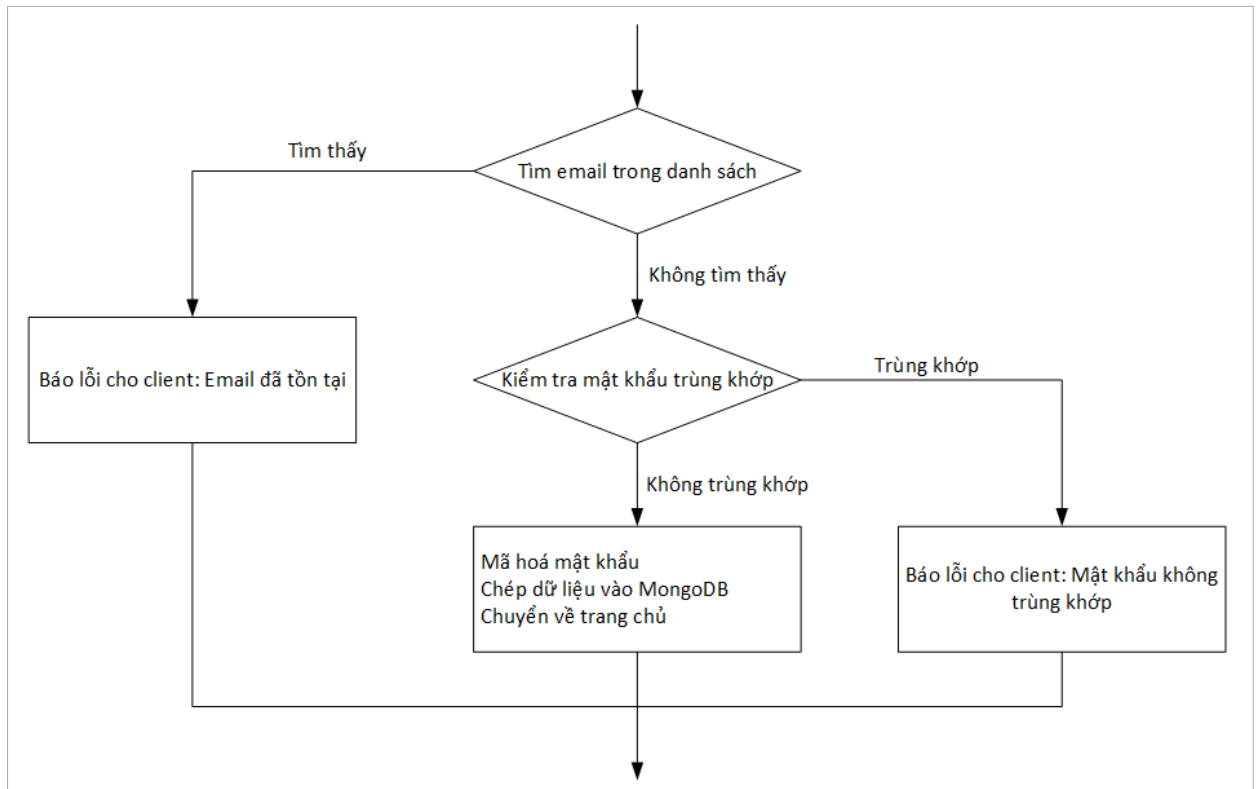
Hình 4.3. Giao diện đăng kí

Ở phía server, sau khi nhận được yêu cầu đăng kí, server sẽ kiểm tra trong danh sách người dùng hiện tại (lưu trữ trong MongoDB), nếu email trùng sẽ báo lỗi về client, nếu không sẽ kiểm tra tính trùng khớp của mật khẩu và mật khẩu xác nhận. Nếu mật khẩu không trùng sẽ báo lỗi để client biết. Ngược lại, server sẽ chép vào cơ sở dữ liệu thông tin về người dùng, trong đó mật khẩu sẽ được mã hoá bằng kĩ thuật băm trước khi lưu

trở, đồng thời chuyển người dùng về trang chủ.

```
//Mã hoá mật khẩu bằng kĩ thuật băm
bcrypt.genSalt(10, function(err, salt) {
  bcrypt.hash(newUser.password, salt, function(err, hash) {
    newUser.password = hash;
  });
});
```

Lưu đồ xử lí đăng kí được thể hiện qua hình 4.4.



Hình 4.4. Lưu đồ xử lí đăng kí

```
router.post('/register', function(req, res, next) {
  userModel.getUserByEmail(req.body.email, function (err,userFound) {
    var errors = [];
    if (err) console.log(err);
    if (userFound){
      var err = {};
      err.msg = "Existing email address";
      errors.push(err);
    }
    else {
      req.checkBody('password_confirmation','Password confirmation does not match').equals(req.body.password);
      errors = req.validationErrors();
    }
  }
  if (errors){
    res.render('register',{errors : errors});
  } else {
    var UserSchema = new userModel ({
      firstName : req.body.first_name,
```

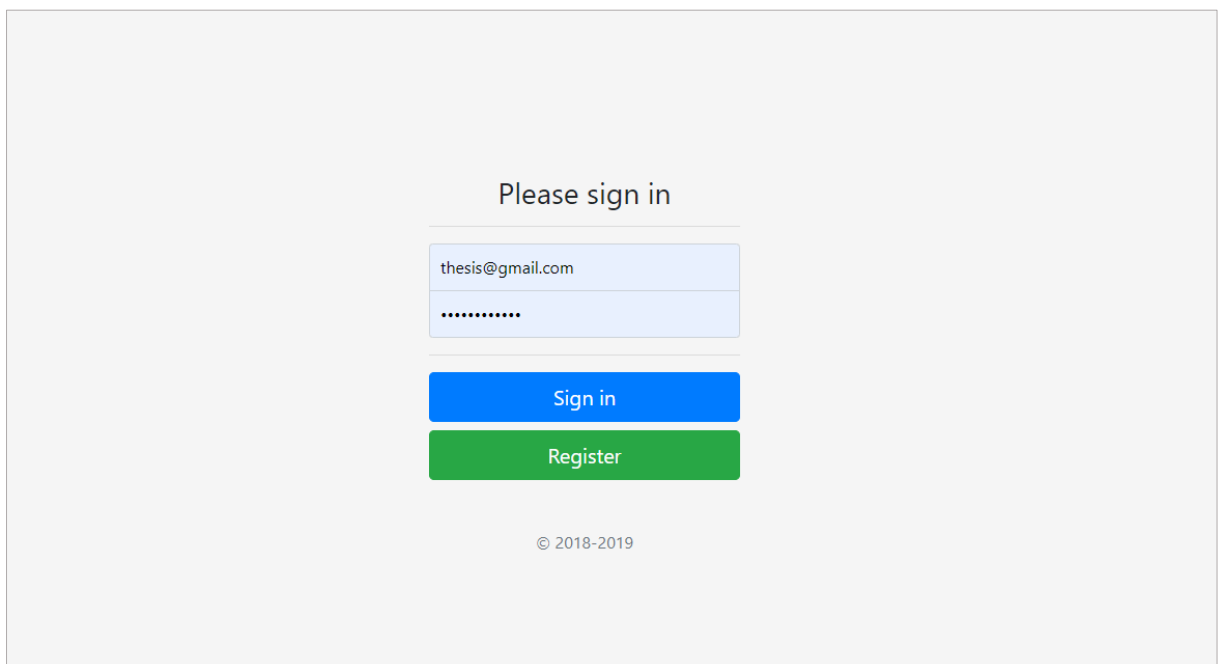
```

        lastName : req.body.last_name,
        email : req.body.email,
        password: req.body.password,
    });
    userModel.createUser(UserSchema);
    req.flash('success_msg', 'Successful registration');
    file.createUserDir(req.body.email);
    res.redirect('/');
}
});
});

```

4.3.3.2. Đăng nhập

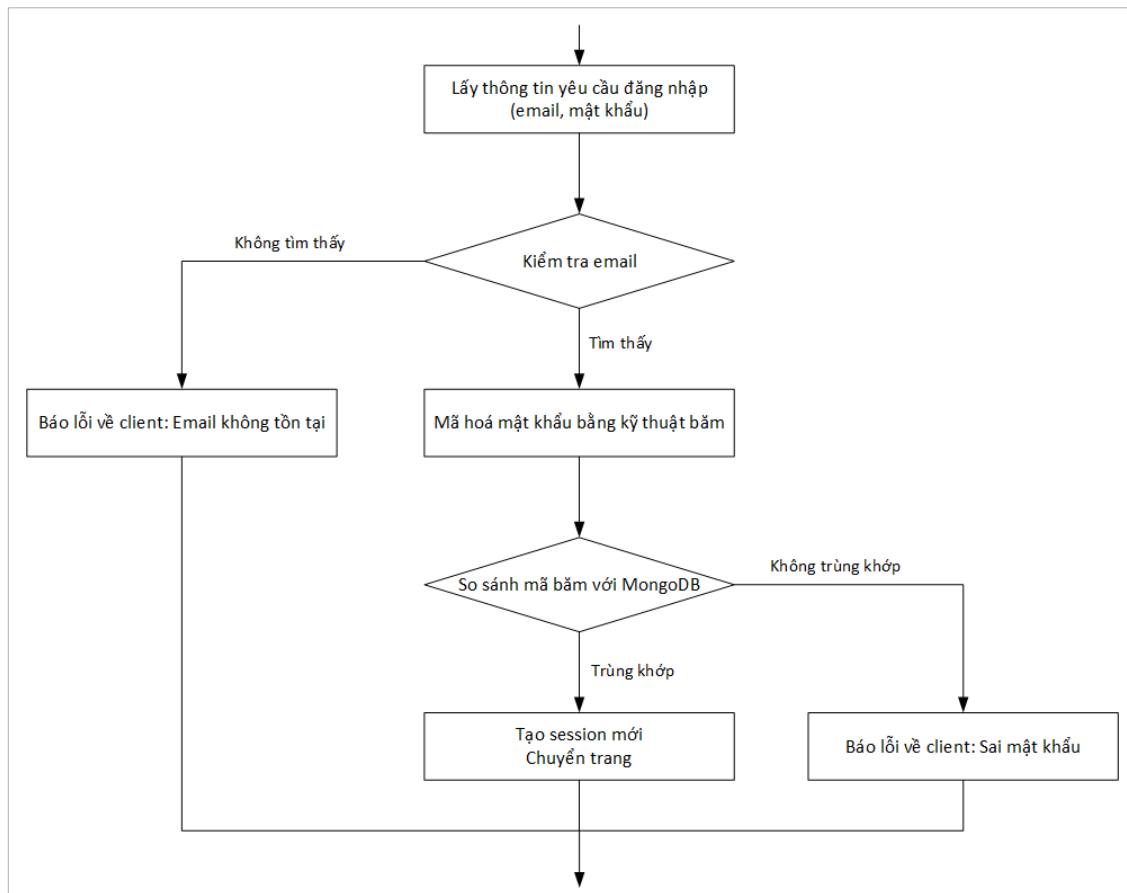
Giao diện đăng nhập được thiết kế như hình 4.5, đây là trang mặc định khi người dùng truy cập vào hệ thống.



The screenshot displays a login page with a light gray background. At the top, the text "Please sign in" is centered. Below it is a form with two input fields: the first contains the email "thesis@gmail.com" and the second contains masked characters ".....". Under the form are two buttons: a blue "Sign in" button and a green "Register" button. At the bottom center, the copyright notice "© 2018-2019" is visible.

Hình 4.5. Giao diện đăng nhập

Mặc định, passport.js dùng username và password để xác thực, cần phải tự cấu hình lại kiểu xác thực và cách thức xác thực theo lưu đồ hình 4.6. Với mỗi yêu cầu đăng nhập, server kiểm tra email đăng nhập so với dữ liệu trong hệ thống. Nếu không tìm thấy email sẽ báo lỗi về client. Nếu tìm được email sẽ tiến hành kiểm tra mật khẩu. Nếu mật khẩu trùng khớp sẽ tạo session mới cho user này, ngược lại sẽ báo lỗi về client. Để kiểm tra mật khẩu, server mã hoá bằng giải thuật băm đối với mật khẩu được gửi lên, sau đó so sánh với chuỗi mật khẩu mã hoá trong cơ sở dữ liệu.



Hình 4.6. Lưu đồ kiểm tra đăng nhập

```

//Cấu hình passport
passport.use(new LocalStrategy({
  usernameField: 'email',
  passwordField: 'password'
}),
function (username, password, done) {
  userModel.getUserByEmail(username, function (err, userFound) {
    if (err) console.log(err);
    if (!userFound)
      return done(null, false, { message: 'This email does not exist. Please
register before' });
    userModel.checkPassword(password, userFound.password, function (err, isMatch)
    {
      if (err) console.log(err);
      if (isMatch)
        return done(null, userFound);
      else return done(null, false, { message: 'Wrong password' });
    });
  });
});

//Hàm kiểm tra mật khẩu
function checkPassword (pass, hash, callback) {
  bcrypt.compare (pass, hash, function (err, isMatch) {
    if (err) console.log(err);
    callback(null, isMatch);
  });
};

```

Nếu đăng nhập thành công, server sẽ chuyển người dùng về trang quản lí gateway, ngược lại sẽ thông báo lỗi tại trang login.

```
router.post('/login',
  passport.authenticate('local', {
    successRedirect: '/device', //Trang quản lý gateway
    failureRedirect: '/login',  //Trang đăng nhập
    failureFlash: true
  })
);
```

Trong một ứng dụng web thông thường, thông tin đăng nhập để xác thực user chỉ được truyền một lần duy nhất khi yêu cầu đăng nhập. Nếu xác thực thành công, một session được thiết lập và duy trì thông qua một chuỗi các cookie được lưu trong trình duyệt. Những request sau đó sẽ không chứa thông tin đăng nhập, nhưng kèm theo một cookie đặc biệt (và duy nhất) để xác định session. Việc này vừa làm giảm kích thước của request, vừa bảo mật thông tin người dùng, đồng thời cũng thuận tiện vì người dùng không cần đăng nhập lại nhiều lần. Để thực hiện điều này, passport hỗ trợ hai hàm `serialize` và `deserialize` các session.

```
passport.serializeUser(function (email, done) {
  done(null, email.id);
});

passport.deserializeUser(function (id, done) {
  userModel.getUserById(id, function (err, email) {
    done(err, email);
  });
});
```

Ngoài ra, để tăng cường tính bảo mật, server chỉ cho phép những client đã đăng nhập mới được truy cập vào các dịch vụ mà nó cung cấp. Với mỗi request của client, server kiểm tra client đã được xác thực chưa, nếu thành công sẽ cung cấp dịch vụ, ngược lại sẽ trả về trang đăng nhập. Đoạn code bên dưới mô tả một route đến trang quản lí các gateway, kèm theo đó là hàm `checkAuthentication` để xác nhận client.

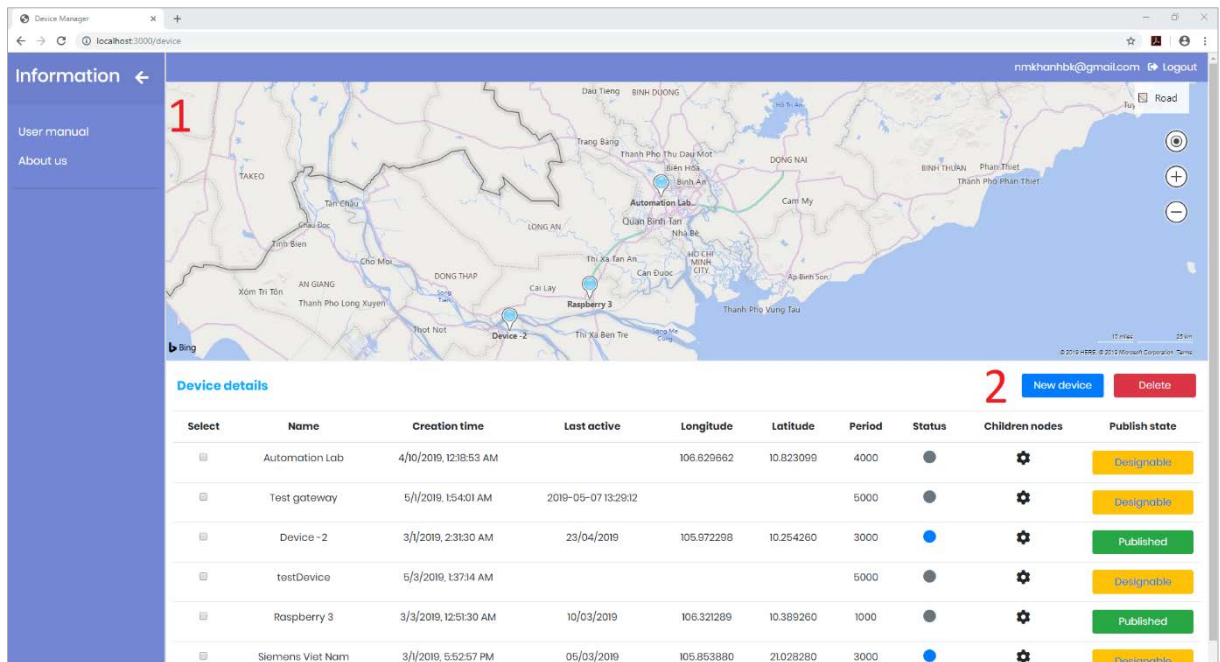
```
//Route đến trang quản lý gateway
router.get('/device', checkAuthntication, function (req, res, next) {
  res.render('device', { errors: null, user: req.user.email });
});

//Hàm kiểm tra client đã đăng nhập hay chưa
function checkAuthntication(req, res, next) {
  if (req.isAuthenticated()) next();
  else res.redirect('/login');
}
```

4.3.4. Quản lí gateway

Sau khi người dùng đăng nhập thành công, trang quản lí gateway trở thành trang chủ của hệ thống. Tại đây, người dùng có thể quan sát số lượng gateway đã cấu hình, vị trí địa lí cũng như trạng thái của từng gateway cụ thể. Giao diện quản lí gateway được thể hiện như hình 4.7. Trong hình 4.7, khu vực số 1 là bản đồ để quan sát phân bố của các gateway, hỗ trợ nhiều loại bản đồ khác nhau (vệ tinh, giao thông, hành chính); khu vực số 2 là các nút nhấn để thêm gateway mới hoặc xoá gateway hiện có; bảng “Device details” cung cấp thông tin chi tiết của từng gateway:

- Name: Tên gateway.
- Creation time: Thời gian tạo gateway.
- Last active: Thời gian gateway online lần cuối.
- Longitude và Latitude: Kinh độ và vĩ độ của gateway.
- Period: Chu kì gateway gửi dữ liệu về server, đơn vị milliseconds.
- Status: Trạng thái hiện tại của gateway (online hay offline).
- Children nodes: Nhấn nút sẽ hiện ra danh sách PLC và các biến mà gateway đang quản lí.
- Publish state: Trạng thái thiết kế hiện tại, nếu là “Designable” thì gateway chưa được thiết kế SCADA, nhấn vào liên kết đó để đi đến trang thiết kế; nếu là “Published” thì SCADA đã được thiết kế, nhấn vào liên kết để đi đến trang SCADA.

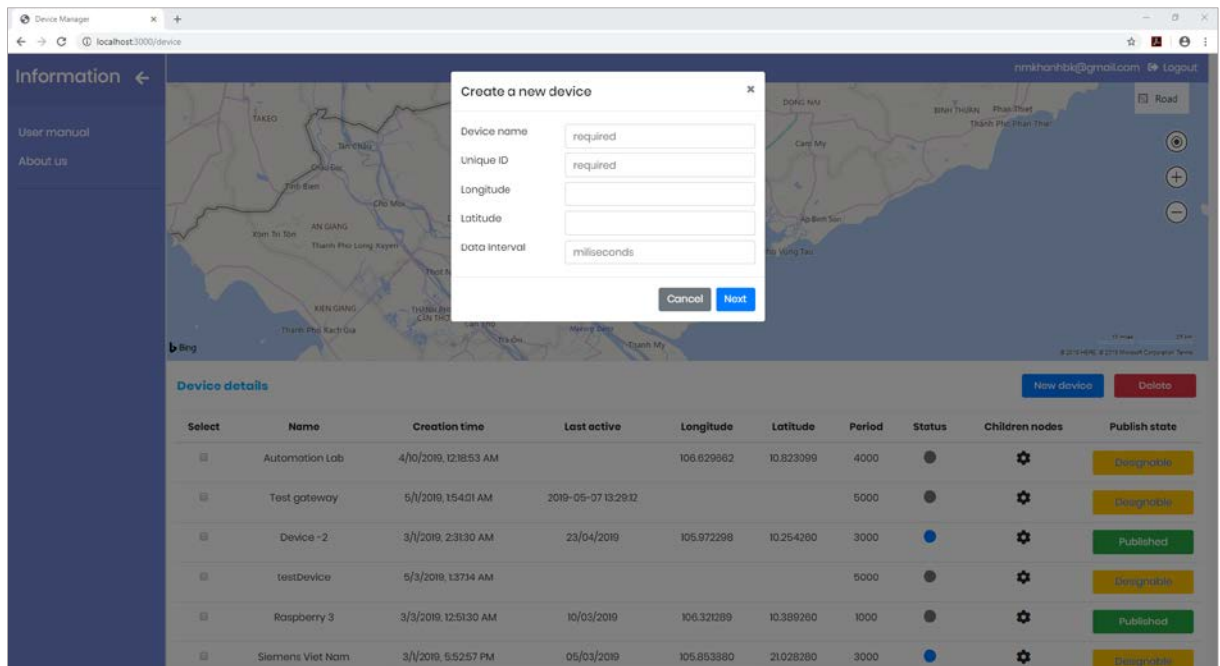


Hình 4.7. Giao diện quản lý gateway

4.3.4.1. Tạo gateway mới

Để tạo gateway mới, nhấn nút “New device” trên giao diện quản lý, modal new device xuất hiện với các tùy chọn như hình 4.8.

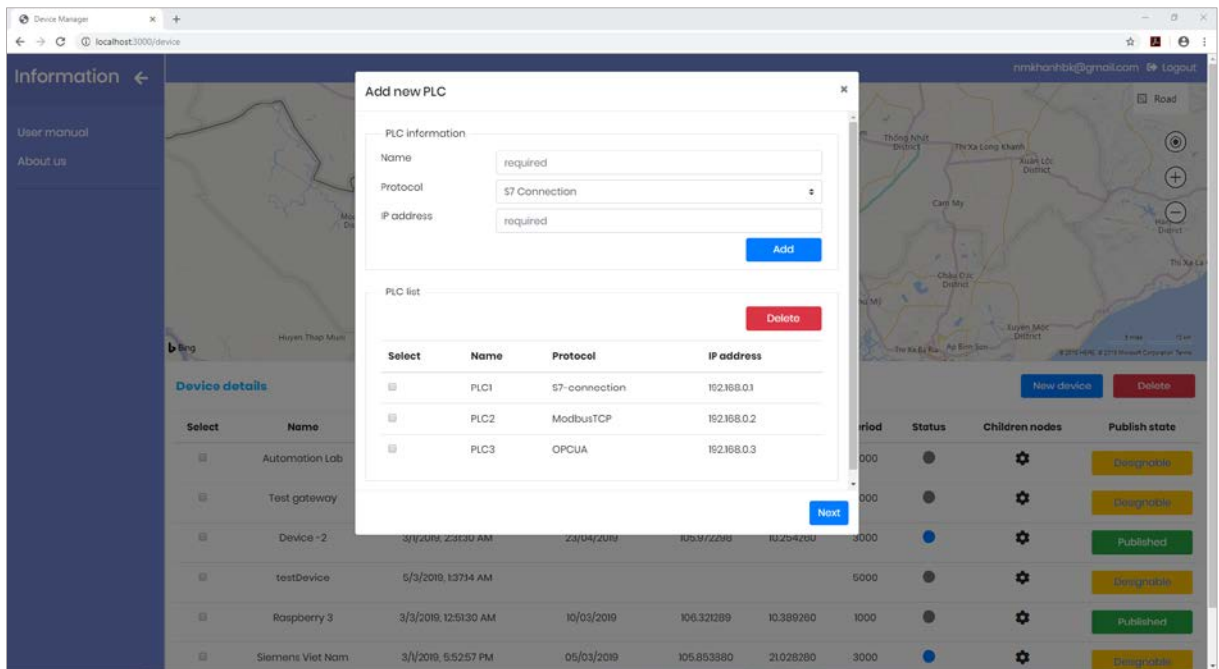
- Device name: Tên gateway.
- Unique ID: Mã số định danh của gateway, mã số này là duy nhất, dùng để phân biệt các gateway với nhau.
- Longitude, latitude: Kinh độ, vĩ độ của gateway (có thể bỏ qua các tham số này).
- Data interval: Chu kì gateway thu thập dữ liệu và gửi về server, đơn vị là milliseconds, nếu để trống thì giá trị mặc định là 5000ms (5s).



Hình 4.8. Giao diện tạo gateway mới

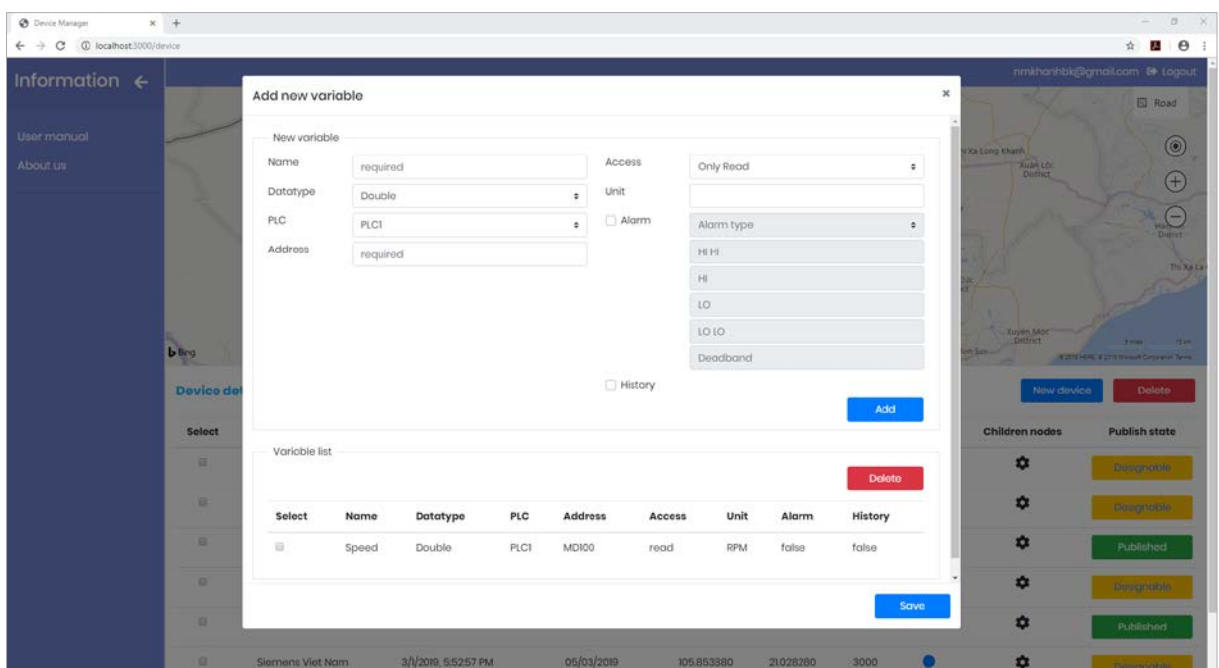
Sau khi nhập xong các tham số bắt buộc, nếu nhấn nút Next, thông số gateway sẽ được ghi nhận lại, modal new device tắt, đồng thời modal new PLC xuất hiện cho phép thêm các PLC vào gateway như hình 4.9. Các tham số cần phải xác định là:

- Name: Tên của PLC.
- Protocol: Giao thức để kết nối với PLC (S7-connection, OPC UA, Modbus TCP/IP).
- IP address: Địa chỉ IP của PLC.



Hình 4.9. Thêm PLC vào gateway

Nhấn nút Add để thêm PLC vào danh sách. Thực hiện tương tự cho các PLC khác. Có thể nhấn Delete để xóa PLC ra khỏi danh sách. Nhấn Next để hoàn thành, modal new PLC tắt, modal new variable xuất hiện cho phép cấu hình danh sách biến cho từng PLC, như hình 4.10.



Hình 4.10. Thêm danh sách biến cho từng PLC

Sau khi cấu hình xong danh sách biến, nhấn nút Save để hoàn tất. Toàn bộ cấu hình (từ lúc tạo gateway, thêm PLC, thêm biến) được gửi đến server bằng socket.IO theo cấu trúc JSON như sau:

```
{
  "deviceName": "demo",
  "deviceID": "123456",
  "longitude": "",
  "latitude": "",
  "period": "10000",
  "PLCs": [
    {
      "name": "PLC1",
      "protocol": "S7-connection",
      "ipAddress": "192.168.0.1",
      "variables": [
        {
          "name": "Speed",
          "dataType": "Double",
          "plc": "PLC1",
          "address": "MD100",
          "access": "read",
          "unit": "RPM",
          "isAlarm": false,
          "alarmType": null,
          "parameters": {
            "lolo": null,
            "lo": null,
            "hi": null,
            "hihi": null,
            "deadband": null
          },
          "isHistory": false
        }
      ]
    }
  ],
  "user": "nmkhanhbk@gmail.com",
  "creationTime": "5/24/2019, 2:11:40 PM",
  "lastActive": "",
  "published": false,
  "status": false
}
```

Server nhận được yêu cầu sẽ thêm một thuộc tính mới vào JSON object nhận được là **fileName** - tên của file cấu hình được lưu trên server. Thuộc tính này giúp server nhanh chóng truy xuất đến file cấu hình khi có yêu cầu. Tên file được đặt theo cú pháp: **“deviceConfig_” + device ID + “.json”** và được lưu ở đường dẫn: **Database/email người dùng/Config**. Bên cạnh đó, server sẽ gửi file cấu hình này cho gateway thông qua giao thức MQTT để gateway cấu hình.

```

socket.on('deviceConfig', function (data) {
  console.log(data);
  if (data) {
    try {
      var receivedObject = JSON.parse(data);
      if (receivedObject) {
        receivedObject.fileName = 'deviceConfig_' + receivedObject.deviceID +
        '.json';
        receivedObjectJSON = JSON.stringify(receivedObject, null, 4);
        var filePath = path.resolve(databasePath, receivedObject.user, 'Config',
        'deviceConfig_' + receivedObject.deviceID + '.json');

        fs.writeFile(filePath, receivedObjectJSON, function (err) {
          if (err) console.log(err);
          else console.log('Succeed');
        });

        //Send configuration to gateway via MQTT
        mqttClient.publish('/') + receivedObject.deviceID + '/config',
        receivedObjectJSON, retainOption);
      } catch (error) {
        console.log(error);
      }
    }
  }
});

```

4.3.4.2. Xoá gateway

Để xoá một hoặc nhiều gateway, nhấp chuột lên gateway cần xoá rồi nhấn nút Delete như hình 4.11.

Device details									
									New device Delete
Select	Name	Creation time	Last active	Longitude	Latitude	Period	Status	Children nodes	Publish state
<input type="checkbox"/>	Automation Lab	4/10/2019, 12:18:53 AM		106.629862	10.823099	4000	●	⚙	Designable
<input type="checkbox"/>	Test gateway	5/1/2019, 15:4:01 AM	2019-05-07 13:29:12			5000	●	⚙	Designable
<input checked="" type="checkbox"/>	demo	5/24/2019, 2:11:40 PM				10000	●	⚙	Designable
<input type="checkbox"/>	Device - 2	3/1/2019, 2:31:30 AM	23/04/2019	105.972296	10.254260	3000	●	⚙	Published
<input type="checkbox"/>	testDevice	5/3/2019, 1:37:14 AM				5000	●	⚙	Designable
<input type="checkbox"/>	Raspberry 3	3/3/2019, 12:51:30 AM	10/03/2019	106.321289	10.369260	1000	●	⚙	Published

Hình 4.11. Xoá gateway

Khi nhấn nút Delete, client sẽ tìm tên gateway trong bảng Device details. Sau đó client quét tất cả các gateway hiện có trong danh sách, nếu tìm thấy tên gateway cần xoá thì gửi về server qua socket.IO tên file cấu hình gateway này. Server nhận được sẽ gửi yêu cầu xoá file về gateway tương ứng, đồng thời kiểm tra và xoá file cấu hình trong thư mục Config. Khi gateway nhận được yêu cầu xoá file sẽ xoá mọi cấu hình cũ, trở về trạng thái đợi cấu hình mới.

```

//Xử lý phía client
var delRow = $(this).parents("tr");
var delDevice = $(this).parents("tr").find('td')[1].innerHTML;
for (var device of rcvDeviceObject) {
    if (device.deviceName == delDevice) {
        socket.on('deleteSuccess', function (data) {
            delRow.remove();
            rcvDeviceObject.splice( rcvDeviceObject.indexOf(device), 1);
        });
        socket.emit('deleteDevice', device.user + '/Config/' + device.fileName);
        break;
    }
}

//Xử lý phía server
socket.on('deleteDevice', function (filePath) {
    var fileBase = path.parse(filePath).base;
    var deviceID = fileBase.replace('deviceConfig_', '').replace('.json', '');
    var mqttResetTopic = '/' + deviceID + '/reset';
    mqttClient.publish(mqttResetTopic, JSON.stringify({ CMD: true }, null, 4),
        retainOption);

    fs.unlink(path.resolve(databasePath, filePath), function (err) {
        if (err) console.log(err)
        else socket.emit('deleteSuccess', 1);
    });
});

```

4.3.4.3. Yêu cầu danh sách cấu hình

Ngay khi trang quản lý gateway được tải xong, client gửi yêu cầu danh sách gateway của người dùng về server. Server nhận được yêu cầu sẽ trích xuất email của người dùng, sau đó tìm tất cả các file cấu hình gateway hiện có của người dùng đó, đọc nội dung từng file rồi gửi về client.

```

//Xử lý phía client
socket.on('/' + _user + '/resDeviceConfig', function (data) {
    rcvDeviceObject = data;
    loadDeviceTable(rcvDeviceObject);
    socketOnStatus(socket); //Subscribe status topic
    //Choose first td when click on tr
    $('table tbody tr').each(function () {
        $(this).click(function () {
            var checkbox = $(this).find('input');
            checkbox.prop('checked', !checkbox.prop('checked'));
        });
    });
});


//Xử lý phía server
socket.on('/reqDeviceConfig', function (user) {
    var arrConfig = [];
    var userFolder = path.resolve(databasePath, user, 'Config');
    async.series([
        function (callback) {
            var fileList = fs.readdirSync(userFolder);
            if (fileList.length > 0) {

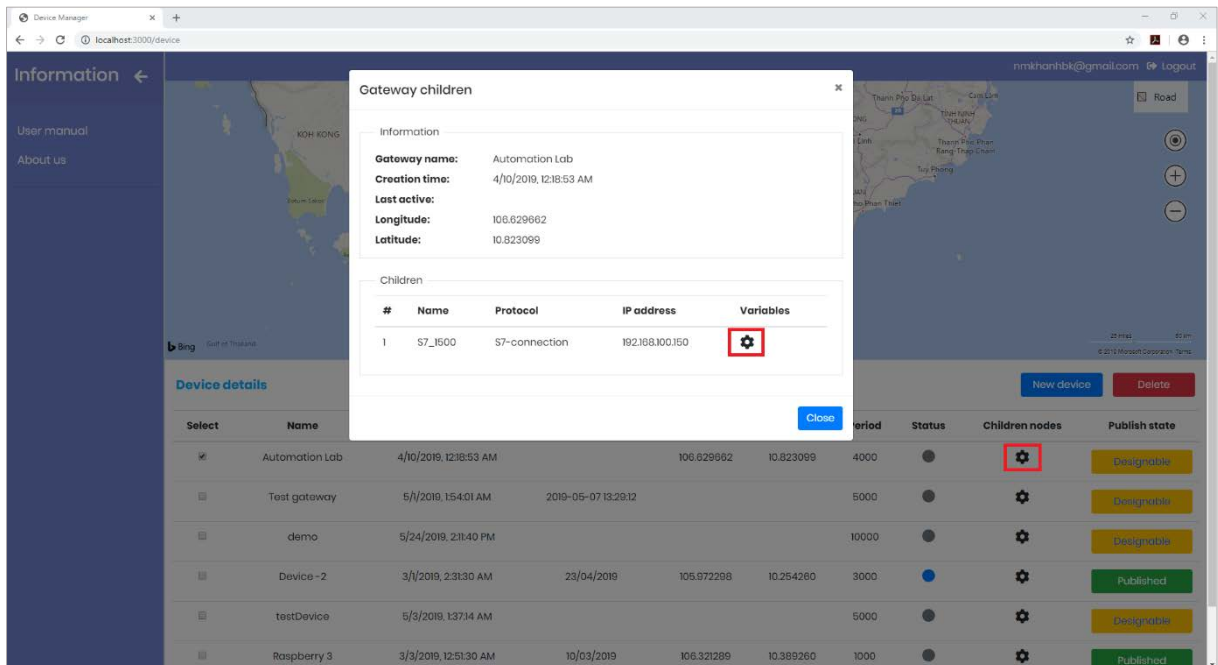
```

```


        fileList.forEach(function (file) {
            if (path.extname(file) == '.json')
                arrConfig.push(JSON.parse(fs.readFileSync(path.resolve(userFolder,
file))))
        });
    }
    callback();
}
], function (err) {
    if (err) console.log(err);
    else socket.emit('/') + user + '/resDeviceConfig', arrConfig);
});
});

```













Người dùng có thể xem cấu hình của từng gateway bằng cách nhấp vào biểu tượng  tại cột Children nodes như hình 4.12. Ở Gateway children modal, nhấp vào biểu tượng tương tự tại cột Variables để xem danh sách biến của từng PLC (hình 4.13).



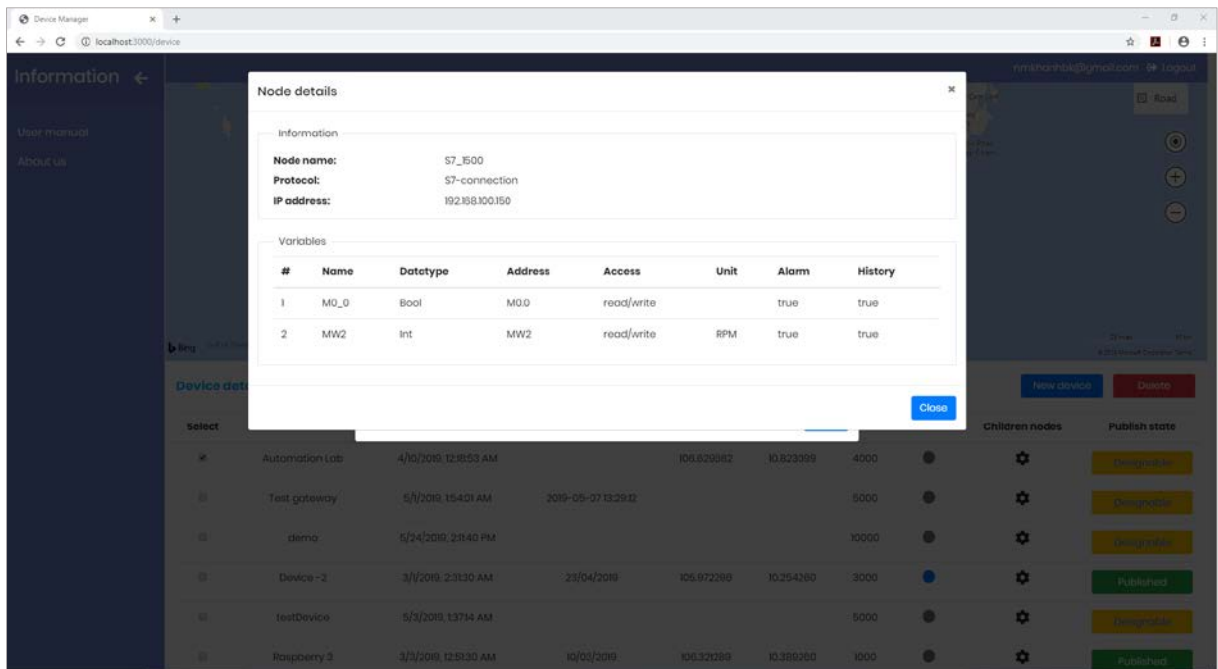
The screenshot shows the 'Device Manager' web application. A modal window titled 'Gateway children' is open, displaying details for the 'Automation Lab' gateway. The modal includes a table of children nodes with columns: #, Name, Protocol, IP address, and Variables. A red box highlights the gear icon in the 'Variables' column of the first child node.

#	Name	Protocol	IP address	Variables
1	S7_1500	S7-connection	192.168.100.150	

Below the modal, a table lists various devices with columns: Select, Name, Creation time, Last active, Longitude, Latitude, Period, Status, Children nodes, and Publish state. A red box highlights the gear icon in the 'Children nodes' column of the 'Automation Lab' row.

Select	Name	Creation time	Last active	Longitude	Latitude	Period	Status	Children nodes	Publish state
<input checked="" type="checkbox"/>	Automation Lab	4/10/2019, 12:38:53 AM		106.629962	10.823099	4000			Designable
<input type="checkbox"/>	Test gateway	5/1/2019, 15:40:1 AM	2019-05-07 13:29:12			5000			Designable
<input type="checkbox"/>	demo	5/24/2019, 2:11:40 PM				10000			Designable
<input type="checkbox"/>	Device-2	3/1/2019, 2:31:30 AM	23/04/2019	106.672298	10.254260	3000			Published
<input type="checkbox"/>	testDevice	5/3/2019, 1:37:14 AM				5000			Designable
<input type="checkbox"/>	Raspberry 3	3/3/2019, 12:51:30 AM	10/03/2019	106.321289	10.389260	1000			Published

Hình 4.12. Children node – Danh sách PLC



Hình 4.13. Node details – Danh sách biến

4.3.4.4. Trạng thái gateway

Gateway có hai loại trạng thái là online/ offline và designable/ published như hình 4.14.

Device details										New device	Delete
Select	Name	Creation time	Last active	Longitude	Latitude	Period	Status	Children nodes	Publish state		
<input type="checkbox"/>	Automation Lab	4/10/2019, 12:18:53 AM		106.629662	10.823099	4000	●	⚙	Designable		
<input type="checkbox"/>	Test gateway	5/1/2019, 15:40:1 AM	2019-05-07 13:29:12			5000	●	⚙	Designable		
<input type="checkbox"/>	demo	5/24/2019, 2:14:40 PM				10000	●	⚙	Designable		
<input type="checkbox"/>	Device -2	3/1/2019, 2:31:30 AM	23/04/2019	105.972298	10.254260	3000	●	⚙	Published		
<input type="checkbox"/>	testDevice	5/3/2019, 1:37:14 AM				5000	●	⚙	Designable		

Hình 4.14. Trạng thái gateway

Online/ offline cho biết gateway hiện có kết nối với server hay không, được thể hiện qua cột Status trong bảng Device details (online tương ứng màu xanh, offline tương ứng màu xám). Mỗi khi gateway online, nó sẽ publish vào topic **/device ID/status** với nội dung là “true”. Khi offline, do đã cấu hình will topic từ

trước, nên broker sẽ thay mặt gateway publish vào will topic (**/device ID/status**) với nội dung là “false”. Cả hai lần publish này đều có cờ retain là “true”, do đó trạng thái hiện tại của gateway chính là tin nhắn cuối cùng mà broker đang nắm giữ. Server subscribe topic **/+/status** để nhận trạng thái của tất cả gateway. Khi một gateway thay đổi trạng thái, server ngay lập tức chỉnh sửa thuộc tính status trong file cấu hình của gateway đó, đồng thời gửi thông báo về client. Nhờ vậy client luôn biết trạng thái của từng gateway một cách chính xác.

Designable/ published cho biết gateway hiện đã có một ứng dụng SCADA hay chưa, được thể hiện qua cột Publish state trong bảng Device details. Nếu giá trị là Designable, gateway chưa có ứng dụng SCADA nào, có thể nhấp vào nút Designable để bắt đầu thiết kế SCADA cho gateway. Nếu giá trị là Published, gateway đã có ứng dụng SCADA, nhấp vào để đi đến ứng dụng đó. Liên kết của ứng dụng SCADA được lưu trong thuộc tính “link” ở file cấu hình gateway.

4.3.5. Chức năng thiết kế giao diện

4.3.5.1. Giới thiệu

Chức năng thiết kế giao diện là trọng tâm của web-based SCADA, cho phép người dùng tự thiết kế giao diện theo ý muốn. Các yêu cầu đối với chức năng này là:

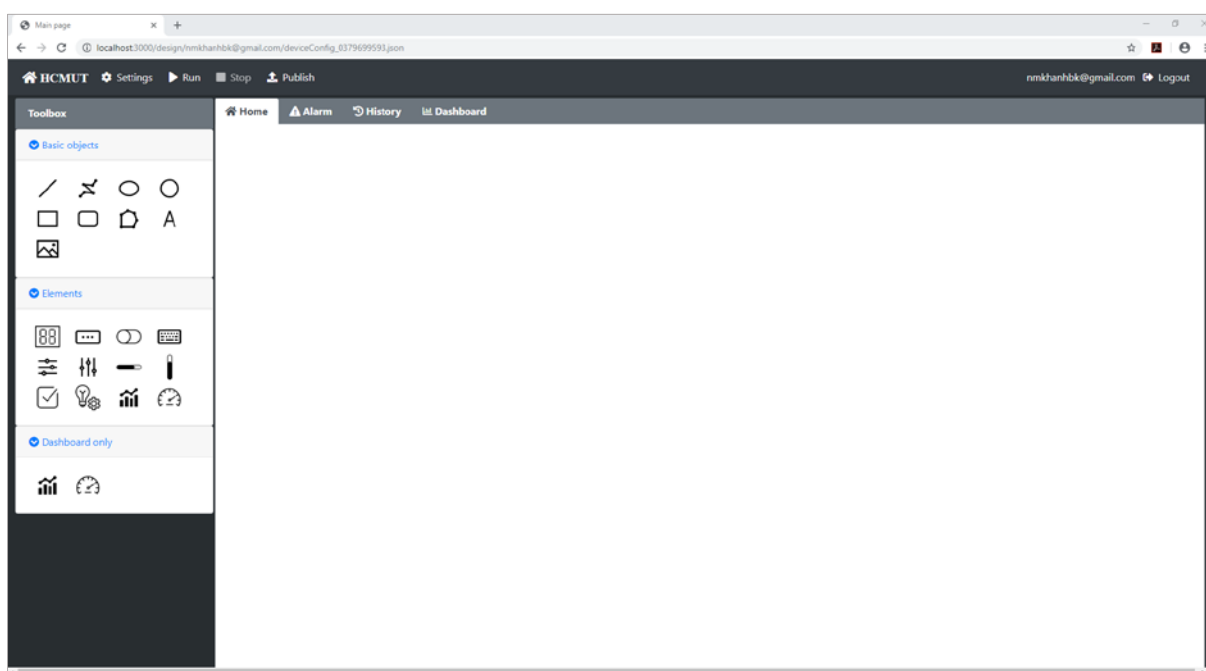
- Hỗ trợ nhiều loại đối tượng cơ bản.
- Tạo đối tượng bằng nhấp chuột hoặc kéo thả chuột.
- Di chuyển các đối tượng linh hoạt.
- Tùy chỉnh nhiều thuộc tính của đối tượng.

Hiện tại luận văn đã hỗ trợ 21 loại đối tượng SCADA, được chia thành 4 nhóm:

- Nhóm basic: gồm các đối tượng hình học cơ bản như line, oval, circle, rectangle, round rectangle, polyline, polygon.
- Nhóm display: gồm text, image, display value, symbol set, progress bar, vertical progress bar.

- Nhóm control: gồm các đối tượng để điều khiển như button, switch, input, slider, vertical slider, checkbox.
- Nhóm advance: gồm các đối tượng phức tạp như chart, gauge.

Giao diện thiết kế như hình 4.15 với thanh Toolbox ở bên trái, khu vực làm việc nằm ở bên phải, thanh công cụ ở trên cùng. Toolbox chứa các đối tượng mà hệ thống hỗ trợ. Khu vực làm việc là nơi thiết kế các ứng dụng SCADA, hỗ trợ bốn cửa sổ: Home (SCADA), alarm (cảnh báo), history (lịch sử) và dashboard (chart, gauge). Thanh công cụ cung cấp các chức năng bản như Settings (chỉnh màu nền các trang), Run (vận hành thử), Stop (ngừng vận hành thử), Publish (xuất bản trang thiết kế). Để “vẽ” một đối tượng, người dùng cần nhấp chuột vào đối tượng bên Toolbox, sau đó đưa chuột sang khu vực làm việc và nhấp chuột, đối tượng sẽ được tạo ngay tại vị trí đó, nhấn ESC để thoát khỏi chế độ vẽ. Để chỉnh sửa thuộc tính của đối tượng, nhấp đúp chuột lên đối tượng, modal thuộc tính xuất hiện, thay đổi các giá trị thuộc tính trong modal này.



Hình 4.15. Giao diện thiết kế SCADA

Để hỗ trợ nhiều loại đối tượng cần phải sử dụng thêm một số thư viện mà Javascript không có sẵn, chúng được dùng ở phía client để tương tác với người dùng:


- JQuery UI draggable: Hỗ trợ kéo thả các đối tượng HTML (không hỗ trợ svg).
- Plain draggable: Hỗ trợ kéo thả các đối tượng svg.
- Bootstrap form helper: Chỉnh sửa font, font size của đối tượng.
- Bootstrap slider: Vertical slider, HTML truyền thống chỉ hỗ trợ slider theo chiều ngang.
- Context menu: Tạo menu khi nhấn chuột phải
- Svg.js và Svg.draw.js: Vẽ các đối tượng cơ bản như line, rectangle, circle,...
- Moment.js, Chart.js và hammer.js: Vẽ đồ thị.
- Raphael.js và Justgage.js: Vẽ gauge.
- RGBA color picker: Chọn màu RGBA, dùng cho gauge.

Để quản lí một lượng lớn các đối tượng, ta sử dụng hai mảng là shapes và elementHTML. Mảng shapes chứa tất cả object Javascript của đối tượng, với object này ta có thể chỉnh sửa các thuộc tính của đối tượng. Cũng vì vậy mà mảng này có cấu trúc phức tạp, kích thước lớn. Mảng elementHTML chứa các thuộc tính của từng đối tượng, mỗi phần tử là một object JSON nên kích thước nhỏ. Trong quá trình xuất bản trang web, truyền mảng elementHTML giúp tiết kiệm băng thông và hạn chế thấp nhất các rủi ro hơn mảng shapes. Sau khi truyền xong, server sẽ căn cứ vào các thuộc tính nhận được mà khởi tạo lại đối tượng SCADA.

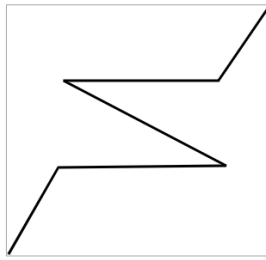
4.3.5.2. Nhóm basic

Nhóm này gồm các đối tượng hình học cơ bản như line, oval, circle, rectangle, round rectangle, polyline và polygon. Tất cả các đối tượng trên đều có kiểu SVG. Một lưu ý với các đối tượng SVG là không thể kéo thả bằng JQuery UI draggable thông thường, do đó luận văn sử dụng thêm thư viện Plain draggable để hỗ trợ kéo thả các đối tượng này.

Polyline

Polyline là các đoạn thẳng nối tiếp nhau, điểm cuối của đoạn thẳng này là điểm bắt đầu của đoạn thẳng kế tiếp. Để vẽ đối tượng này, nhấp chuột vào biểu tượng  trên

Toolbox, sau đó đưa chuột ra màn hình làm việc rồi lần lượt nhấp chuột để tạo các điểm nối. Nhấn Enter để kết thúc quá trình vẽ.



Hình 4.16. Polyline

Trước khi bắt đầu vẽ một đối tượng, ta phải đảm bảo không có đối tượng nào đang được vẽ. Hàm `stopDraw` đảm nhận nhiệm vụ thông báo cho client biết ngừng nhận sự kiện vẽ của tất cả các đối tượng.

```
var stopDraw = function (addContext) {  
    draw.off();  
    $('#mainPage1').off('mousedown', imageMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', textMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', displayValueMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', buttonMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', switchMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', inputMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', checkboxMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', sliderMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', verticalSliderMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', processbarMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', verticalProcessbarMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', symbolsetMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', chartMouseDownEventHandler);  
    $('#mainPage1').off('mousedown', gaugeMouseDownEventHandler);  
    $('#dashboard').off('mousedown', chartDashboardMouseDownEventHandler);  
    $('#dashboard').off('mousedown', gaugeDashboardMouseDownEventHandler);  
    if (addContext) addContextMenu();  
}
```

Khi hàm `stopDraw` thực hiện xong, bắt đầu vẽ polyline và đăng kí sự kiện `drawstart` để biết khi nào người dùng nhấp chuột vẽ, đồng thời đăng kí sự kiện `keydown` để biết khi nào người dùng nhấn phím Enter, báo hiệu đã vẽ xong. Tại sự kiện `keydown`, client thông báo kết thúc quá trình vẽ bằng sự kiện `done`, điều này khởi tạo sự kiện `drawstop`. Tại hàm `drawstop`, huỷ lắng nghe sự kiện `keydown` phím Enter, quá trình vẽ hoàn tất.

```

//Sự kiện bắt đầu vẽ
shapes[index].on('drawstart', function (e) {
    //Subscribe keydown event to detect ENTER key
    document.addEventListener('keydown', keyEnterDownHandler);

    //Các hàm khác được viết trong sự kiện này
});

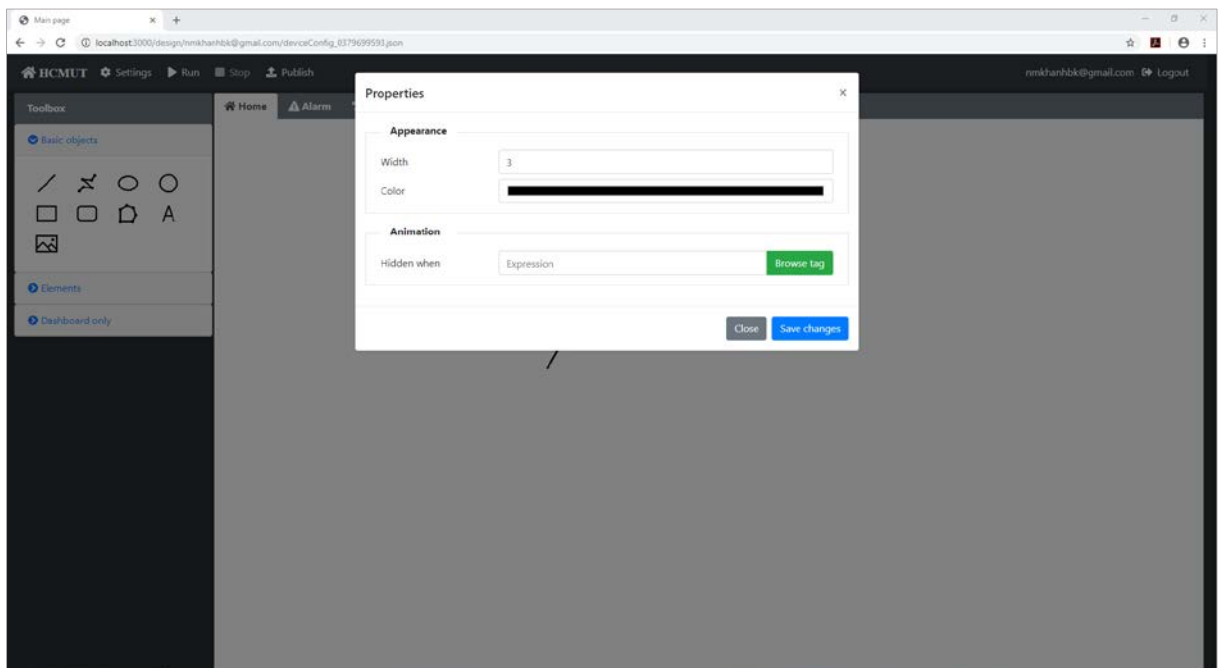
//Sự kiện vẽ kết thúc
shapes[index].on('drawstop', function () {
    //Remove enter key event
    document.removeEventListener('keydown', keyEnterDownHandler);
});

//Sự kiện keydown phím Enter
function keyEnterDownHandler(e) {
    if (e.keyCode == 13) {
        shapes[index].draw('done');
        shapes[index].off('drawstart');
        index++;
        stopDraw();
    }
}

```

Để thay đổi thuộc tính của đối tượng, nhấp đúp chuột vào đối tượng, modal tương ứng với đối tượng hiện ra như hình 4.17.

- Width: Độ rộng nét vẽ.
- Color: Màu sắc nét vẽ.
- Hidden when: Điều kiện để ẩn đối tượng. Nhấn nút browse tags để lấy danh sách biến và viết điều kiện theo cú pháp Javascript.



Hình 4.17. Modal thuộc tính của polyline

Để bắt được sự kiện nhấp đúp chuột, cần đăng kí sự kiện này cho mỗi polyline khi khởi tạo. Hàm này được viết bên trong sự kiện drawstart, giúp chỉnh sửa các thuộc tính polyline theo tham số người dùng nhập.

```
shapes[index].on('dblclick', function (mouseEvent) {
    $('#polylineModal').one('show.bs.modal', function (showEvent) {
        var element;
        for (var item of shapes) {
            try {
                if (item.node.id == mouseEvent.target.id) {
                    element = item;
                    break;
                }
            } catch {
                console.log('Item not found');
            }
        }
        if (element) {
            var elemWidth = element.attr('stroke-width');
            var elemColor = element.attr('stroke');
            var itemModal = $('#polylineModal')[0];

            itemModal.querySelector('#inputWidth').value = elemWidth;
            itemModal.querySelector('#inputColor').value = elemColor;

            if (mouseEvent.target.hiddenWhen) {
                itemModal.querySelector('.inputHiddenWhen').value =
mouseEvent.target.hiddenWhen;
            }
            else {
                itemModal.querySelector('.inputHiddenWhen').value = '';
            }

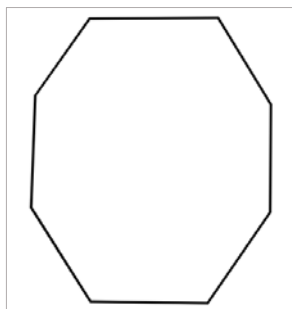
            $('.saveChangeButton').on('click', function (event) {
                element.attr({
                    'stroke-width': itemModal.querySelector('#inputWidth').value,
                    'stroke': itemModal.querySelector('#inputColor').value,
                });

                mouseEvent.target.hiddenWhen =
itemModal.querySelector('.inputHiddenWhen').value;

                var _foundIndex = findElementHTMLById(mouseEvent.target.id);
                if (_foundIndex != -1) elementHTML[_foundIndex].properties[0].value =
mouseEvent.target.hiddenWhen;
            });
            $('.btnHiddenWhen').on('click', function (onConditionClickEvent) {
                $('#tagModal').one('hide.bs.modal', function (modalHideEvent) {
                    if
($($('#tagModal')[0].querySelector('input[name="rdoChoseTag"]:checked')) {
                        itemModal.querySelector('.inputHiddenWhen').value +=
$($('#tagModal')[0].querySelector('input[name="rdoChoseTag"]:checked').value;
                    }
                });
            });
        }
    });
    $('#polylineModal').one('hide.bs.modal', function (hideEvent) {
        $('.saveChangeButton').off('click');
        $('.btnHiddenWhen').off('click');
    });
    $('#polylineModal').modal();
});
```

Polygon

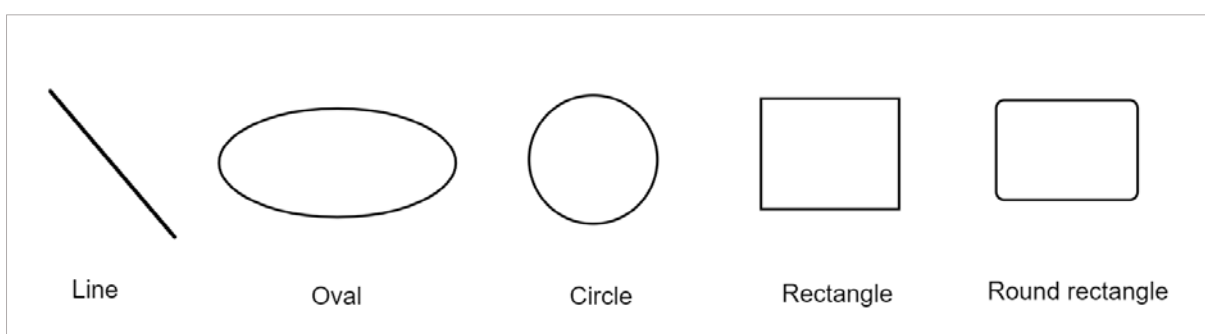
Polygon là đa giác có nhiều cạnh như hình 4.18. Cách sử dụng và kỹ thuật lập trình hoàn toàn tương tự như polyline, chỉ thay hàm polyline() bằng hàm polygon().



Hình 4.18. Polygon

Line, oval, circle, rectangle, round rectangle

Các đối tượng này có cùng cách xử lý code nên được gom nhóm cùng nhau. Để vẽ một đối tượng, nhấp chuột vào biểu tượng tương ứng, đưa chuột ra màn hình làm việc rồi tiến hành kéo thả. Có thể vẽ nhiều đối tượng liên tục bằng cách lặp lại thao tác kéo thả như trên. Để kết thúc quá trình vẽ, nhấn phím ESC.



Hình 4.19. Line, oval, circle, rectangle, round rectangle

Giống như các đối tượng khác, trước khi vẽ mới ta cần đảm bảo không có đối tượng nào đang được vẽ bằng hàm stopDraw(). Client lắng nghe sự kiện mousedown để phát hiện thời điểm bắt đầu vẽ. Tùy thuộc và chức năng được chọn mà gọi hàm vẽ tương ứng.

```
draw.on('mousedown', function (event) {  
  switch (shape) {  
    case 'line': {  
      shapes[index] = draw.line().attr(defaultLineOption);  
      modalId = '#lineModal';  
      break;  
    }  
    case 'ellipse': {  
      shapes[index] = draw.ellipse().attr(defaultOption);  
    }  
  }  
});
```

```

        modalId = '#ellipseModal';
        break;
    }
    case 'circle': {
        shapes[index] = draw.circle(10).attr(defaultOption);
        modalId = '#circleModal';
        break;
    }
    case 'rect': {
        shapes[index] = draw.rect().attr(defaultOption);
        modalId = '#rectModal';
        break;
    }
    case 'roundRect': {
        shapes[index] = draw.rect().attr(defaultOption);
        shapes[index].radius(10);
        modalId = '#roundRectModal';
        break;
    }
}
shapes[index].draw(event);
}, false);

```

Quá trình vẽ kết thúc khi người dùng nhả chuột (mouse up), lúc này phải khởi tạo Plain draggable để có thể di chuyển đối tượng.

```

draw.on('mouseup', function (event) {
    shapes[index].draw(event);
    var element = document.getElementById(shapes[index].node.id);
    draggable = new PlainDraggable(element, { leftTop: true });
    draggable.autoScroll = true;
    draggable.containment = document.getElementById('mainPage1');
    draggableObjects.push(draggable);
})

```

Một điểm cần lưu ý với các đối tượng loại này là khi thay đổi vị trí đối tượng, cần chỉnh sửa thuộc tính translate về tọa độ (0,0) để đối tượng nhận vị trí mới. Nếu không chỉnh lại tham số này, tọa độ của lần di chuyển tiếp theo sẽ sai.

```

element.attr({
    'stroke-width': itemModal.querySelector('#inputStrokeWidth').value,
    'stroke-linecap': itemModal.querySelector('#inputLinecap').value,
    'stroke': itemModal.querySelector('#inputColor').value,
    'x1': itemModal.querySelector('#inputX1').value,
    'y1': itemModal.querySelector('#inputY1').value,
    'x2': itemModal.querySelector('#inputX2').value,
    'y2': itemModal.querySelector('#inputY2').value,
    'transform': 'translate(0,0)',
});

```

Ngoài ra, svg của HTML còn có một đặc điểm khá khó chịu, là khi di chuyển đến vị trí mới sẽ bị cố định tại đó, nghĩa là không thể thay đổi vị trí của đối tượng nữa! Để giải quyết vấn đề đó, ta dùng một mảng để quản lý các object plain draggable. Khi thay đổi vị trí đối tượng, ta xoá object plain draggable cũ, khởi tạo lại object mới cho đối tượng.

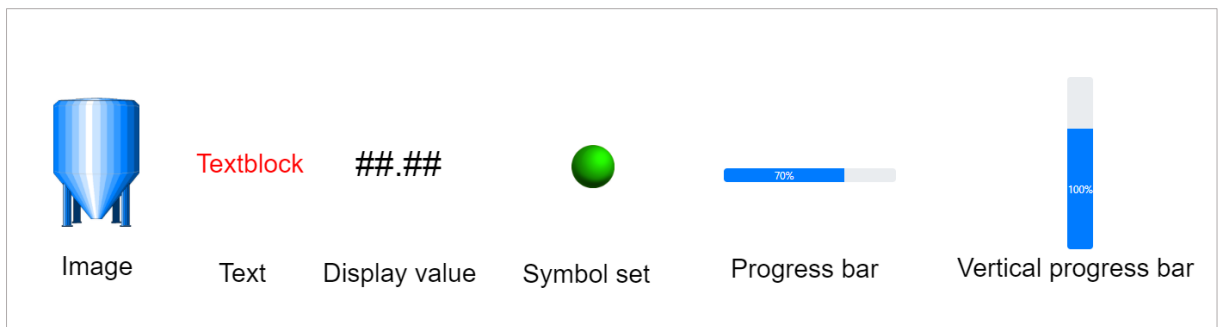

```

var html = document.getElementById(mouseEvent.target.id);
for (draggableItem of draggableObjects) {
    if (draggableItem.element.id == html.id) {
        draggableObjects.splice(draggableObjects.indexOf(draggableItem), 1);
        break;
    }
}
draggable = new PlainDraggable(html, { leftTop: true });
draggable.autoScroll = true;
draggable.containment = document.getElementById('mainPage1');
draggableObjects.push(draggable);

```

4.3.5.3. Nhóm display

Nhóm này gồm các đối tượng image, text, display value, symbol set và progress bar như hình 4.20. Mỗi loại có cách tạo khác nhau, nhưng kỹ thuật xử lý thì tương tự. Khi tạo các đối tượng này, nhấp chuột vào biểu tượng tương ứng, đưa con trỏ chuột ra khu vực làm việc rồi nhấp chuột lần nữa để tạo đối tượng. Có thể tạo nhiều đối tượng bằng cách nhấp chuột nhiều lần. Nhấn ESC để kết thúc vẽ.



Hình 4.20. Nhóm display

Image

Image là hình ảnh tĩnh trên web. Hiện tại chỉ hỗ trợ các hình ảnh có sẵn, chưa thể dùng hình ảnh do người dùng tải lên. Các thuộc tính của Image được thể hiện như hình 4.21.

Hình 4.21. Thuộc tính của Image

Khi khởi tạo đối tượng này, cần xác định vị trí của con trỏ chuột, đây phải là vị trí tương đối so với cửa sổ làm việc chính, không phải so với cả màn hình. Đối tượng mới tạo ra được đưa vào mảng shapes để quản lí.

```
var left = event.pageX - leftOffset + 'px';
var top = event.pageY - topOffset + 'px';

//Declare new image
var defaultImageSrc = '../images/png/default-image.png';
shapes[index] = document.createElement('img');
shapes[index].id = 'img' + nameIndex;
shapes[index].className += ' contextMenu '

//Image css style
shapes[index].src = defaultImageSrc;
shapes[index].style.height = '100px';
shapes[index].style.width = '150px';
shapes[index].style.position = 'absolute';
shapes[index].style.top = top;
shapes[index].style.left = left;

...

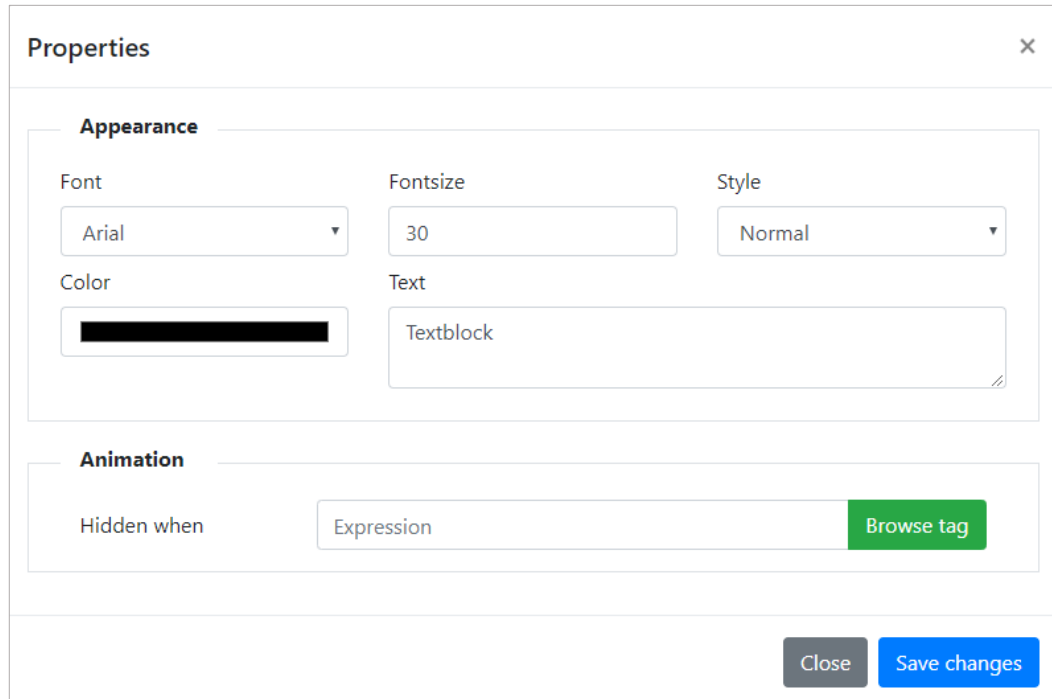
$('#mainPage1').append(shapes[index]);
```

Nhóm đối tượng này dễ dàng kéo thả bằng thư viện JQuery UI draggable và không cần phải khởi tạo lại khi thay đổi vị trí như svg.

```
$(shapes[index]).addClass('draggable');
$('.draggable').draggable({
    refreshPositions: true,
    containment: $('#mainPage1'),
});
```

Text

Text là chuỗi kí tự cố định trên web. Có thể tùy chỉnh các thuộc tính cho text như hình 4.22.



The image shows a 'Properties' dialog box for a text element. It has two tabs: 'Appearance' and 'Animation'. The 'Appearance' tab is selected and contains the following controls:

- Font:** A dropdown menu showing 'Arial'.
- Fontsize:** A text input field containing '30'.
- Style:** A dropdown menu showing 'Normal'.
- Color:** A color picker showing a black color bar.
- Text:** A text area containing 'Textblock'.

The 'Animation' tab is partially visible below and contains:

- Hidden when:** A label.
- Expression:** A text input field.
- Browse tag:** A green button.

At the bottom of the dialog are two buttons: 'Close' (grey) and 'Save changes' (blue).

Hình 4.22. Thuộc tính của text

Giải thuật xử lí text cũng tương tự như Image, chỉ khác nhau ở phần tạo đối tượng.

```
//Declare new paragrap
var para = document.createElement('p');
var text = document.createTextNode('Textblock');
para.appendChild(text);
para.id = 'text' + nameIndex;
para.className += ' contextMenu ';

//Paragraph css style
para.style.fontSize = '30px';
para.style.fontFamily = 'Arial';
para.style.fontStyle = 'normal';
para.style.color = '#000000';
para.style.position = 'absolute';
para.style.top = top;
para.style.left = left;
```

Display value

Về bản chất, display value là một trường hợp riêng của text, trong đó giá trị text thay đổi theo một biến đã cài đặt trước.

Properties [X]

Appearance

Font: Arial [v]
 Fontsize: 40
 Style: Normal [v]
 Color: [black]
 Text: ###.###

Display value

Tag: Insert tag [Browse tag]
 Format: ####.### [v]

Animation

Hidden when: Expression [Browse tag]

[Close] [Save changes]

Hình 4.23. Thuộc tính của Display value

Để thay đổi giá trị của display value, ta đọc giá trị của tag mà nó liên kết, kiểm tra kiểu dữ liệu của tag trước khi chuyển về dạng text để hiển thị ra display value.

```
if (item.tag) {
  if (item.tag.includes(variableName)) {
    if (typeof (eval(item.tag)) == 'boolean') $(item).text(eval(item.tag));
    else $(item).text(eval(item.tag).toFixed(item.format));
  }
}
```

Symbol set

Symbol set là các biểu tượng có thể thay đổi khi thoả mãn một điều kiện nào đó. Ví dụ đối tượng LED có biểu tượng màu xám (tắt) khi LED bằng 0 và trở thành màu xanh (bật) khi LED bằng 1. Đối tượng này được sử dụng phổ biến trong các ứng dụng SCADA, để biểu diễn quá trình bật/ tắt của thiết bị. Các thuộc tính của Symbol set được thể hiện

như hình 4.24.

Properties [X]

Appearance

Width: 50 Position X: 432

Height: 50 Position Y: 695

Symbol

ON condition: Expression [Browse tag]

ON symbol: Expression [Choose image]

OFF symbol: Expression [Choose image]

Animation

Hidden when: Expression [Browse tag]

[Close] [Save changes]

Hình 4.24. Thuộc tính của Symbol set

Quá trình tạo Symbol set tương tự như tạo Image. Để thay đổi hình ảnh theo điều kiện, ta kiểm tra điều kiện và thay đổi thuộc tính source của đối tượng symbol set theo các giá trị đã đặt trước.

```
if (item.onCondition) {  
    if (item.onCondition.includes(variableName)) {  
        if (eval(item.onCondition)) item.src = item.onSymbol;  
        else item.src = item.offSymbol;  
    }  
}
```

Progress bar

Progress bar thường được dùng trong các ứng dụng SCADA để thể hiện mức độ hoàn thành công việc, hoặc mức của chất lỏng trong bồn chứa. Các thuộc tính của progress bar được thể hiện như hình 4.25.

×

Properties

Appearance

Width

30

Height

300

Input

Value

Expression

Browse

☐ Hide value label
 ☐ Raw value

Min tag

Expression

Browse

Max tag

Expression

Browse

Min value

Insert tag

Max value

100

Animation

Hidden when

Expression

Browse tag

Close

Save changes

Hình 4.25. Thuộc tính của progress bar

Trong HTML, progress bar được tạo bởi hai đối tượng div chồng lên nhau. Div thứ nhất là toàn bộ thanh progress bar (100%), div thứ hai nằm bên trong div thứ nhất, có màu sắc khác và độ dài thay đổi được (từ 0 đến 100%) thể hiện tiến trình. Vị trí của div thứ hai phải tuyệt đối so với div thứ nhất.

```
//Horizontal progress bar
var progressbar = document.createElement('div');
progressbar.className = 'progress contextMenu';
progressbar.id = 'progressbar' + nameIndex;
progressbar.isHideLabel = false;
progressbar.min = 0;
progressbar.max = 100;

progressbar.minValue = progressbar.min;
progressbar.maxValue = progressbar.max;

var bar = document.createElement('div');
bar.className = 'progress-bar';
bar.style.width = '70%';
bar.innerText = '70%';

progressbar.appendChild(bar);

//Vertical progress bar
var verticalProgressbar = document.createElement('div');
verticalProgressbar.className = 'progress contextMenu vertical-progress ';
verticalProgressbar.id = 'verticalProgressbar' + nameIndex;
verticalProgressbar.isHideLabel = false;
verticalProgressbar.min = 0;
```

```

verticalProgressbar.max = 100;
verticalProgressbar.minValue = verticalProgressbar.min;
verticalProgressbar.maxValue = verticalProgressbar.max;

var bar = document.createElement('div');
bar.className = 'progress-bar';
bar.style.position = 'absolute';
bar.style.top = '30%';
bar.style.width = '100%';
bar.style.height = '70%';
bar.innerText = '70%';

verticalProgressbar.appendChild(bar);

```

Nếu progress bar nằm ngang thì thay đổi thuộc tính width của thanh tiến trình sẽ làm progress thay đổi. Nếu thanh nằm dọc, cần phải thay đổi cả thuộc tính height và top của thanh tiến trình.

```

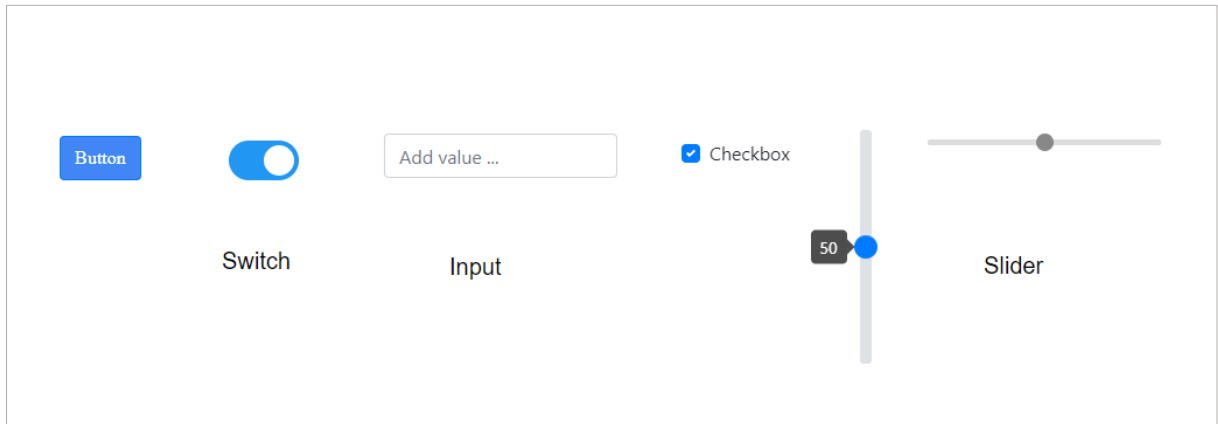
//Horizontal progress bar
if (item.tag) {
  if (item.tag.includes(variableName)) {
    if (eval(item.tag) <= item.min) {
      var _width = '0%';
    } else if (eval(item.tag) >= item.max) {
      var _width = '100%';
    } else {
      var _width = (eval(item.tag) - item.min) / _range * 100 + '%';
    }
    $(item).children('div').css({
      'width': _width,
    });
    if (item.isHideLabel) $(item).children('div').text('');
    else {
      if (!item.isRawValue) $(item).children('div').text(_width);
      else $(item).children('div').text(eval(item.tag));
    }
  }
}

//Vertical progress bar
if (item.tag) {
  if (item.tag.includes(variableName)) {
    if (eval(item.tag) <= item.min) {
      var _height = '0%';
      var _top = '100%';
    } else if (eval(item.tag) >= item.max) {
      var _height = '100%';
      var _top = '0%';
    } else {
      var _height = Number((eval(item.tag) - item.min) / _range * 100).toFixed(3)
+ '%';
      var _top = (100 - (eval(item.tag) - item.min) / _range * 100) + '%';
    }
    $(item).children('div').css({
      'height': _height,
      'top': _top,
    });
    if (item.isHideLabel) $(item).children('div').text('');
    else {
      if (!item.isRawValue) $(item).children('div').text(_height);
      else $(item).children('div').text(Number(eval(item.tag)).toFixed(3));
    }
  }
}
}

```

4.3.5.4. Nhóm control

Nhóm này gồm các đối tượng như button, switch, checkbox, slider như hình 4.26. Chúng có điểm chung là cùng tác động đến đối tượng khác bằng phương thức write.



Hình 4.26. Nhóm control

Button

Button là đối tượng SCADA cơ bản, khi nhấn hoặc nhả sẽ thực hiện một hoặc một chuỗi lệnh được cấu hình trước. Các thuộc tính của button được thể hiện như hình 4.27.

Properties

Appearance

Font	Fontsize	Style
<input type="text"/>	16	Normal
Foreground	Background	Text
<input type="text"/>	<input type="text"/>	Button
Width	70	Position X
Height	38	Position Y
		269
		407

Command

Expression

Animation

Disable when

Hình 4.27. Thuộc tính của Button

Để bắt được sự kiện khi người dùng nhấn nút, button phải được kích hoạt sự kiện click. Tại đây, lệnh được đóng gói và gửi về để server xử lý, sau đó server gửi về gateway để thực thi.

```
$_shape).on('click', function (event) {  
    var _sendObj = {  
        deviceID: deviceID,  
        command: this.command,  
    }  
    _socket.emit('/write', _sendObj);  
})
```

Input

Input được sử dụng trong các yêu cầu về nhập liệu, ví dụ nhập vận tốc đặt hay nhập thời gian reset bộ đếm. Các thuộc tính của input được thể hiện trong hình 4.28.

The screenshot shows a 'Properties' panel for an 'Input' widget. It has three main sections: 'Appearance', 'Input', and 'Animation'. In the 'Appearance' section, 'Width' is set to 200, 'Height' is 38, and 'Input type' is a dropdown menu currently showing 'Number'. The 'Input' section has a 'Tag' field containing 'Expression' and a green 'Browse tag' button. The 'Animation' section has a 'Disable when' field containing 'Expression' and another green 'Browse tag' button. At the bottom right of the panel are two buttons: 'Close' and 'Save changes'.

Hình 4.28. Thuộc tính của Input

Nếu dùng sự kiện changed để phát hiện giá trị input thay đổi và thực thi lệnh thì không phù hợp, vì có thể lúc đó người dùng nhập liệu chưa xong. Thay vì vậy, ta có thể dùng sự kiện keyup và kiểm tra khi nào người dùng nhấn phím Enter sẽ gửi lệnh đi, vừa tiết kiệm tài nguyên cho server, vừa đảm bảo giá trị nhập liệu chính xác.

```

$(_shape).on('keyup', function (event) {
    if (event.keyCode == 13) {
        if (this.tag) {
            if (this.type == 'text') {
                var _sendObj = {
                    deviceID: deviceID,
                    command: this.tag + ' = ' + '"' + this.value + '"'
                }
            }
            else {
                var _sendObj = {
                    deviceID: deviceID,
                    command: this.tag + ' = ' + this.value
                }
            }
            _socket.emit('/write', _sendObj);
        }
    }
});

```

Switch và checkbox

Switch là một dạng checkbox được tùy biến cho phù hợp với yêu cầu về bật/ tắt trong SCADA. Nếu như checkbox là sự kết hợp của input với label thì switch phức tạp hơn, khi cần dùng đến input, label, span và các tùy chỉnh CSS khác. Đoạn code CSS để tạo ra switch như sau:

```

.switch {
    position: relative;
    display: inline-block;
    width: 60px;
    height: 34px;
    float: right;
}
/* Hide default HTML checkbox */
.switch input {
    display: none;
}
/* The slider */
.slider-sw {
    position: absolute;
    cursor: pointer;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background-color: #ccc;
    -webkit-transition: .4s;
    transition: .4s;
}
.slider-sw:before {
    position: absolute;
    content: "";
    height: 26px;
    width: 26px;
    left: 4px;
    bottom: 4px;
    background-color: white;
    -webkit-transition: .4s;
    transition: .4s;
}

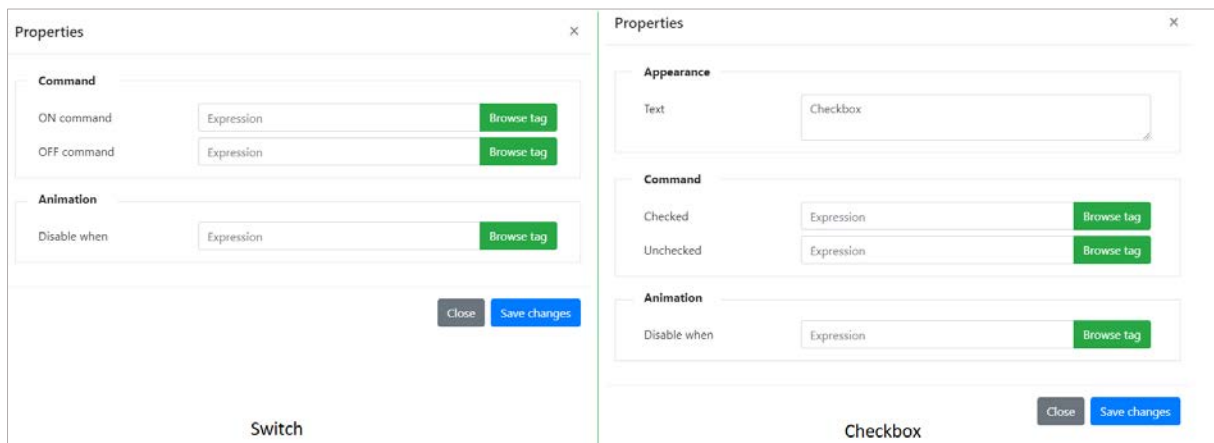
```

```

input.default:checked+.slider-sw {
    background-color: #444;
}
input.primary:checked+.slider-sw {
    background-color: #2196F3;
}
input.success:checked+.slider-sw {
    background-color: #8bc34a;
}
input.info:checked+.slider-sw {
    background-color: #3de0f5;
}
input.warning:checked+.slider-sw {
    background-color: #FFC107;
}
input.danger:checked+.slider-sw {
    background-color: #f44336;
}
input:focus+.slider-sw {
    box-shadow: 0 0 1px #2196F3;
}
input:checked+.slider-sw:before {
    -webkit-transform: translateX(26px);
    -ms-transform: translateX(26px);
    transform: translateX(26px);
}
/* Rounded sliders */
.slider-sw.round {
    border-radius: 34px;
}
.slider-sw.round:before {
    border-radius: 50%;
}

```

Do có cùng nguồn gốc nên thuộc tính của chúng cũng gần giống nhau và được thể hiện qua hình 4.29.



Hình 4.29. Thuộc tính của switch và checkbox

Để phát hiện nút được nhấn, ta dùng sự kiện change của input. Tùy vào trạng thái hiện tại là check (ON) hay uncheck (OFF) mà gửi các lệnh tương ứng về server để xử lý, sau đó server gửi về gateway thực thi.

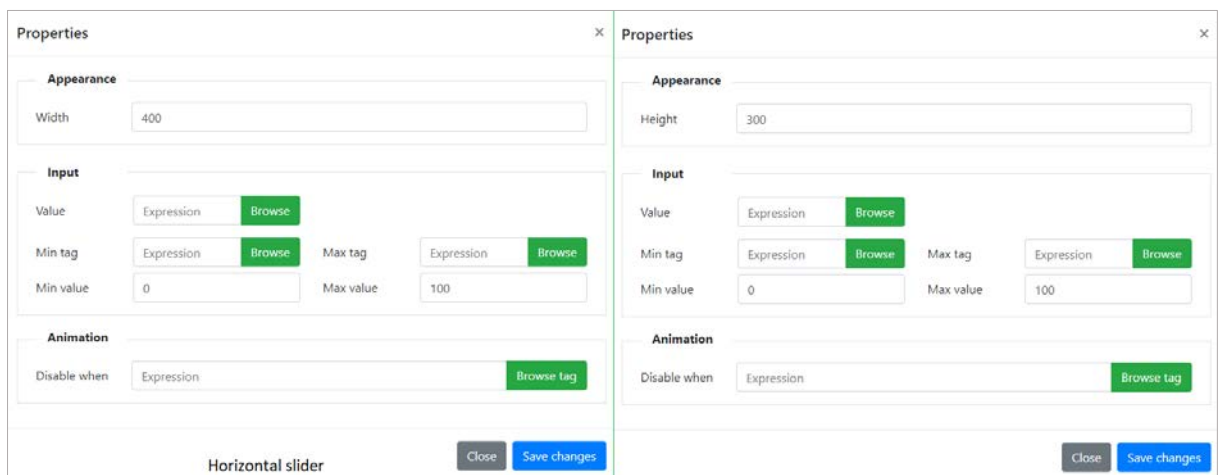
```

var _checkbox = $(_shape).find('input')[0];
var _span = $(_shape).find('span')[0];
if (_checkbox) {
    $(_checkbox).on('change', function () {
        if ($(this).is(':checked')) {
            var _sendObj = {
                deviceID: deviceID,
                command: _span.onCommand,
            };
            _socket.emit('/write', _sendObj);
        } else {
            var _sendObj = {
                deviceID: deviceID,
                command: _span.offCommand,
            };
            _socket.emit('/write', _sendObj);
        }
    });
}

```

Slider

Slider là thanh trượt được dùng như một input với giá trị bị giới hạn. Giá trị của slider bị giới hạn giữa hai mức min và max, có thể là một giá trị cụ thể hoặc liên kết với một biến nào đó. Trong ứng dụng SCADA, slider thường dùng để điều chỉnh tốc độ đặt hay thay đổi công suất máy. Luận văn hỗ trợ hai loại slider là horizontal slider và vertical slider. Thuộc tính của hai loại slider được thể hiện như hình 4.30.



Hình 4.30. Thuộc tính của slider

Mặc dù cùng là slider nhưng hai loại này lại có cách thiết kế và cơ chế hoạt động khác nhau. Horizontal slider đơn giản chỉ là input có kiểu range, kết hợp với một số thuộc tính CSS để tạo hình dáng đẹp.

```

var slider = document.createElement('input');
slider.type = 'range';
slider.className = 'custom-range contextMenu ';
slider.id = 'slider' + nameIndex;
slider.min = 0;
slider.max = 100;
slider.minValue = slider.min;
slider.maxValue = slider.max;
//CSS style
slider.style.position = 'absolute';
slider.style.top = top;
slider.style.left = left;
slider.style.width = '400px';

```

Vertical slider phức tạp hơn, vì mặc định HTML không hỗ trợ loại input này. Để tạo vertical slider, cần một div chứa input, sau đó dùng thêm thư viện bootstrapSlider. Tuy nhiên thư viện này chỉ hỗ trợ Bootstrap 3.0 trở xuống, trong khi phiên bản mà luận văn sử dụng là 4.0, dẫn đến không tạo được slider. Hàm fixTooltip được viết để giải quyết vấn đề này.

```

var verticalSliderDiv = document.createElement('div');
verticalSliderDiv.className = ' contextMenu ';
verticalSliderDiv.style.background = 'transparent';
var verticalSlider = document.createElement('input');
verticalSlider.type = 'range';
verticalSlider.id = 'verticalSlider' + nameIndex;
verticalSlider.min = 0;
verticalSlider.max = 100;
verticalSlider.minValue = verticalSlider.min;
verticalSlider.maxValue = verticalSlider.max;
//CSS style
verticalSliderDiv.style.position = 'absolute';
verticalSliderDiv.style.top = top;
verticalSliderDiv.style.left = left;
verticalSliderDiv.append(verticalSlider);
$('#mainPage1').append(verticalSliderDiv);

//Create vertical slider
$(verticalSlider).bootstrapSlider({
  min: verticalSlider.min,
  max: verticalSlider.max,
  value: 50,
  orientation: 'vertical',
  tooltip_position: 'left',
  reversed: true,
  enabled: false,
});
fixTooltip();
$(verticalSlider).siblings('div')[0].style.height = '300px';

//Fix tooltip for vertical slider
function fixTooltip(sliderID) {
  $('.slider-vertical').each(function () {
    $(this).children('.tooltip')[0].classList.add('bs-tooltip-left');
    $(this).children('.tooltip')[0].childNodes[0].classList.add('arrow', 'my-2');
  })
  $('.slider-vertical').find('.slider-handle').each(function () {
    $(this).css({ background: '#007bff' })
  });
}

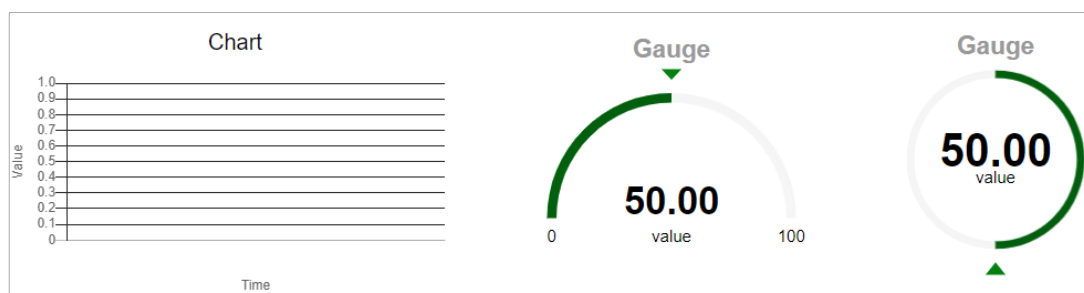
```

Slider hoạt động theo cơ chế hai chiều, nghĩa là vừa có thể thay đổi giá trị của biến, vừa cập nhật giá trị của biến để hiển thị trên slider. Ở chiều hiển thị, horizontal slider chỉ cần gán giá trị cần hiển thị vào thuộc tính value của nó, trong khi vertical slider phải thông qua hàm bootstrapSlider('setValue'). Ở chiều điều khiển, horizontal slider cung cấp sự kiện change, còn vertical slider có sự kiện slideStop, cả hai sự kiện đều diễn ra khi người dùng thả chuột.

```
//Horizontal slider
if (item.tag) {
  if (item.tag.includes(variableName)) {
    item.value = eval(item.tag);
  }
}
$(_shape).on('change', function (event) {
  var _sendObj = {
    deviceID: deviceID,
    command: this.tag + ' = ' + this.value
  }
  if (this.tag) _socket.emit('/write', _sendObj);
});
//Vertical slider
if (item.tag) {
  if (item.tag.includes(variableName)) {
    $(item).bootstrapSlider('setValue', eval(item.tag))
  }
}
$(_shape).bootstrapSlider('enable');
$(_shape).on('slideStop', function (event) {
  var _sendObj = {
    deviceID: deviceID,
    command: this.tag + ' = ' + this.value
  }
  if (this.tag) _socket.emit('/write', _sendObj);
});
```

4.3.5.5. Nhóm advance

Nhóm này gồm chart và gauge (hình 4.31), là các đối tượng phức tạp, tốn nhiều thời gian render mà HTML thuần túy không thực hiện được. Chart.js và Justgage.js là hai thư viện được sử dụng trong luận văn.



Hình 4.31. Chart và gauge

Chart

Chart là đồ thị dùng để hiển thị trực quan giá trị của một hoặc nhiều đại lượng theo thời gian. Hiện tại luận văn chỉ hỗ trợ một biến cho mỗi chart. Các thuộc tính của chart được thể hiện trong hình 4.32.

The image shows a 'Properties' dialog box for a chart. It is divided into three main sections: 'Appearance', 'Input', and 'Animation'. In the 'Appearance' section, there are input fields for 'Width' (set to 400) and 'Height' (set to 200). The 'Input' section contains a 'Variable' dropdown menu with the text 'Select variable for this chart' and a green 'Browse tag' button, followed by 'X-axis' (set to 'Time') and 'Y-axis' (set to 'Value'). The 'Animation' section has a 'Hidden when' dropdown menu with the text 'Expression' and another green 'Browse tag' button. At the bottom right of the dialog are two buttons: 'Close' and 'Save changes'.

Hình 4.32. Thuộc tính của chart

Bản chất của chart là một canvas, do vậy phải khởi tạo canvas trước. Tuy nhiên, để canvas không bị vỡ độ phân giải, phải đặt nó trong div. Sau khi chuẩn bị xong các nguyên liệu, gọi hàm Chart trong chart.js để chuyển canvas về chart.

```
var canvas = document.createElement('canvas');
canvas.className = 'chart contextMenu';
canvas.id = 'chart' + nameIndex;
canvas.xLabel = 'Time';
canvas.yLabel = 'Value';

var chartDiv = document.createElement('div');
chartDiv.id = 'chartDiv' + nameIndex;
chartDiv.appendChild(canvas);

chartDiv.style.position = 'absolute';
chartDiv.style.top = top;
chartDiv.style.left = left;
canvas.style.height = '200px';
canvas.style.width = '500px';
```

```

//Create chart
var ctx1 = canvas.getContext('2d');
var newChart = new Chart(ctx1, {
  type: 'line',
  data: {
    labels: [],
    datasets: [{
      steppedLine: false,
      backgroundColor: 'rgba(57,172,180 , 0.8)',
      hoverBackgroundColor: 'rgba(57,172,180 , 0.3)',
      data: [],
      label: 'Value',
      borderColor: 'rgb(0,102,10)',
      borderWidth: 2,
      pointRadius: 0,
    }]
  },
  options: {
    legend: false,
    responsive: true,
    maintainAspectRatio: false,
    title: {
      display: false,
    },
    tooltips: {
      mode: 'index',
      intersect: false,
      titleFontSize: 16,
      bodyFontSize: 16
    },
    hover: {
      mode: 'nearest',
      intersect: true
    },
    scales: {
      xAxes: [{
        display: true,
        // type : 'realtime',
        scaleLabel: {
          display: true,
          labelString: canvas.xLabel,
        },
      }],
      yAxes: [{
        ticks: {
          beginAtZero: true
        },
        display: true,
        gridLines: {
          color: '#282525'
        },
        scaleLabel: {
          display: true,
          labelString: canvas.yLabel,
        },
      }]
    },
  },
});

```

Dữ liệu của chart gồm mảng label và mảng data lần lượt chứa nhãn và giá trị ứng với nhãn đó. Với mỗi dữ liệu mà gateway gửi về, cần trích xuất timestamp và giá trị tương ứng để thêm vào chart.


```

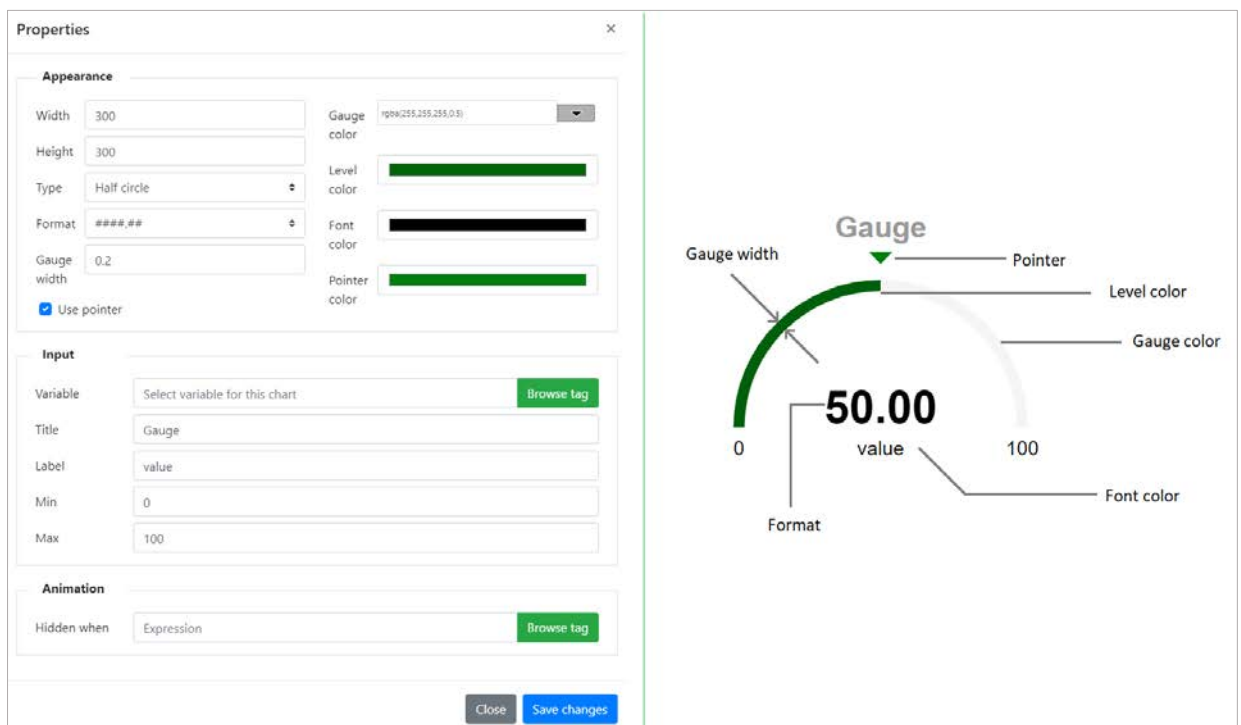
var foundChartIndex = findChartById(canvas.id);
if (foundChartIndex != -1) { //Found chart JS object
    if (canvas.tag) {
        if (canvas.tag.includes(variableName)) {
            var label = moment(variableTimestamp).format('MM:mm:ss');
            var data = eval(canvas.tag);
            addChartData(arrChartJS[foundChartIndex].node, label, data);
        }
    }
}

//Hàm update chart
function addChartData(chart, label, data) {
    chart.data.labels.push(label);
    chart.data.datasets[0].data.push(data);
    chart.update();
}

```

Gauge

Gauge là đồng hồ đo đại lượng vật lý như vận tốc, áp suất, nhiệt độ. Trong các thiết kế giao diện hiện đại, gauge rất được ưa chuộng vì tính thẩm mỹ và tường minh của nó. Các thuộc tính của gauge được thể hiện như hình 4.33.



Hình 4.33. Các thuộc tính của gauge

Về bản chất, gauge là một svg với những thuộc tính phức tạp. Justgage đã giúp đơn giản hoá sự phức tạp đó. Thư viện Justgage tự động vẽ những svg bên trong các div mà

người dùng chỉ định.

```
var gaugeDiv = document.createElement('div');
gaugeDiv.id = 'gauge' + nameIndex;
gaugeDiv.className = 'gauge contextMenu';

gaugeDiv.style.position = 'absolute';
gaugeDiv.style.top = top;
gaugeDiv.style.left = left;
gaugeDiv.style.height = '400px';
gaugeDiv.style.width = '400px';

gaugeDiv.type = false;
gaugeDiv.format = 2;
gaugeDiv.usePointer = true;
gaugeDiv.gaugeWidth = 0.2;
gaugeDiv.gaugeColor = 'rgba(255,255,255,0.5)';
gaugeDiv.levelColor = ['#00660a'];
gaugeDiv.pointerColor = '#00800d';
gaugeDiv.fontColor = '#000000';
gaugeDiv.label = 'value';
gaugeDiv.min = 0;
gaugeDiv.max = 100;
gaugeDiv.title = "Gauge";
$('#mainPage1').append(gaugeDiv);

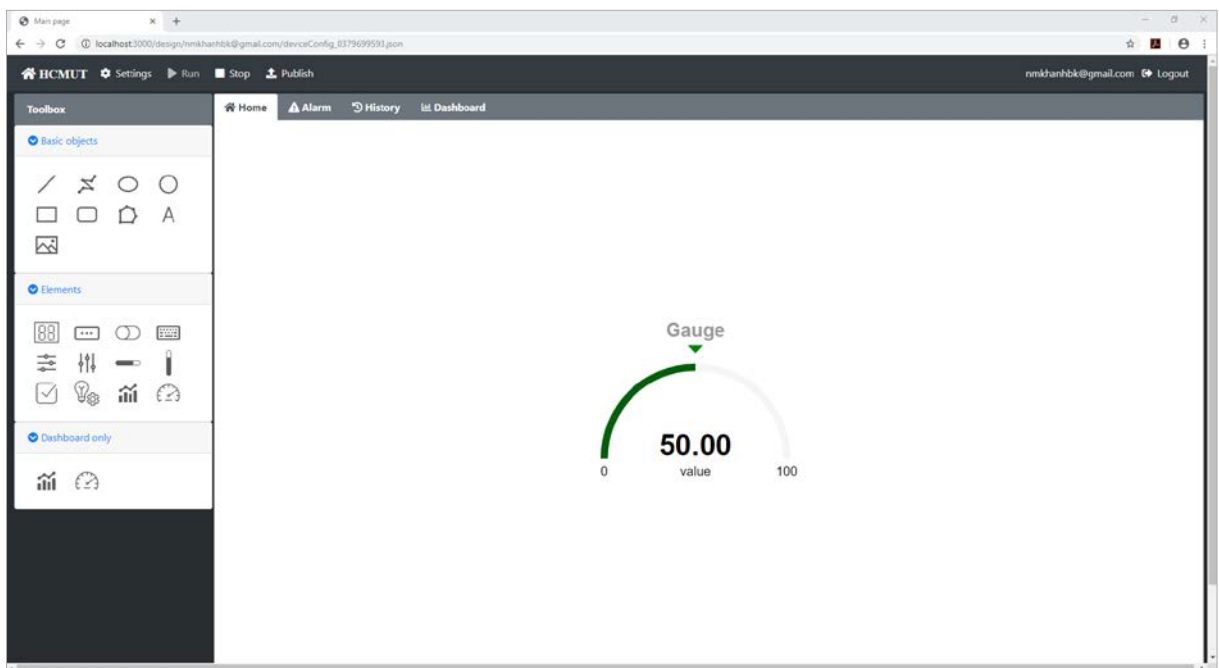
var gauge = new JustGage({
  id: gaugeDiv.id,
  value: 50,
  decimals: gaugeDiv.format,
  title : gaugeDiv.title,
  min: gaugeDiv.min,
  max: gaugeDiv.max,
  label: gaugeDiv.label,
  labelFontColor: gaugeDiv.fontColor,
  donut: gaugeDiv.type,
  relativeGaugeSize: true,
  valueFontColor: gaugeDiv.fontColor,
  valueFontSize: '10px',
  gaugeColor: gaugeDiv.gaugeColor,
  levelColors: gaugeDiv.levelColor,
  pointer: gaugeDiv.usePointer,
  pointerOptions: {
    toplength: 8,
    bottomlength: -20,
    bottomwidth: 6,
    color: gaugeDiv.pointerColor
  },
  gaugeWidthScale: gaugeDiv.gaugeWidth,
  counter: true,
});
```

Khi cần thay đổi giá trị của gauge, Justgage cung cấp hàm refresh với tham số là giá trị muốn thay đổi.

```
var foundGaugeIndex = findGaugeById(item.id);
if (foundGaugeIndex != -1) { //Found chart JS object
  if (item.tag) {
    if (item.tag.includes(variableName)) {
      arrGauge[foundGaugeIndex].node.refresh(eval(item.tag));
    }
  }
}
```

4.3.6. Chức năng vận hành thử

Vận hành thử là chức năng mà bất kì phần mềm thiết kế SCADA nào cũng phải có. Người dùng thiết kế giao diện xong, kiểm tra hoạt động bằng cách nhấn nút Run trên thanh công cụ. Khi chức năng Run được kích hoạt, quá trình trao đổi dữ liệu (tags, alarm) giữa server và client bắt đầu, mọi hoạt động nhằm can thiệp đến thiết kế hệ thống đều bị cấm, toolbox bị vô hiệu như hình 4.34. Hai cửa sổ Home và Dashboard liên tục cập nhật trạng của đối tượng khi có thông báo mới. Cửa sổ alarm theo dõi các cảnh báo, nếu có cảnh báo mới sẽ nhấp nháy tiêu đề màu vàng để gây chú ý. Các nút nhấn ở cửa sổ History cũng được kích hoạt, cho phép người dùng truy vấn dữ liệu và xuất báo cáo. Nhấn nút Stop trên thanh công cụ để thoát khỏi chế độ vận hành thử, trở về chế độ thiết kế.



Hình 4.34. Kích hoạt chức năng Run (toolbox bị vô hiệu)

Quản lí giao diện

Khi ở chức năng Run, nút Run, toolbox và mọi modal thuộc tính đối tượng bị vô hiệu, nút Stop được kích hoạt (để thoát khỏi chế độ Run). Bên cạnh đó, các đối tượng bị khoá tại vị trí hiện tại, không thể kéo thả để thay đổi vị trí.

```

$(this).prop('disabled', true);
$('#btnStop').prop('disabled', false);
$('.button-icon').prop('disabled', true);
$('#alarm *').prop('disabled', false);
$('#history *').prop('disabled', false);
draggableObjects.forEach(function (item) {
    item.disabled = true;
});
$('.draggable').draggable('disable');
$('.draggable2').draggable('disable');
$('.inputModal').prop('disabled', true);
$('.btnBrowseTag').prop('disabled', true);
$('.btnChooseImage').prop('disabled', true);
$('.saveChangeButton').prop('disabled', true);

```

Dựa vào mảng shapes (mảng quản lí các object Javascript của đối tượng), client sẽ khởi tạo SCADA cho nhóm control bằng cách đăng kí sự kiện để lắng nghe thay đổi của chúng.

```

function initSCADA(_shapes, _socket) {
    _shapes.forEach(function (_shape) {
        var _id = _shape.id.toString().toLowerCase().replace(/[0-9]/g, '');
        switch (_id) {
            case 'button': {
                $(_shape).on('click', function (event) {
                    var _sendObj = {
                        deviceId: deviceId,
                        command: this.command,
                    }
                    _socket.emit('/write', _sendObj);
                })
                break;
            }
            case 'input': {
                $(_shape).on('keyup', function (event) {
                    if (event.keyCode == 13) {
                        if (this.tag) {
                            if (this.type == 'text') {
                                var _sendObj = {
                                    deviceId: deviceId,
                                    command: this.tag + ' = ' + "'" + this.value + "'"
                                }
                            }
                            else {
                                var _sendObj = {
                                    deviceId: deviceId,
                                    command: this.tag + ' = ' + this.value
                                }
                            }
                            _socket.emit('/write', _sendObj);
                        }
                    }
                });
                break;
            }
            case 'slider': {
                $(_shape).on('change', function (event) {
                    var _sendObj = {
                        deviceId: deviceId,
                        command: this.tag + ' = ' + this.value
                    }
                    if (this.tag) _socket.emit('/write', _sendObj);
                });
            }
        }
    });
}

```

```

        break;
    }
    case 'verticalslider': {
        $(_shape).bootstrapSlider('enable');
        $(_shape).on('slideStop', function (event) {
            var _sendObj = {
                deviceID: deviceID,
                command: this.tag + ' = ' + this.value
            }
            console.log(_sendObj)
            if (this.tag) _socket.emit('/write', _sendObj);
        });
        break;
    }
    default: {
        if ($(_shape).find('span')[0]) {
            var _checkbox = $(_shape).find('input')[0];
            var _span = $(_shape).find('span')[0];
            if (_checkbox) {
                $(_checkbox).on('change', function () {
                    if ($(this).is(':checked')) {
                        var _sendObj = {
                            deviceID: deviceID,
                            command: _span.onCommand,
                        }
                        _socket.emit('/write', _sendObj);
                    }
                    else {
                        var _sendObj = {
                            deviceID: deviceID,
                            command: _span.offCommand,
                        }
                        _socket.emit('/write', _sendObj);
                    }
                });
            }
            else {
                var _label = $(_shape).find('label')[0];
                var _checkbox = $(_shape).find('input')[0];
                if (_checkbox) {
                    $(_checkbox).on('change', function () {
                        if ($(this).is(':checked')) {
                            var _sendObj = {
                                deviceID: deviceID,
                                command: _label.checkedCommand,
                            }
                            _socket.emit('/write', _sendObj);
                        }
                        else {
                            var _sendObj = {
                                deviceID: deviceID,
                                command: _label.unCheckedCommand,
                            }
                            _socket.emit('/write', _sendObj);
                        }
                    });
                }
            }
        }
    }
}

```

Một socket được mở giữa client và server để trao đổi giá trị các tag. Socket này có dạng: **/device ID/tag**. Do device ID là duy nhất nên socket này cũng là duy nhất với mỗi

client, đảm bảo dữ liệu đến đúng nơi và bảo mật. Khi có dữ liệu mới, các tag được cập nhật giá trị trong bộ nhớ của client (giúp giảm thời gian truy xuất sau này) và được hàm SCADA() điều khiển để thực hiện các hiệu ứng đã đặt trước.

```
socket.on('/') + deviceID + '/tag', function (data) {
    var arrVarObjects = JSON.parse(data);
    if (arrVarObjects) {
        arrVarObjects.variables.forEach(function (varObject) {
            eval(varObject.tagName + '=' + varObject.value);
            SCADA(shapes, varObject.tagName, varObject.timeStamp);
        });
    }
});
//Hàm SCADA để điều khiển hiển thị các đối tượng
function SCADA(arrHtmlElems, variableName, variableTimestamp) {
    shapes.forEach(function (_shape) {
        var _id = _shape.id.toString().toLowerCase().replace(/[0-9]/g, '');
        switch (_id) {
            case 'text': {
                scadaTextObject(_shape, variableName);
                break;
            }
            case 'img': {
                scadaImageObject(_shape, variableName);
                break;
            }
            case 'displayvalue': {
                scadaDisplayValueObject(_shape, variableName);
                break;
            }
            case 'input': {
                scadaInputObject(_shape, variableName);
                break;
            }
            case 'switch': {
                scadaSwitchObject(_shape, variableName);
                break;
            }
            case 'button': {
                scadaButtonObject(_shape, variableName);
                break;
            }
            case 'slider': {
                scadaSliderObject(_shape, variableName);
                break;
            }
            case 'verticalslider': {
                scadaVerticalSliderObject(_shape, variableName);
                break;
            }
            case 'progressbar': {
                scadaProgressBarObject(_shape, variableName);
                break;
            }
            case 'verticalprogressbar': {
                scadaVerticalProgressBarObject(_shape, variableName);
                break;
            }
            case 'checkbox': {
                scadaCheckboxObject(_shape, variableName);
                break;
            }
            case 'symbolset': {
```

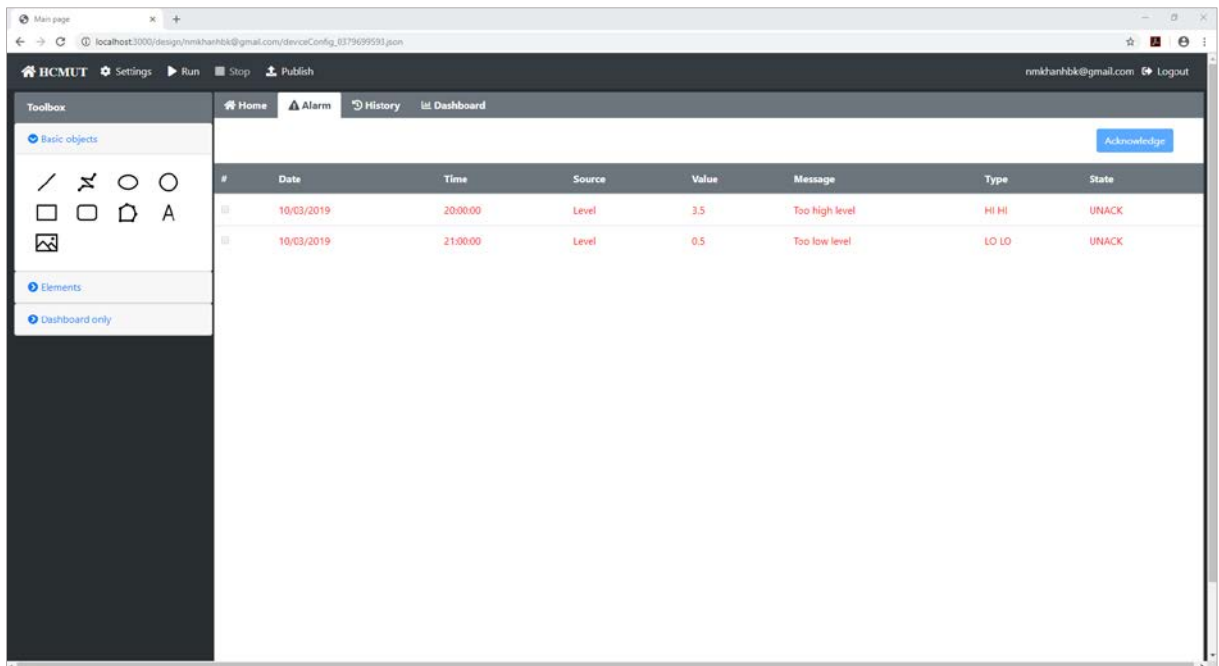
```

        scadaSymbolSetObject(_shape, variableName);
        break;
    }
    case 'chartdiv': {
        scadaChartObject(_shape, variableName, variableTimestamp);
        break;
    }
    case 'gauge' : {
        scadaGaugeObject(_shape, variableName);
        break;
    }
    default: {
        if ($(_shape).find('span')[0]) {
            scadaSwitchObject(_shape, variableName);
        } else if ($(_shape).find('label')[0]) {
            scadaCheckboxObject(_shape, variableName);
        } else {
            scadaSvgObject(_shape, variableName);
        }
    }
}
})
}

```

Quản lí alarm

Màn hình cảnh báo được thiết kế như hình 4.35. Khi có cảnh báo mới, tiêu đề Alarm sẽ nhấp nháy vàng để người dùng chú ý cho đến khi các cảnh báo được xác nhận. Khu vực chính của màn hình là nơi hiển thị danh sách các cảnh báo, cảnh báo mới nhất sẽ nằm trên cùng. Các cảnh báo chưa được xác nhận (UNACK) có màu đỏ, các cảnh báo đã được xác nhận trở về màu đen. Nút nhấn Acknowledge dùng xác nhận một hoặc nhiều cảnh báo. Việc chọn cảnh báo được thực hiện bằng cách nhấp chuột lên hàng tương ứng của cảnh báo đó.



Hình 4.35. Màn hình alarm

Ngay khi kích hoạt chức năng Run, bảng alarm được xóa để đảm bảo mọi cảnh báo trong bảng đều là mới nhất. Một socket được mở giữa client và server để truyền nhận cảnh báo. Socket này có dạng: **/device ID/alarm**, do đó là duy nhất với mỗi client. Khi có cảnh báo mới, client kiểm tra nguồn và loại cảnh báo đã có trong bảng hay chưa, nếu chưa thì thêm mới vào bảng với trạng thái UNACK (màu đỏ). Nếu nguồn và loại cảnh báo đã có thì kiểm tra trạng thái của cảnh báo đó là ACKED hay UNACK. Nếu UNACK thì cập nhật cảnh báo cũ, nếu là ACKED thì thay đổi màu và trạng thái cảnh báo (đỏ thành đen, UNACK thành ACKED).

```
$('#alarmTable tbody').empty();
socket.on('/' + deviceID + '/alarm', function (alarmObject) {
    var arrAlarmSource = Array.from($('#alarmTable tr td:nth-child(4)'));
    var arrAlarmType = Array.from($('#alarmTable tr td:nth-child(7)'));
    var arrAlarmState = Array.from($('#alarmTable tr td:nth-child(8)'));
    var _isExist = false;
    var _timeStamp = new Date(alarmObject.timestamp)
    for (var i = 0; i < arrAlarmSource.length; i++) {
        if ((arrAlarmSource[i].innerText == alarmObject.source) &&
            (arrAlarmType[i].innerText == alarmObject.type) && (arrAlarmState[i].innerText ==
            'UNACK')) {
            if (alarmObject.state == 'UNACK') {
                var _expression = '#alarmTable tr:nth(' + (i + 1) + ') td';
                var tableRow = $(_expression);
                tableRow[1].innerText = _timeStamp.toLocaleDateString();
                tableRow[2].innerText = _timeStamp.toLocaleTimeString();
                tableRow[4].innerText = alarmObject.value;
                tableRow[5].innerText = alarmObject.message;
            }
        }
    }
});
```



```

        tableRow[6].innerText = alarmObject.type;
        tableRow[7].innerText = alarmObject.state;
    }
    else { //ACKED
        var _expression = '#alarmTable tr:nth(' + (i + 1) + ') td';
        var tableRow = $(_expression);
        tableRow[1].innerText = _timeStamp.toLocaleDateString();
        tableRow[2].innerText = _timeStamp.toLocaleTimeString();
        tableRow[7].innerText = alarmObject.state;
        $(arrAlarmSource[i].closest('tr')).css('color', 'black');
    }
    _isExist = true;
    break;
}
}
if (!_isExist) { //Not found item
    var _htmlMarkup =
        `<tr class = "row-pointer">
            <td><input type="checkbox" class = "alarmCheckbox"></td>
            <td>` + _timeStamp.toLocaleDateString() + `</td>
            <td>` + _timeStamp.toLocaleTimeString() + `</td>
            <td>` + alarmObject.source + `</td>
            <td>` + alarmObject.value + `</td>
            <td>` + alarmObject.message + `</td>
            <td>` + alarmObject.type + `</td>
            <td>` + alarmObject.state + `</td>
        </tr>`
    $('#alarmTable').prepend(_htmlMarkup);

    $('#alarmTable tbody tr:nth-child(1)').click(function () {
        var _checkbox = $(this).children('td').children('input');
        _checkbox.prop('checked', !_checkbox.prop('checked'));
        if (_checkbox.prop('checked')) $(this).addClass('alarm-selected');
        else $(this).removeClass('alarm-selected');
    })
}
});

```

Để tạo hiệu ứng nhấp nháy khi có cảnh báo mới, client sẽ tìm trong bảng alarm, nếu có phần tử nào UNACK sẽ khởi động timer, liên tục thay đổi thuộc tính color của CSS với chu kỳ 1 giây. Khi tất cả cảnh báo đã được xác nhận (ACKED), client thu hồi timer, trả thuộc tính color của CSS về giá trị của class đang nắm giữ nó.

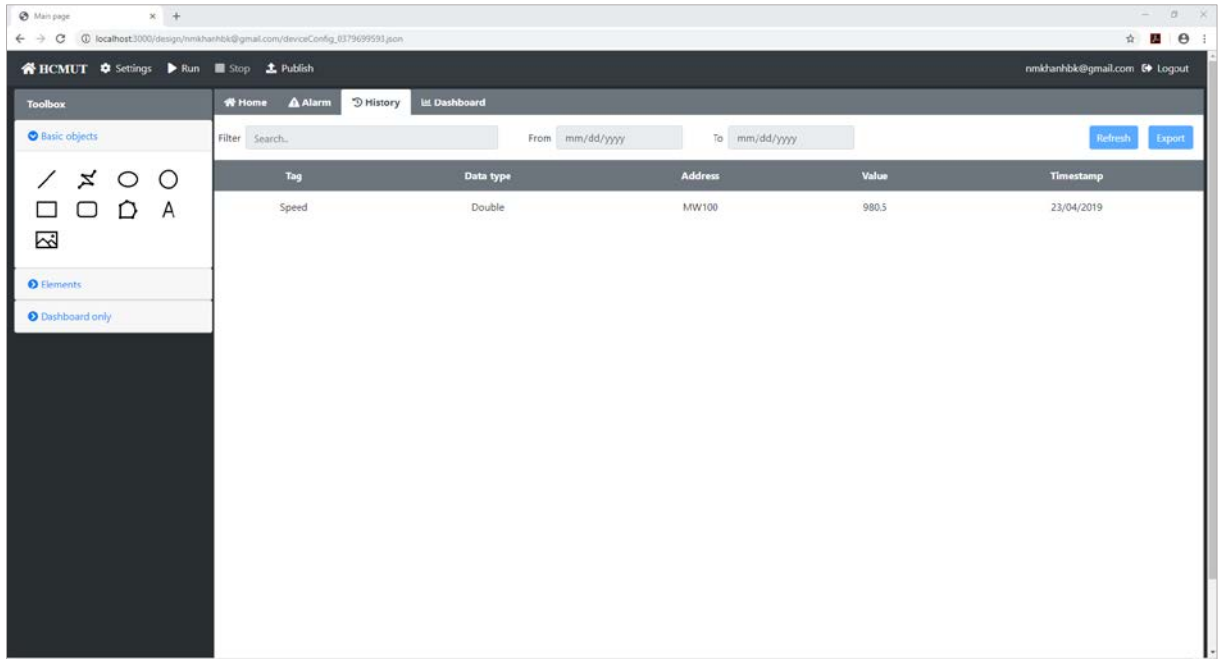
```

var isUNACK = false;
var arrAlarmState_New = Array.from($('#alarmTable tr td:nth-child(8)'));
var arrAlarmStateValue = [];
for (var i = 0; i < arrAlarmState_New.length; i++) {
    arrAlarmStateValue.push(arrAlarmState_New[i].innerText);
};
if (arrAlarmStateValue.includes('UNACK')) {
    if (!isFlashing) {
        alarmEffectInterval = setInterval(function () {
            if ($('#alarmTitle').css('color') == 'rgb(255, 255, 255)')
                $('#alarmTitle').css('color', 'orange');
            else $('#alarmTitle').css('color', ''), 1000;
            isFlashing = true;
        }
    }
} else {
    isFlashing = false;
    clearInterval(alarmEffectInterval);
    $('#alarmTitle').css('color', '');
}

```

Quản lý history

Màn hình quản lý history được thiết kế như hình 4.36. Vì lượng dữ liệu history có thể rất lớn nên client không tự tải về, mà sẽ đợi người dùng nhấn Refresh mới yêu cầu dữ liệu từ server. Việc này giúp client không bị phân tán tài nguyên khi đang xử lý tác vụ khác. Nút Export để xuất dữ liệu ra bảng excel hoặc pdf. Bên trái màn hình là dãy các bộ lọc, giúp tìm ra dữ liệu phù hợp với yêu cầu của người dùng.



Hình 4.36. Màn hình history

Cũng giống như alarm, bảng history sẽ được xóa khi bắt đầu chế độ Run. Socket **/device ID/resHistory** mở ra để client nhận dữ liệu từ server. Server chỉ gửi dữ liệu khi client có yêu cầu, thông qua socket **/device ID/reqHistory**.

```
socket.on('/' + deviceID + '/resHistory', function (data) {
  if (data.length > 0) {
    $('#historyTable tbody').empty();
    data.forEach(function (dataItem) {
      var _htmlMarkup =
        `<tr>
          <td>` + dataItem.tag + `</td>
          <td>` + dataItem.type + `</td>
          <td>` + dataItem.address + `</td>
          <td>` + dataItem.value + `</td>
          <td>` + dataItem.timestamp + `</td>
        </tr>`;
      $('#historyTable tbody').append(_htmlMarkup);
      $('#historyTable tbody').css({'height' : '800px', 'overflow-y' :
```

```

    'auto' });
    });
  }
});

$('#btnRefreshHistory').click(function () {
    socket.emit('/reqHistory', deviceID);
});

```

Trở về chế độ thiết kế

Để trở về chế độ thiết kế, nhấn nút Stop trên thanh công cụ. Khi đó, nút Stop bị vô hiệu, nút Run và các chức năng thiết kế được phục hồi. Các đối tượng SCADA đang bị vô hiệu hoặc bị ẩn sẽ được kích hoạt lại, dữ liệu của chart được làm mới để không chiếm tài nguyên hệ thống. Client chủ động đóng các socket giao tiếp với server để nhường băng thông cho các client khác.

```

$(this).prop('disabled', true);
$('#btnRun').prop('disabled', false);
$('.button-icon').prop('disabled', false);
$('#alarm *').prop('disabled', true);
$('#history *').prop('disabled', true);
draggableObjects.forEach(function (item) {
    item.disabled = false;
});
$('.draggable').draggable('enable');
$('.draggable2').draggable('enable');
//Enable input
$('.inputModal').prop('disabled', false);
$('.btnBrowseTag').prop('disabled', false);
$('.btnChooseImage').prop('disabled', false);
$('.saveChangeButton').prop('disabled', false);
//Disable vertical slider
$('.slider-vertical').siblings('input').bootstrapSlider('disable');

clearChartData();
showHiddenItems();
enableAllItems();

socket.off('/') + deviceID + '/tag');
socket.off('/') + deviceID + '/alarm');
socket.off('/') + deviceID + '/resHistory');
//Reset alarm color, clear Interval
$('#alarmTitle').css('color', '');
clearInterval(alarmEffectInterval);

function clearChartData() {
    for (var i = 0; i < arrChartJS.length; i++) {
        arrChartJS[i].node.data.labels = [];
        arrChartJS[i].node.data.datasets[0].data = [];
        arrChartJS[i].node.update();
    }
}

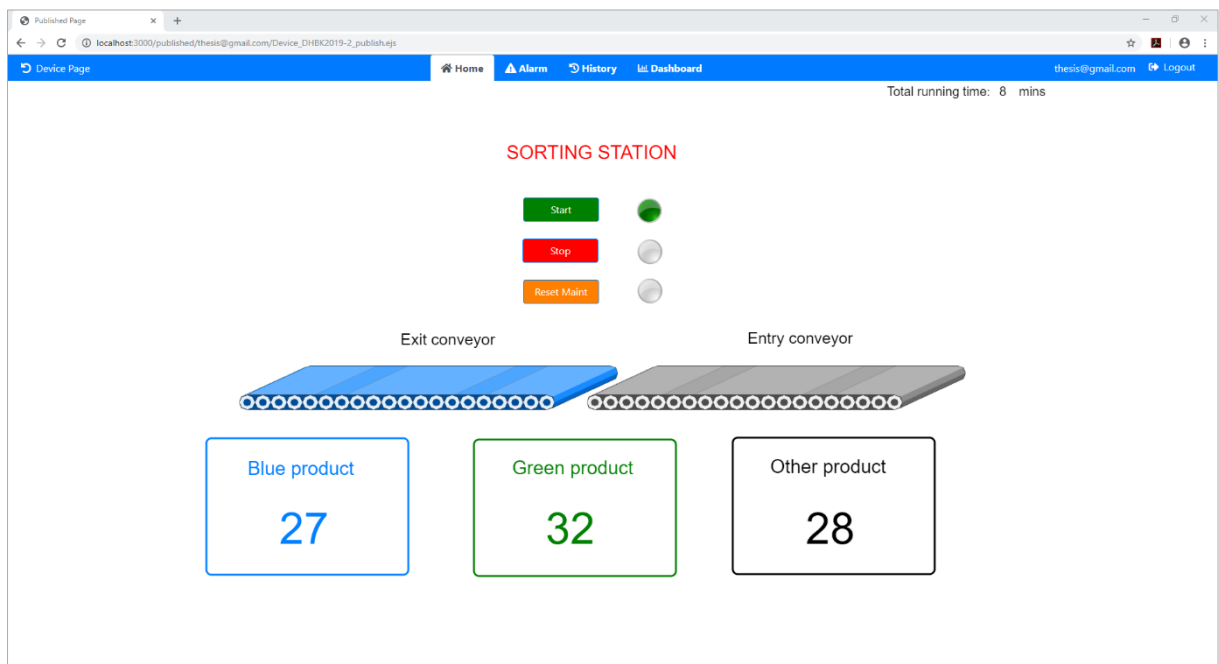
function showHiddenItems() {
    for (var i = 0; i < elementHTML.length; i++) {
        if (elementHTML[i].type != 'verticalslider')
            $('#' + elementHTML[i].id).show();
    }
}

```

```
function enableAllItems() {
  for (var i = 0; i < elementHTML.length; i++) {
    if (elementHTML[i].type == 'switch' || elementHTML[i].type == 'checkbox') {
      $(document.getElementById(elementHTML[i].id).parentNode).find('input')
        $(document.getElementById(elementHTML[i].id).parentNode).find('input').prop('
disabled', false);
    } else $('#' + elementHTML[i].id).prop('disabled', false);
  }
}
```

4.3.7. Chức năng publish

Publish là chức năng xuất bản thiết kế SCADA ra một trang web cố định như hình 4.37. Liên kết dẫn đến trang web SCADA này nằm ở trang quản lý gateway. Server nhận biết một gateway đã được publish hay chưa bằng cách kiểm tra thuộc tính published trong file cấu hình của gateway.



Hình 4.37. Giao diện publish

Khi nhấn nút Publish trên thanh công cụ, client đọc tất cả đối tượng trong Home và Dashboard. Các giá trị này được nén vào object `_sendObject` cùng với các mảng quản lý đối tượng. Với object này server có thể tái tạo lại các đối tượng SCADA đã thiết kế. Ngoài ra, để tránh làm phức tạp cấu trúc của `_sendObject`, client trích xuất các thông số liên quan đến màu nền sang một object riêng gọi là `backgroundObject`. Socket `/publish` được mở ra, cho phép client truyền hai object đó đến server. Client lắng nghe

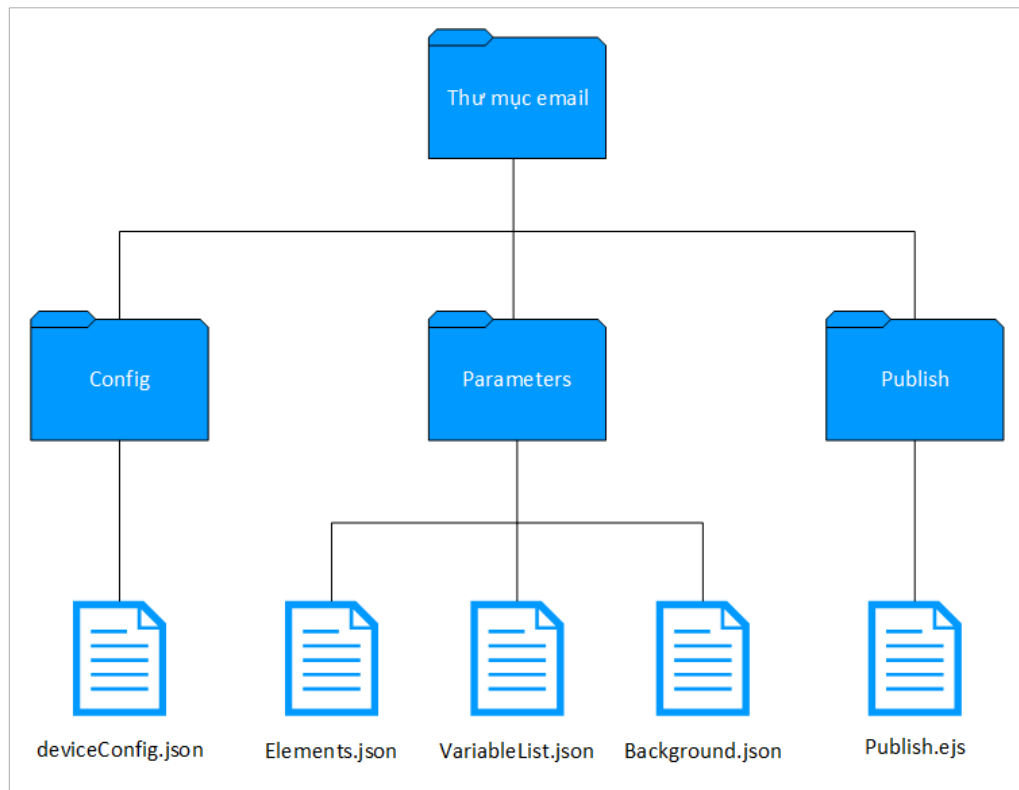
phản hồi của server từ socket **/device ID/publishSuccess** để xác nhận quá trình publish thành công.

```
var mainPage1 = document.getElementById('mainPage1').innerHTML;
var dashboard = document.getElementById('dashboard').innerHTML;
var _sendObject = {
    user: user,
    deviceID: deviceID,
    html: mainPage1,
    dashboard : dashboard,
    elements: elementHTML,
    variableList: variableList
}

var mainPageColor = $('#mainPage1')[0].style.background;
if (mainPageColor) mainPageColor = rgb2hex(mainPageColor);
var alarmPageColor = $('#alarm')[0].style.background;
if (alarmPageColor) alarmPageColor = rgb2hex(alarmPageColor);
var historyPageColor = $('#history')[0].style.background;
if (historyPageColor) historyPageColor = rgb2hex(historyPageColor);
var dashboardPageColor = $('#dashboard')[0].style.background;
if (dashboardPageColor) dashboardPageColor = rgb2hex(dashboardPageColor);
var backgroundObject = {
    mainPage: mainPageColor,
    alarmPage: alarmPageColor,
    historyPage: historyPageColor,
    dashboardPage: dashboardPageColor
}

socket.emit('/publish', _sendObject, backgroundObject);
$('#spinnerModal').modal('show');
socket.on('/' + deviceID + '/publishSuccess', function (data) {
    setTimeout(function () {
        $('#spinnerModal').modal('hide');
        var href = '/published/'+user+'/'+'Device_'+deviceID+'_publish.ejs';
        $('#publishedLink').attr('href', href)
        $('#successModal').modal('show');
    }, 3000);
});
```

Ở phía server, nó luôn lắng nghe socket **/publish** để nhận yêu cầu publish từ tất cả client. Khi có yêu cầu, server sẽ kiểm tra dữ liệu nhận về, nếu hợp lệ nó sẽ chép nội dung HTML publish sang thẻ <body> của file HTML mẫu. File HTML mẫu này giống như khung sườn của trang web, đã được cấu hình đặc biệt để tương thích với dữ liệu từ client. File HTML hoàn chỉnh được chép vào thư mục Publish bên trong thư mục email người dùng. Các file khác cũng được tạo ra để thuận tiện cho việc tái tạo đối tượng SCADA. Cấu trúc thư mục mỗi người dùng được thể hiện như hình 4.38.



Hình 4.38. Cấu trúc thư mục người dùng

Khi việc chép file hoàn tất, server thông báo cho client thông qua socket **/device ID/publishSuccess**, đồng thời sửa thuộc tính published trong file cấu hình gateway thành “true”, thêm thuộc tính link với nội dung là đường dẫn tới trang web publish của gateway.

```

socket.on('/publish', function (dataObj, backgroundObject) {
  var template = fs.readFileSync('../Server/views/publish-template.ejs').toString();
  var htmlObject = HTMLParser.parse(template);
  htmlObject.querySelector('#mainPage1').appendChild(dataObj.html);
  htmlObject.querySelector('#dashboard').appendChild(dataObj.dashboard);
  var fileName = 'Device_' + dataObj.deviceID + '_publish.ejs';

  fs.writeFileSync(path.resolve(databasePath, dataObj.user, 'Publish', fileName),
    htmlObject);

  fs.writeFileSync(path.resolve(databasePath, dataObj.user, 'Parameters', 'Device_'
    + dataObj.deviceID + '_Elements.json'), JSON.stringify(dataObj.elements, null, 4));

  fs.writeFileSync(path.resolve(databasePath, dataObj.user, 'Parameters', 'Device_'
    + dataObj.deviceID + '_VariableList.json'), JSON.stringify(dataObj.variableList,
    null, 4));

  fs.writeFileSync(path.resolve(databasePath, dataObj.user, 'Parameters', 'Device_'
    + dataObj.deviceID + '_Background.json'), JSON.stringify(backgroundObject, null,
    4));
  socket.emit('/') + dataObj.deviceID + '/publishSuccess', 1);
}

```

```
//Update published state in Config file
fs.readFile(path.resolve(databasePath, dataObj.user, 'Config', 'deviceConfig_' +
dataObj.deviceID + '.json'), function (err, data) {
  if (err) console.log(err);
  else {
    var deviceObj = JSON.parse(data);
    deviceObj.published = true;
    deviceObj.link = '/published/' + deviceObj.user + '/Device_' + deviceObj.deviceID
+ '_publish.ejs';
    fs.writeFile(path.resolve(databasePath, dataObj.user, 'Config', 'deviceConfig_'
+ dataObj.deviceID + '.json'), JSON.stringify(deviceObj, null, 4), function (err) {
      if (err) console.log(err);
      else console.log('Write success in ' + path.resolve(databasePath, dataObj.user,
'Config', 'deviceConfig_' + dataObj.deviceID + '.json'));
    })
  }
})
});
```

Khi người dùng đi đến trang web được publish, client gửi yêu cầu tham số về server thông qua socket **/reqPublishParameters**, nội dung gửi là email người dùng và device ID của gateway.

```
socket.emit('/reqPublishParameters', { user: $user, deviceID: $deviceID });
```

Server nhận được yêu cầu sẽ gửi ba file trong thư mục Parameters về client thông qua socket **/device ID/resPublishParameters**.

```
socket.on('/reqPublishParameters', function (userObject) {
  var elementFile = path.resolve(databasePath, userObject.user, 'Parameters',
'Device_' + userObject.deviceID + '_Elements.json');
  var variableFile = path.resolve(databasePath, userObject.user, 'Parameters',
'Device_' + userObject.deviceID + '_VariableList.json');
  var backgroundFile = path.resolve(databasePath, userObject.user, 'Parameters',
'Device_' + userObject.deviceID + '_Background.json');
  var elementObj = JSON.parse(fs.readFileSync(elementFile));
  var variableObj = JSON.parse(fs.readFileSync(variableFile));
  var backgroundObj = JSON.parse(fs.readFileSync(backgroundFile));
  socket.emit('/' + userObject.deviceID + '/resPublishParameters', {
htmlElements: elementObj, variableList: variableObj, background: backgroundObj });
});
```

Client nhận được dữ liệu sẽ tiến hành tái tạo đối tượng SCADA giống như trong chế độ vận hành thử đã trình bày.

Chương 5. XÂY DỰNG CHƯƠNG TRÌNH CHO GATEWAY

5.1. Giới thiệu

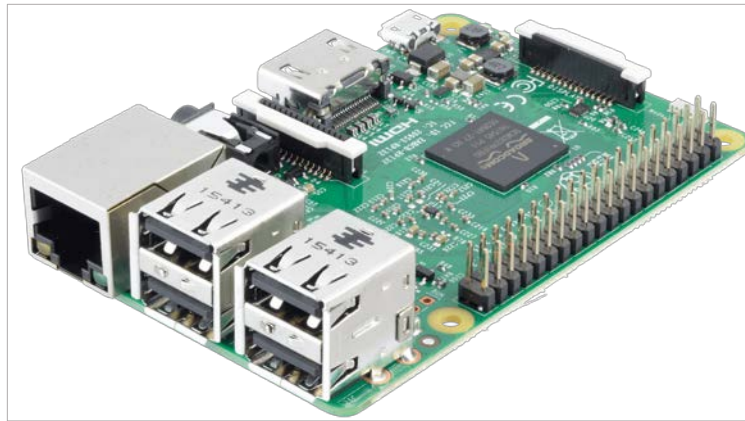
Nhiệm vụ của gateway là đọc dữ liệu từ PLC và gửi lên server theo giao thức MQTT. Việc chỉ định các biến sẽ đọc do người dùng thực hiện trên giao diện web. Server gửi yêu cầu về gateway, nó tự động cấu hình theo dữ liệu đó. Quá trình cấu hình hoàn tất, gateway bắt đầu thu thập dữ liệu và gửi lên theo chu kỳ đã cài đặt. Ngoài chức năng thu thập dữ liệu, gateway còn hỗ trợ server quản lý các cảnh báo của hệ thống. Việc này góp phần giảm tải cho server và tận dụng tối đa vi xử lý của gateway.

Lựa chọn phần cứng để thực hiện gateway phải được cân nhắc kỹ. Mặc dù NodeJS chỉ chạy single thread nhưng gateway phải thực hiện nhiều chức năng khác nhau, mỗi chức năng được cụ thể hoá bằng một module, nếu lựa chọn vi xử lý yếu sẽ không đáp ứng được tài nguyên. Với một vi xử lý đa lõi, ta có thể phân chia các nhiệm vụ trên các lõi khác nhau, cải thiện hiệu suất phần mềm. Raspberry Pi 3B và Simatic IoT2040 là những lựa chọn phù hợp ở thời điểm hiện tại.

Raspberry Pi

Raspberry Pi là máy tính đơn board có kích thước nhỏ (chỉ bằng thẻ tín dụng) nhưng mạnh mẽ và có đầy đủ tính năng của một máy tính chạy hệ điều hành Linux. Nó hỗ trợ hầu hết các ngoại vi thông dụng như USB cho chuột và bàn phím, HDMI cho màn hình, cổng audio 3.5mm cho loa và cổng Ethernet để kết nối mạng. Các phiên bản Raspberry sau này còn hỗ trợ cả giao tiếp không dây wifi, giúp Raspberry có thể thực hiện các tác vụ như một máy tính bình thường. Về phần mềm, Raspbian là hệ điều hành chính thức của Raspberry, đây là một distro dựa trên Debian, hỗ trợ hầu hết các phần mềm cơ bản. Ngoài ra, nó còn hỗ trợ nhiều hệ điều hành khác nhau để phục vụ nhu cầu đa dạng của người dùng như Ubuntu Mate, Ubuntu Core, Ubuntu Server, Window 10 IoT Core, OSMC, LibreELEC, PiNet, RISC OS,... Raspberry Pi 3 là một trong những thế hệ mới nhất của dòng Raspberry, sở hữu cấu hình mạnh mẽ như chip Broadcom BCM2835, CPU ARM Cortex A53 64 bit, GPU VideoCore IV, RAM DDR2 dung lượng 1GB và hỗ trợ hầu

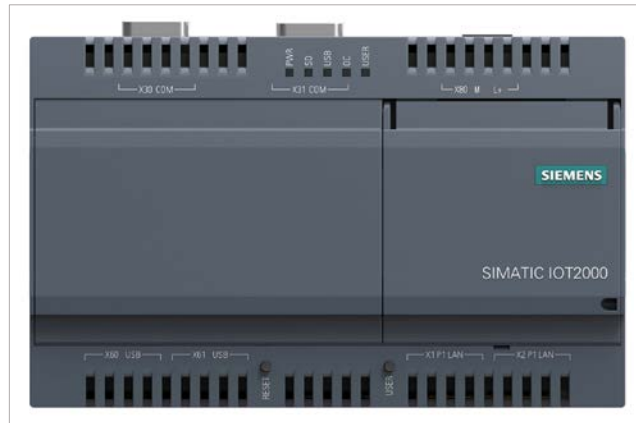
hết các ngoại vi cơ bản (GPIO, UART, SPI, I2C, USB, Ethernet, Wifi, Bluetooth).



Hình 5.1. Raspberry Pi 3

Simatic IoT2040

Simatic IoT2040 là một sản phẩm trong dòng Simatic IoT2000 của hãng Siemens (Đức). Đây là một nền tảng mở dùng cho mục đích thu thập, xử lý và truyền dữ liệu trong môi trường công nghiệp. Nó thật sự là một cầu nối lí tưởng giữa điện toán đám mây với môi trường sản xuất. Được thiết kế để làm việc 24/7, IoT2040 sẽ đảm bảo dữ liệu luôn được truyền liên tục về máy chủ. Về phần cứng, Simatic IoT2040 chạy con chip Quarkx1020 của Intel, với 1 GB RAM, 2 cổng Ethernet và 2 giao tiếp RS232/485. Với hai cổng Ethernet khác nhau, IoT2040 giúp tách rời mạng công nghiệp với đường truyền Internet thông thường, hạn chế các nguy cơ tấn công mạng. Ngoài ra, giao tiếp RS232/485 giúp nó có thể kết nối với các PLC thế hệ cũ, hoặc giao tiếp với Modbus RTU mà không cần cáp chuyển đổi. Về phần mềm, Simatic IoT2040 sử dụng hệ điều hành Yocto Linux, đây là một hệ điều hành mã nguồn mở cung cấp các công cụ giúp xây dựng các hệ thống nhúng hoạt động trên nhân Linux.



Hình 5.2. Simatic IoT2040

5.2. Các module chức năng

5.2.1. Module S7

Module S7 hỗ trợ giao tiếp với các dòng PLC Siemens qua giao thức S7-connection. Module này sử dụng gói thư viện nodes7 do NodeJS cung cấp.

Để khởi tạo kết nối với PLC, ta dùng hàm `initateConnection` và truyền các tham số như `port`, `rack`, `slot` và `địa chỉ PLC`. Mỗi PLC được quản lý bằng một object gồm các thuộc tính: `name` – tên PLC, `s7Node` - object đại diện cho PLC đó, `address` – địa chỉ của PLC.

```
function initConnection(plcName, plcAddress, plcPort, plcRack, plcSlot , callback)
{
    var newPLC = new nodes7;
    newPLC.initiateConnection({
        port : plcPort,
        rack : plcRack,
        slot : plcSlot,
        host : plcAddress,
        timeout : 10000, //10 seconds timeout
    }, function(err) {
        if (typeof(err) !== 'undefined') {
            console.log(err .red);
            callback(null)
        } else {
            callback({
                name : plcName,
                s7Node : newPLC,
                address : plcAddress,
            });
        }
    })
}
```

Một khó khăn khi sử dụng thư viện nodes7 là cách ghi tên biến không đồng nhất với quy định của Siemens. Ví dụ, địa chỉ MR4 của nodes7 tương ứng với MD4 kiểu real của

Siemens; DB1,REAL0 tương đương DB1.MD0 kiểu real,... Để thuận tiện cho người dùng, ta vẫn giữ nguyên quy tắc đặt tên của Siemens và thực hiện chuyển đổi địa chỉ Siemens sang địa chỉ nodes7.

```
//Convert Siemens address to nodes7 address
function modifyAddress(address , datatype) {
    var modifiedAddress = address;
    if (address.includes('DB')) {        //Datablock
        modifiedAddress = address.replace('.', ',');
        switch(datatype) {
            case 'Bool' : {
                if (address.includes('DBX')) modifiedAddress =
modifiedAddress.replace('DBX' , 'X');
                break;
            }
            case 'Double' : {
                if (address.includes('DBD')) modifiedAddress =
modifiedAddress.replace('DBD' , 'R');
                break;
            }
            case 'sInt' : {
                if (address.includes('DBW')) modifiedAddress =
modifiedAddress.replace('DBW' , 'I');
                else if (address.includes('DBB')) modifiedAddress =
modifiedAddress.replace('DBB' , 'B');
                break;
            }
            case 'uInt' : {
                if (address.includes('DBW')) modifiedAddress =
modifiedAddress.replace('DBW' , 'W');
                else if (address.includes('DBB')) modifiedAddress =
modifiedAddress.replace('DBB' , 'B');
                break;
            }
            case 'sDInt' : {
                if (address.includes('DBD')) modifiedAddress =
modifiedAddress.replace('DBD' , 'DI');
                break;
            }
            case 'uDInt' : {
                if (address.includes('DBD')) modifiedAddress =
modifiedAddress.replace('DBD' , 'DW');
                break;
            }
            case 'String' : {
                if (address.includes('DBS')) modifiedAddress =
modifiedAddress.replace('DBS' , 'S') + '.254';
            }
        }
    } else {        //PLC tags
        switch(datatype) {
            case 'Double' : {
                if (address.includes('MD')) modifiedAddress =
address.replace('MD' , 'MR');
                break;
            }
            case 'sInt' : {
                if (address.includes('MW')) modifiedAddress =
address.replace('MW' , 'MI');
                else if (address.includes('MB')) modifiedAddress =
address.replace('MB' , 'MI');
                break;
            }
        }
    }
}
```

```

        case 'sDInt' : {
            if (address.includes('MD')) modifiedAddress =
address.replace('MD' , 'MDI');
            break;
        }
    }
}
return modifiedAddress;
}

```

Thao tác đọc - ghi dữ liệu trong nodes7 gồm hai bước: khởi tạo danh sách và đọc/ghi danh sách đó. Việc khởi tạo chỉ thực hiện một lần duy nhất, nếu bỏ qua sẽ báo lỗi làm ngừng gateway.

```

// "item" can be a single variable address or an array of addresses
function initReadingList(s7Connection, items) {
    if (s7Connection) s7Connection.addItem(items);
}

// Read all data
function readAllData(s7Connection, callback) {
    s7Connection.readAllItems(function(err, values) {
        if (err) {
            console.log(err);
            callback({});
        } else {
            callback(values);
        }
    })
}

// Write value
function writeData(s7Connection, addresses, values) {
    s7Connection.writeItems(addresses, values, function(err) {
        if (err) console.log('Fail to set S7-Connection value' .red);
        else console.log('Set S7-Connection value succeeded' .green);
    });
}

```

5.2.2. Module OPC UA

OPC là mô hình truyền thông được chuẩn hoá cho các ứng dụng tự động hoá công nghiệp, giúp các thiết bị không cùng nhà sản xuất có thể trao đổi thông tin với nhau. OPC UA là phiên bản mới nhất của OPC nhưng có cơ chế hoạt động khác biệt. Thay vì phụ thuộc vào công nghệ của Microsoft như các phiên bản OPC cũ, OPC UA dựa vào kiến trúc hướng dịch vụ trên nền TCP/IP, do đó có thể hoạt động trên nhiều nền tảng hệ điều hành. Ngoài ra, OPC UA còn hỗ trợ mã hoá bằng giao thức SSL để bảo mật thông tin trong giao tiếp.

NodeJS cung cấp thư viện node-opcua để thực hiện các tác vụ liên quan đến OPC UA.

Thư viện cung cấp đầy đủ các chức năng của một OPC UA SDK. Thư viện quản lý các node OPC UA bằng những session đại diện, việc đọc – ghi dữ liệu đều được thực hiện qua các session này. Để thuận tiện hơn trong quá trình quản lý, tương tự như module S7, sau khi khởi tạo thành công, ta chia các node theo từng object với các thuộc tính: name – tên node; opcuaNode - session đại diện cho node, address – địa chỉ của node.

```
//Initialize connection
function initConnection(plcName, endpointUrl, callback) {
    var client = new opcua.OPCUAClient();
    var theSession;
    async.series([
        //STEP 1: CONNECT
        function (_callback) {
            client.connect(endpointUrl, function (err) {
                if (err) {
                    console.log("Cannot connect to endpoint: ", endpointUrl);
                } else {
                    console.log("Connected to: ", endpointUrl);
                }
                _callback(err);
            });
        },
        //STEP 2: CREATE SESSION
        function (_callback) {
            client.createSession(function (err, session) {
                if (!err) {
                    theSession = session;
                    console.log('Created session successfully');
                } else console.log('Create session failed');
                _callback(err);
            });
        },
        //FINAL STEP
        function(err, result) {
            if (err) console.log('Fail to create connection');
            else {
                console.log('Create connection successfully');
            }
            callback({
                name : plcName,
                opcuaNode : theSession,
                address : endpointUrl
            });
        }
    ])
}
```

Việc đọc dữ liệu khá đơn giản, tuy nhiên ghi dữ liệu lại khó khăn hơn vì phải đồng bộ kiểu dữ liệu của PLC với kiểu dữ liệu chuẩn của OPC UA.

```
//Read a variable
function readSingleValue(session, nodeId, callback) {
    session.readVariableValue(nodeId, function(err,dataValue) {
        if (err) {
            console.log('Reading error: ', err);
        }
        if (dataValue.value) callback(dataValue.value.value);
    });
}
```

```

        else callback(null);
    });
}

//Read multiple variables
function readMultipleValue(session, arrNodeId, callback){
    var returnArr = [];
    if (!Array.isArray(arrNodeId)) callback(returnArr);
    else {
        if (arrNodeId.length > 0) {
            var index = 0;
            async.whilst(
                //Check condition
                function(){
                    return index < arrNodeId.length;
                },
                //Executing function
                function(_callback) {
                    readSingleValue(session, arrNodeId[index] , function(data){
                        returnArr.push(data);
                        index++;
                        _callback();
                    });
                },
                //Finish function
                function() {
                    callback(returnArr);
                }
            )
        } else callback(returnArr); //If input array is null, return null
    }
}

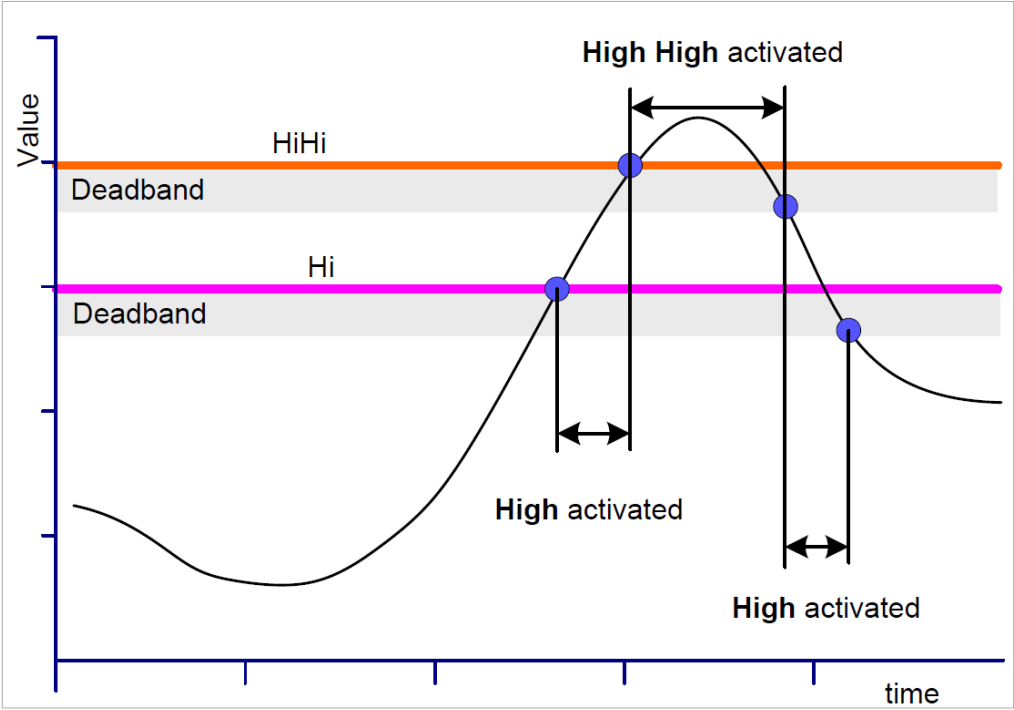
//Write variable
//Datatype: boolean, sInt, uInt, sDInt, uDInt, Double, String
function writeData(session, nodeId, dataType, value, callback) {
    var opcDataType;
    switch(dataType) {
        case 'Bool' : {
            opcDataType = opcua.DataType.Boolean;
            break;
        };
        case 'sByte' : {
            opcDataType = opcua.DataType.SByte;
            break;
        };
        case 'uByte' : {
            opcDataType = opcua.DataType.Byte;
            break;
        };
        case 'sInt' : {
            opcDataType = opcua.DataType.Int16;
            break;
        };
        case 'uInt' : {
            opcDataType = opcua.DataType.UInt16;
            break;
        };
        case 'sDInt' : {
            opcDataType = opcua.DataType.Int32;
            break;
        };
        case 'uDInt' : {
            opcDataType = opcua.DataType.UInt32;
            break;
        };
    }
}

```

```
var nodeToWrite = [{
  nodeId : nodeId,
  attributeId: opcua.AttributeIds.Value,
  value: {
    value: {
      dataType : opcDataType,
      value : value
    }
  }
}];
session.write(nodeToWrite, function(err, statusCodes) {
  if (err || (statusCodes[0].name != 'Good')) callback(false);
  else callback(true);
})
}
```

5.2.3. Module alarm

Có hai loại cảnh báo: Digital alarm và Analog alarm. Digital alarm là loại cảnh báo cơ bản dành cho các biến kiểu Boolean, kích hoạt khi giá trị là “true” và tắt khi giá trị là “false”. Analog alarm là loại cảnh báo cho các biến kiểu analog, sử dụng các mốc High High, High, Low, Low Low. Để cảnh báo chính xác, cần phải có một giá trị trễ gọi là deadband, cảnh báo chỉ được kích hoạt khi vượt qua ngưỡng trễ này. Quá trình kích hoạt cảnh báo High và High High được thể hiện qua hình 5.3, Low và Low Low thì ngược lại.



Hình 5.3. Quá trình kích hoạt alarm

Hàm checkAlarm của module thực hiện việc kiểm tra và kết luận cảnh báo theo sơ đồ hình 5.3.

```
function checkAlarm(alarmObject, deviceID, value) {
    // console.log('Alarm config object');
    // console.log(alarmObject);

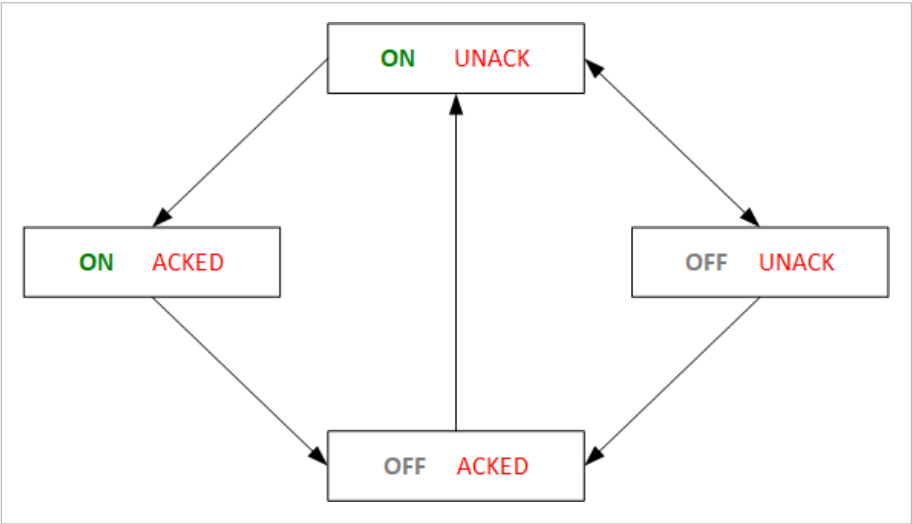
    var returnAlarm = {
        deviceID : deviceID,
        source : alarmObject.name,
        value : value,
        message : null,
        type : alarmObject.currentAlarmState,
        state : 'UNACK',
        timestamp : moment().format('YYYY-MM-DD HH:mm:ss'),
    }

    //console.log('Current alarm state: ' .magenta, returnAlarm.type .magenta);
    if (alarmObject.alarmType == 0) { //Digital alarm
        if (value) { //ON alarm
            returnAlarm.type = 'ON';
            returnAlarm.message = 'Value is ON';
        } else { //OFF alarm
            returnAlarm.type = 'OFF';
            returnAlarm.message = 'Value is OFF';
        }
    } else { //Analog alarm
        switch(alarmObject.currentAlarmState) {
            case 'LOLO' : {
                //LOLO -> HIHI
                if (value >= alarmObject.parameters.hihi) {
                    returnAlarm.type = 'HIHI';
                    returnAlarm.message = 'Value is TOO HIGH';
                    alarmObject.currentAlarmState = 'HIHI';
                }
                //LOLO -> HI
                else if (value >= alarmObject.parameters.hi) {
                    returnAlarm.type = 'HI';
                    returnAlarm.message = 'Value is HIGH';
                    alarmObject.currentAlarmState = 'HI';
                }
                //LOLO -> OK
                else if (value >= (alarmObject.parameters.lo +
alarmObject.parameters.deadband)) {
                    returnAlarm.type = 'OK';
                    returnAlarm.message = 'Value is OK';
                    alarmObject.currentAlarmState = 'OK'
                }
                //LOLO -> LO
                else if (value >= (alarmObject.parameters.lolo +
alarmObject.parameters.deadband)) {
                    returnAlarm.type = 'LO';
                    returnAlarm.message = 'Value is LOW';
                    alarmObject.currentAlarmState = 'LO';
                }
                //LOLO -> LOLO
                else {
                    returnAlarm.type = 'LOLO';
                    returnAlarm.message = 'Value is TOO LOW';
                    alarmObject.currentAlarmState = 'LOLO';
                }
                break;
            };
            case 'LO' : {
                //Thực hiện tương tự trường hợp LOLO
```



```
    }
    case 'OK' : {
        //Thực hiện tương tự trường hợp LOLO
    };
    case 'HI' : {
        //Thực hiện tương tự trường hợp LOLO
    };
    case 'HIHI' : {
        //Thực hiện tương tự trường hợp LOLO
    }
}
}
return returnAlarm;
}
```

Sau khi đánh giá được trạng thái cảnh báo hiện tại, cần so sánh với cảnh báo trước đó để quyết định đưa ra cảnh báo phù hợp. Việc quản lí cảnh báo được thực hiện theo lưu đồ như hình 5.4.



Hình 5.4. Máy trạng thái alarm

Các cảnh báo được lưu trữ trong một mảng và đợi người dùng xác nhận. Để tránh kích thước mảng quá lớn làm chậm hoạt động của gateway, ta giới hạn kích thước của mảng là 1000 phần tử. Khi người dùng xác nhận, cảnh báo đó được xoá khỏi mảng, mảng sắp xếp lại cấu trúc. Khi mảng đầy thì cảnh báo đợi lâu nhất bị xoá, nhường chỗ cho cảnh báo mới.

5.3. Lập trình chương trình cho gateway

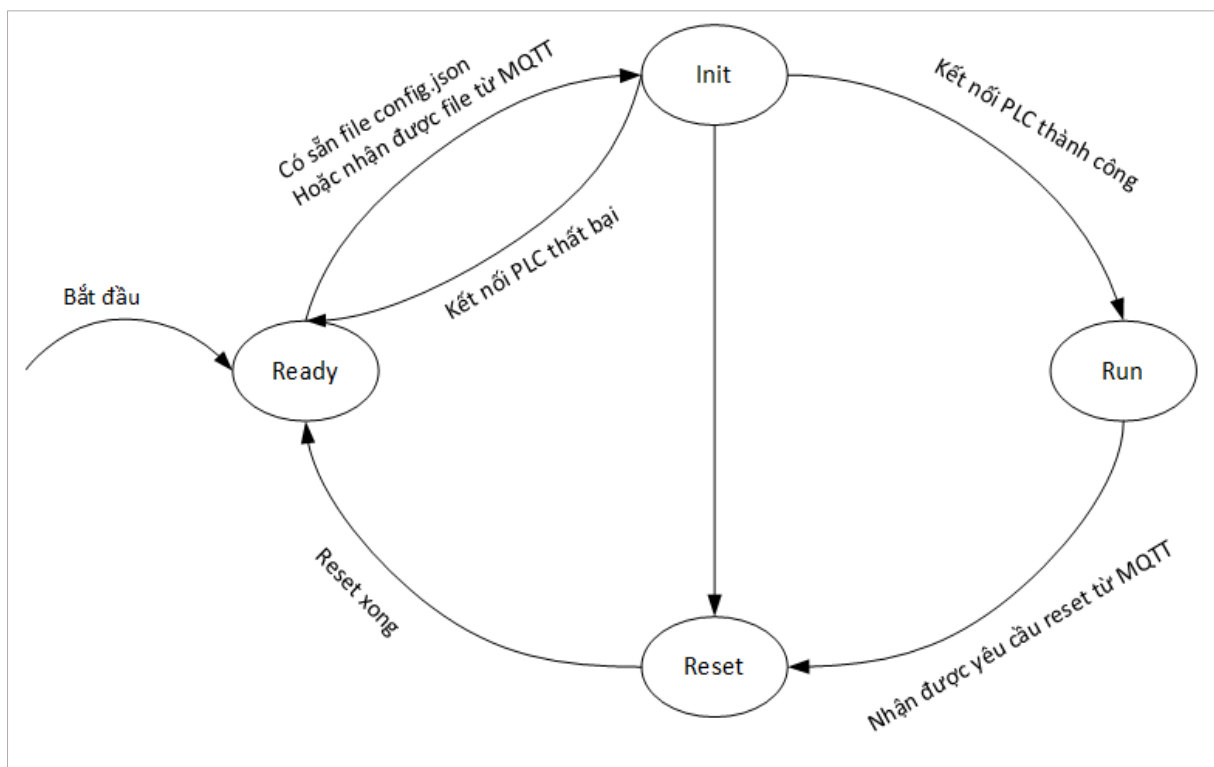
Gateway được thiết kế theo sơ đồ máy trạng thái như hình 5.5. Tuy nhiên, do cơ chế xử lí bất đồng bộ của NodeJS nên việc thực hiện máy trạng thái trên môi trường này

gặp nhiều khó khăn. Ta không kiểm soát được lệnh nào chạy trước, lệnh nào chạy sau, từ đó không kiểm soát được luồng của chương trình. Javascript-state-machine là một thư viện cho phép thực hiện máy trạng thái trên Javascript dễ dàng hơn nhờ các kỹ thuật lập trình đặc biệt. Để sử dụng thư viện này, ta phải khai báo các trạng thái của hệ thống, và chỉ định các hàm sẽ chạy trước khi quá trình chuyển trạng thái thành công.

```

/** là trạng thái bất kỳ
var fsm = new StateMachine({
  init: 'none',
  transitions: [
    { name: 'start', from: '*', to: 'ready' },
    { name: 'nextStep', from: 'ready', to: 'init' },
    { name: 'nextStep', from: 'init', to: 'run' },
    { name: 'reset', from: '*', to: 'reset' },
  ],
  methods: {
    onEnterReady: onEnterReadyFunction,
    onEnterInit: onEnterInitFunction,
    onEnterRun: onEnterRunFunction,
    onEnterReset: onEnterResetFunction
  }
})

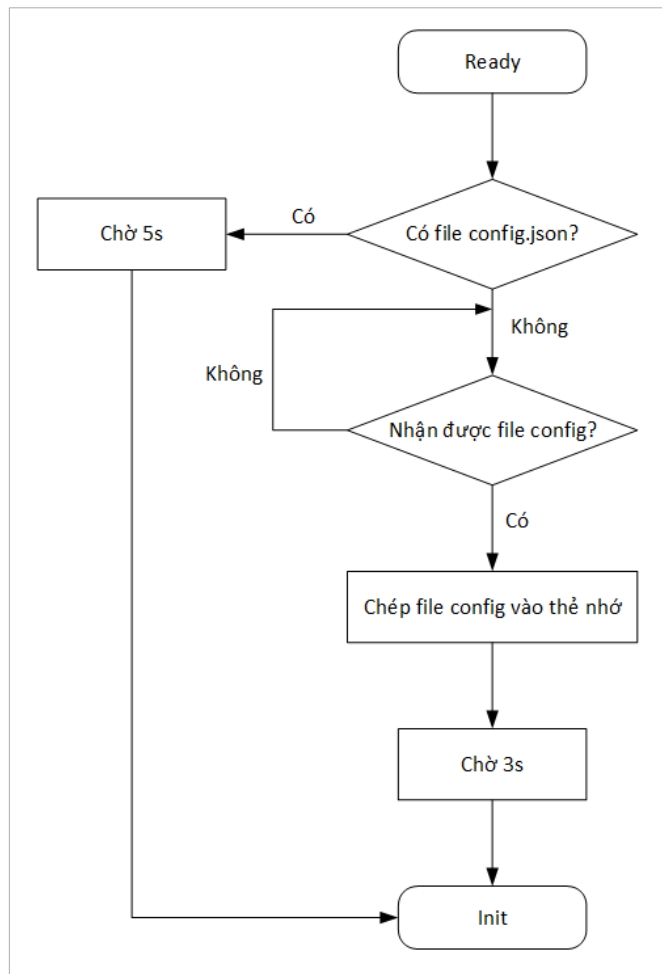
```



Hình 5.5. Máy trạng thái gateway

Máy trạng thái của gateway gồm bốn trạng thái là Ready, Init, Run và Reset.

- Ready là trạng thái đầu tiên khi bật nguồn. Khi bước vào trạng thái này, chương trình kiểm tra xem gateway đã được cấu hình hay chưa, bằng cách kiểm tra sự tồn tại của file config.json. Nếu có file thì đợi 5s và chuyển qua trạng thái Init. Thời gian 5s nhằm đảm bảo trạng thái Ready được “tải” đầy đủ, tránh phát sinh lỗi do thư viện javascript-state-machine gây nên. Nếu không có file config.js, gateway sẽ subscribe topic **/device ID/config** và đợi yêu cầu config. Khi có yêu cầu thì gateway lưu file lại, huỷ subscribe topic trên, đợi 3s và chuyển vào trạng thái Init. Thời gian 3s để đảm bảo các hàm bất đồng bộ đã được xử lý xong. Ngoài ra, tin nhắn từ topic config được bật cờ retain để phòng trường hợp gateway offline, khi online nó vẫn sẽ nhận được dữ liệu cấu hình. Do vậy, sau khi cấu hình xong phải “xóa” tin nhắn cũ đi bằng cách publish một tin nhắn không có nội dung (“”) lên chính topic config này. Lưu đồ giải thuật của trạng thái này được thể hiện như hình 5.6 và cụ thể hoá bằng đoạn chương trình bên dưới.



Hình 5.6. Lưu đồ trạng thái Ready

```

function onEnterReadyFunction() {
  if (fs.existsSync(configFilePath)) { //If config file existed
    setTimeout(function () { //Wait until the transition is completed
      fsm.nextStep();
    }, 5000);
  } else { //Config file not existed
    async.series([
      function (callback) {
        loggingInterval = setInterval(function () { console.log('State READY'.gray)
        }, 2000);
        console.log('State READY'.gray);
        callback();
      },
      function (callback) {
        mqttClient.on('message', function (topic, message) {
          if (topic == mqttConfigTopic) {
            deviceConfig = JSON.parse(message.toString());
            fs.writeFile(configFilePath, JSON.stringify(deviceConfig, null, 4),
            function (err) {
              if (err) console.log('Cannot save config file: '.red, err.red);
              else {
                console.log('Saved config file succesfully'.green);
                mqttClient.unsubscribe(mqttConfigTopic, function (err, packet) {
                  if (err) console.log('Topic unsubscription failed: '.red,
                  mqttConfigTopic.red);
                  else console.log('Topic unsubscription succeeded: '.green,
  
```

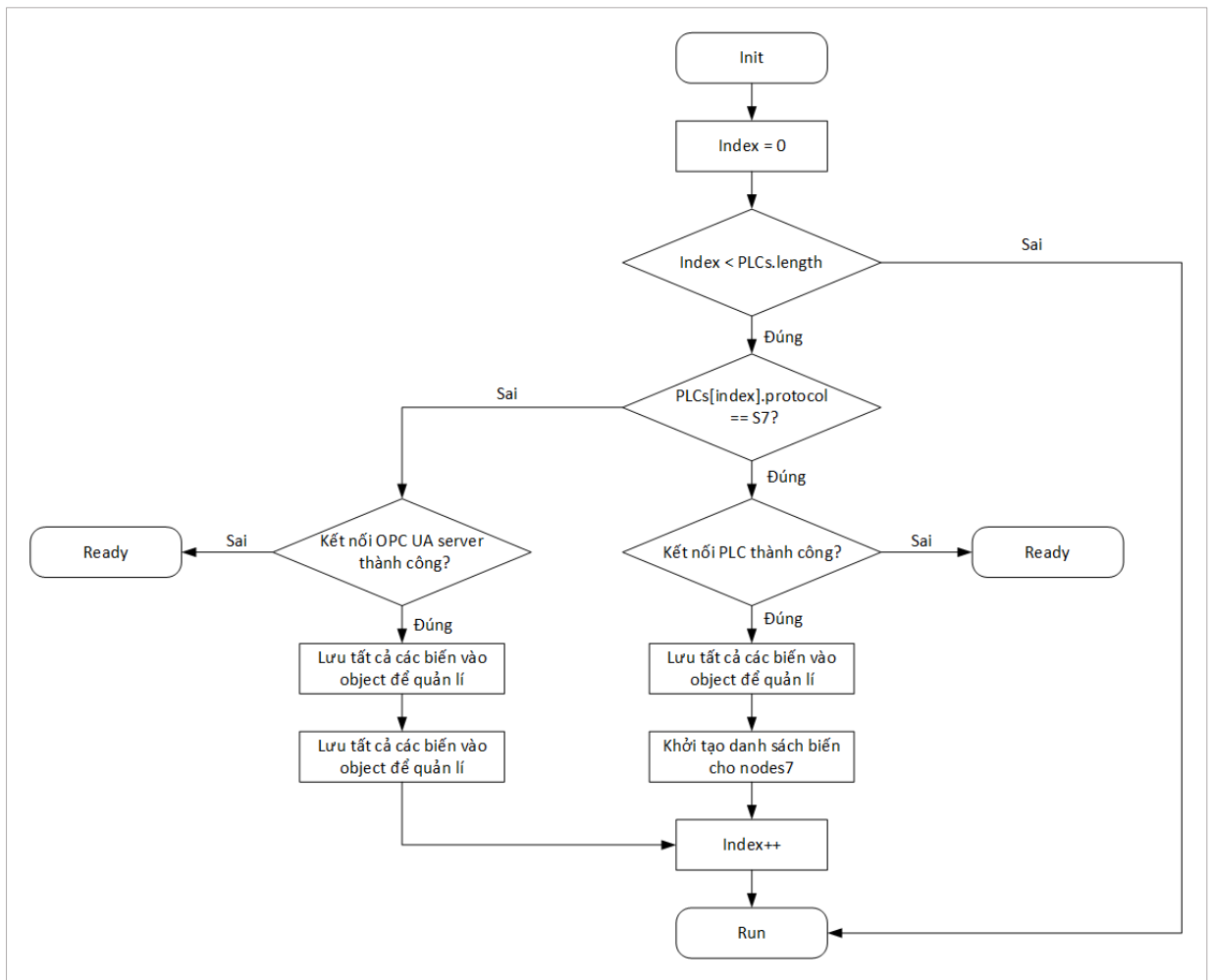
```

mqttConfigTopic.green);
    });
    clearInterval(loggingInterval);
    setTimeout(function () {
        fsm.nextStep(); //Go to Init step
    }, 3000);
    //Clear retained config MQTT message
    mqttClient.publish(mqttConfigTopic, '', retainOption);
}
    })
}
});

mqttClient.subscribe(mqttConfigTopic, null, function (err, granted) {
    if (err) console.log('Topic subscription failed: ', mqttConfigTopic.red);
    else console.log('Topic subscription succeeded: ', mqttConfigTopic.green);
    callback();
});
},
])
}
}
}

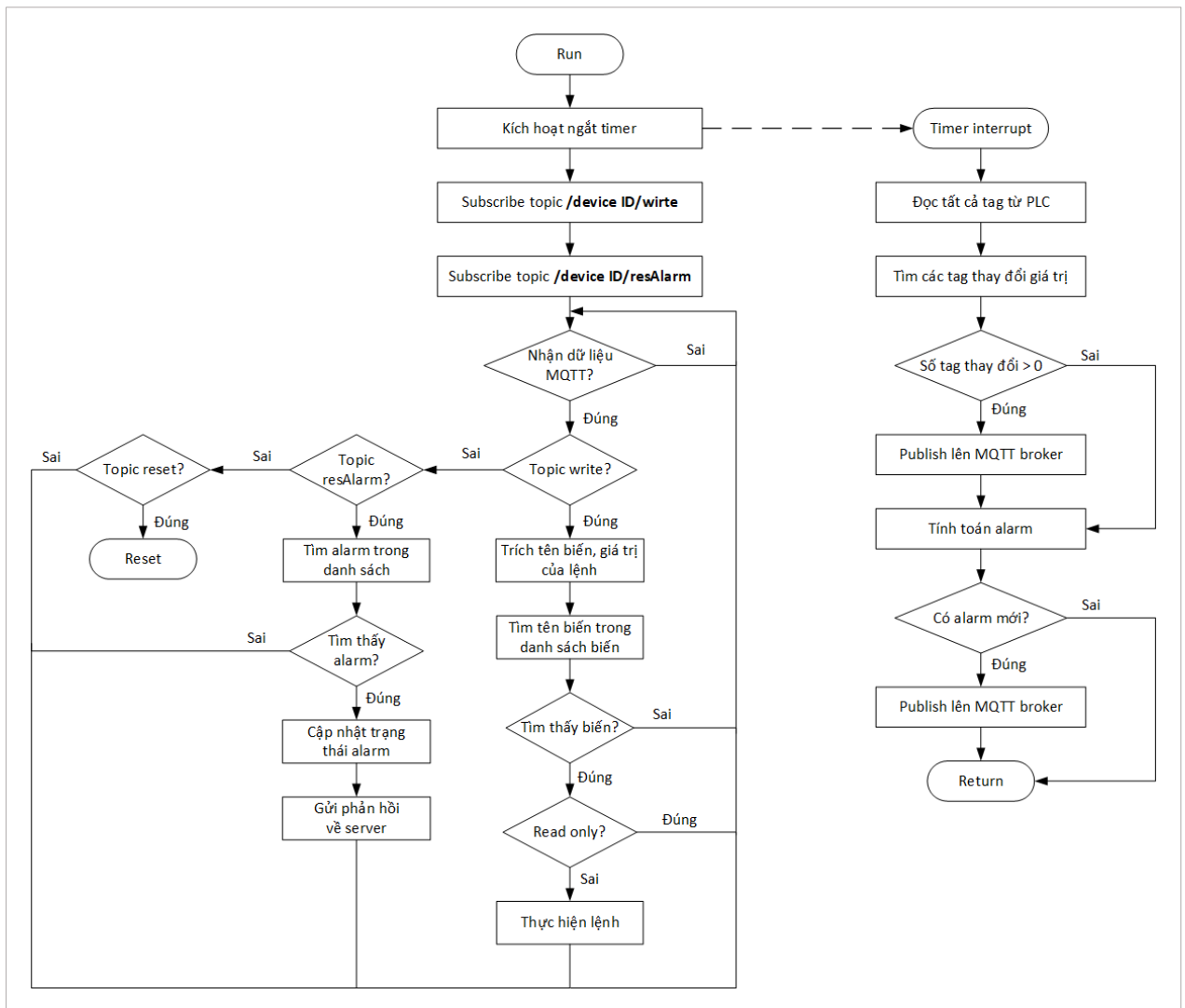
```

- Init là trạng thái khởi tạo kết nối giữa gateway và PLC. Tùy vào nội dung file config.json mà chương trình sẽ lựa chọn giao thức thích hợp kết nối với PLC. Nếu quá trình kết nối thành công sẽ đi đến trạng thái kế tiếp, nếu không sẽ về lại trạng thái Ready. Lưu đồ trạng thái Init được thể hiện trong hình 5.7.



Hình 5.7. Lưu đồ trạng thái Init

- Run là trạng thái gateway đã kết nối với PLC và truyền dữ liệu về server. Dữ liệu được đọc sau mỗi chu kì người dùng cài đặt. Sau khi đọc xong, dữ liệu được đưa vào hàm kiểm tra alarm để xử lí và gửi về server nếu có alarm. Nhằm tiết kiệm băng thông cho nhà máy và giảm tải cho server, chỉ có các biến thay đổi giá trị mới được gửi đi. Trong lúc đọc dữ liệu, gateway cũng lắng nghe topic **/device ID/write** để nhận lệnh điều khiển PLC, topic **/device ID/resAlarm** để xác nhận các alarm và topic **/device ID/reset** để reset gateway.



Hình 5.8. Lưu đồ trạng thái Run

Lưu đồ hình 5.8 được thực hiện cụ thể bằng đoạn chương trình sau:

```

function onEnterRunFunction() {
    //Subscribe write topic
    mqttClient.subscribe(mqttWriteTopic, null, function (err, granted) {
        if (err) console.log('Topic subscription failed: '.red,
            mqttWriteTopic.red);
        else console.log('Topic subscription succeeded: '.green,
            mqttWriteTopic.green);
    });

    //Subscribe response alarm topic
    mqttClient.subscribe(mqttResAlarmTopic, null, function (err, granted) {
        if (err) console.log('Topic subscription failed: '.red,
            mqttResAlarmTopic.red);
        else console.log('Topic subscription succeeded: '.green,
            mqttResAlarmTopic.green);
    });

    //Listen incoming messages
    mqttClient.on('message', function (topic, message) {

```

```

//MQTT write topic
if (topic == mqttWriteTopic) {
    var variableName = message.toString().replace(/\s/g, '').split('=')[0];
    var valueString = message.toString().replace(/\s/g,
    '').replace(variableName + '=', '');
    var valueNumber = Number(valueString);
    var value = isNaN(valueNumber) ? valueString : valueNumber;
    var varObj = getVariableInfo(variableName);
    if ((varObj != null)) {
        if (varObj.access == 'read') { //Only read
            console.log('No write access');
        } else { //Read/write
            if (varObj.protocol == s7Protocol) { //S7-connection
                for (var i = 0; i < plcObjects.s7.length; i++) {
                    if (plcObjects.s7[i].plc.address == varObj.plcAddress) {
                        if (varObj.dataType == 'Bool') value =
Boolean(value);
                        s7.writeData(plcObjects.s7[i].plc.s7Node,
s7.modifyAddress(varObj.variableAddress, varObj.dataType), value);
                        break;
                    }
                }
            } else { //OPC UA
                var plcIndex = 0;
                async.whilst(
                    function () {
                        return plcIndex < plcObjects.opcua.length;
                    },
                    function (callback) {
                        if (plcObjects.opcua[plcIndex].plc.address ==
varObj.plcAddress) {
                            opcua.writeData(plcObjects.opcua[plcIndex].plc.opcuaNode, varObj.variableAddress,
varObj.dataType, value, function (result) {
                                if (result) console.log('Set OPC UA data
succeeded'.green);
                                else console.log('Set OPC UA data
failed'.red);
                                break;
                            })
                        } else {
                            plcIndex++;
                            callback();
                        }
                    }
                )
            }
        }
    }
}

//MQTT response alarm topic
else if (topic == mqttResAlarmTopic) {
    var resAlarmObject = JSON.parse(message.toString());
    console.log('Response alarm object'.cyan);
    console.log(resAlarmObject);
    var alarmIndex = 0;
    async.whilst(
        function () {
            return alarmIndex < resAlarmObject.resAlarm.length;
        },
        function (callback) {
            var newAlarm = resAlarmObject.resAlarm[alarmIndex];
            if (newAlarm.type == 'ON' || newAlarm.type == 'OFF')
newAlarm.value = (newAlarm.value == 'true');
            else newAlarm.value = Number(newAlarm.value);
            findAlarm(newAlarm.source, newAlarm.type, function (result) {
                mqttClient.publish(mqttAlarmTopic, JSON.stringify(newAlarm,

```



```

null, 4), retainOption);
        if (result != -1) { //Update
            arrAlarm[result].state = newAlarm.state;
            arrAlarm[result].timestamp = newAlarm.timestamp;
        }
        alarmIndex++;
        callback();
    });
}

}

});

//Set interval for periodly data read
runInterval = setInterval(function () {
    async.series([
        function (callback) {
            readAllTags();
            callback();
        },
        function (callback) {
            alarmChecking();
            callback();
        }
    ], deviceConfig.period)
}

```

Trong đó, hàm compareArrays dùng để so sánh hai mảng và trả về các phần tử khác nhau của chúng.

```

function compareArrays(oldArr, newArr) {
    var returnIndex = [];
    if (oldArr.length == newArr.length) {
        for (var i = 0; i < oldArr.length; i++) {
            if (oldArr[i].value != newArr[i].value) returnIndex.push(newArr[i]);
        }
    }
    return returnIndex;
}

```

Chương 6. KẾT QUẢ THỬ NGHIỆM

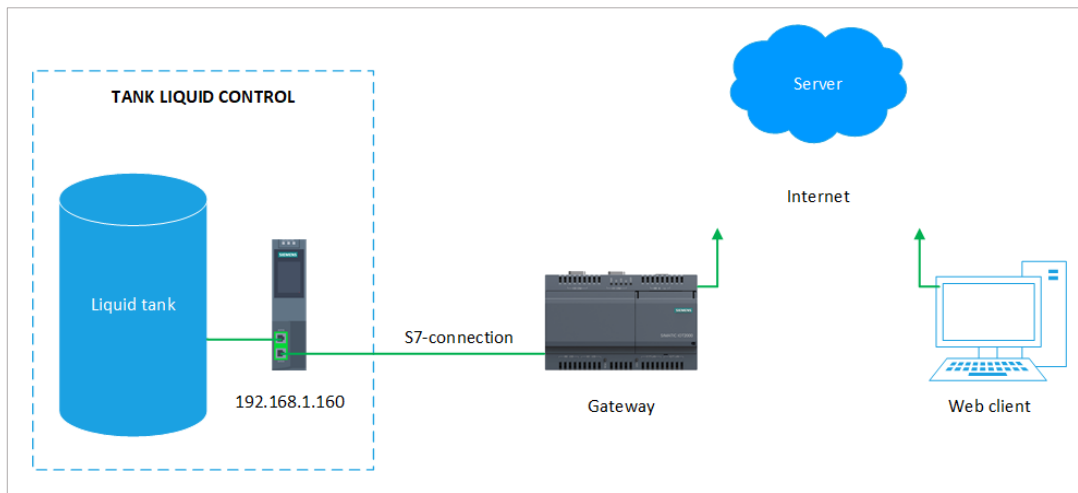
6.1. S7-connection

Hệ thống được thử nghiệm với mô hình điều khiển mức chất lỏng bằng PLC S7-1500. Do giới hạn về phần cứng, nên mô hình điều khiển mức chất lỏng sẽ được mô phỏng bằng phần mềm Factory IO như hình 6.1, PLC S7-1500 cũng mô phỏng bằng phần mềm PLCSim Advanced V2.1.



Hình 6.1. Mô hình điều khiển mức chất lỏng

Mô hình điều khiển mức chất lỏng gồm một tủ điện để điều khiển, một van bơm, một van xả, một bồn nước, một cảm biến đo mức nước và một cảm biến đo lưu lượng van xả. Hệ thống có hai chế độ vận hành: tự động và thủ công. Ở chế độ tự động, van bơm mở để bơm nước vào bồn cho đến khi đầy (3m), sau đó đóng van bơm và mở van xả cho đến khi mức nước về 0m. Ở chế độ thủ công, van bơm và van xả được điều khiển bởi hai công tắc trên tủ điện. PLC Siemens S7-1500 điều khiển hoạt động của hệ thống. Sơ đồ kết nối các phần tử được thể hiện trong hình 6.2.



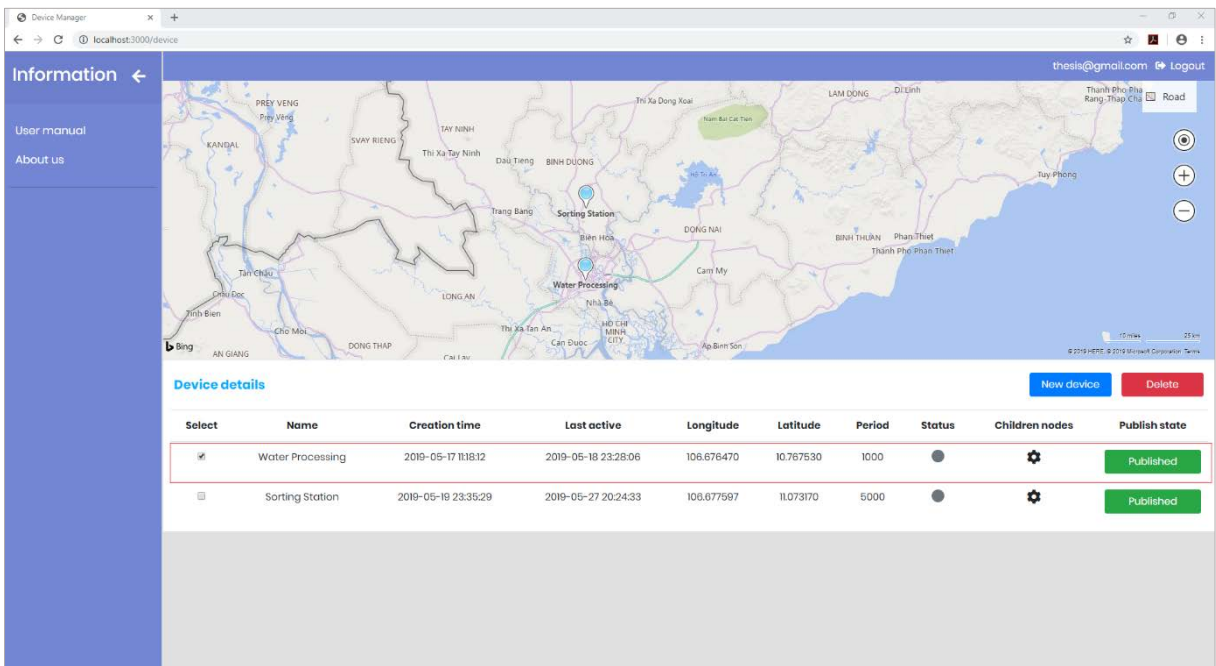
Hình 6.2. Sơ đồ hệ thống tank liquid control

Mapping giữa PLC S7-1500 với Factory IO như hình 6.3.

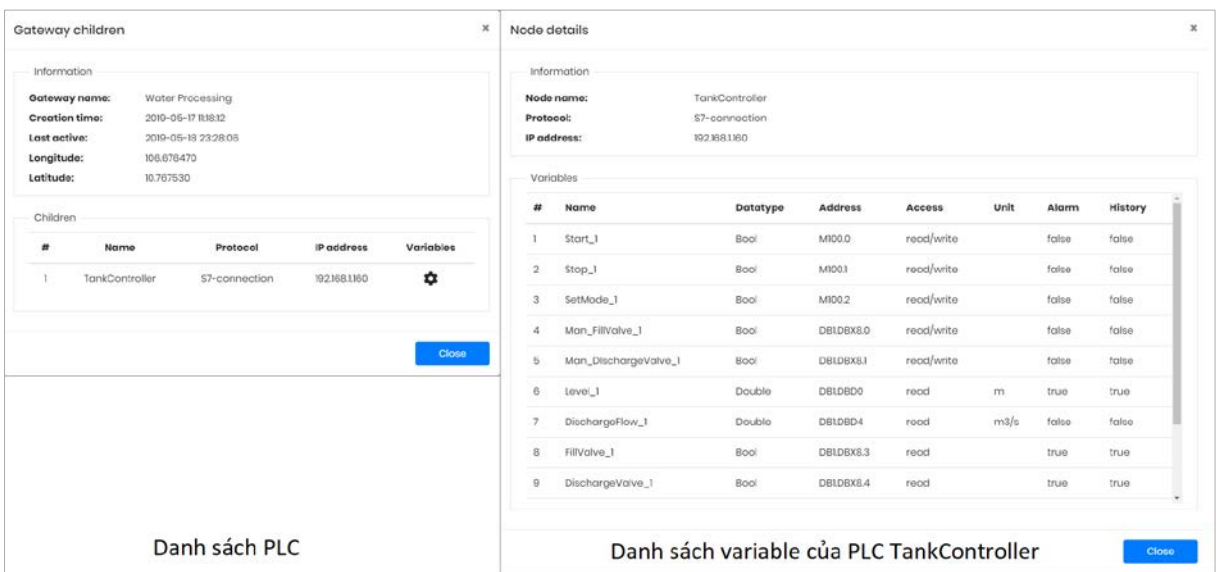
Host: 192.168.1.160			
Tank 1: Start	I512.0	Q0.0	Tank 1: Start light
Tank 1: Stop	I512.1	Q0.1	Tank 1: Stop light
Tank 1: Reset	I512.2	Q0.2	Tank 1: Reset light
	I512.3	Q0.3	
	I512.4	Q0.4	
	I512.5	Q0.5	
Mode 1: Manual (1)	I512.6	Q0.6	
	I512.7	Q0.7	
I: Fill (1)	I513.0	Q1.0	
II: Discharge (1)	I513.1	Q1.1	
	I513.2	Q1.2	
	I513.3	Q1.3	
	I513.4	Q1.4	
	I513.5	Q1.5	
	I513.6	Q1.6	
	I513.7	Q1.7	
Tank 1: Level Meter	IW0 (INT)	(INT) QW2	Tank 1: Fill valve
Tank 1: Flow meter	IW2 (INT)	(INT) QW4	Tank 1: Discharge valve
	IW4	QW6	
	IW6	QW8	
	IW8	(INT) QW10	Tank 1 Level
	IW10	QW12	
	IW12	QW14	
	IW14	QW16	

Hình 6.3. Mô hình điều khiển mức chất lỏng - mapping

Cấu hình và các giao diện web SCADA được thể hiện từ hình 6.4 đến hình 6.9.



Hình 6.4. Giao diện quản lý gateway



Hình 6.5. Cấu hình gateway



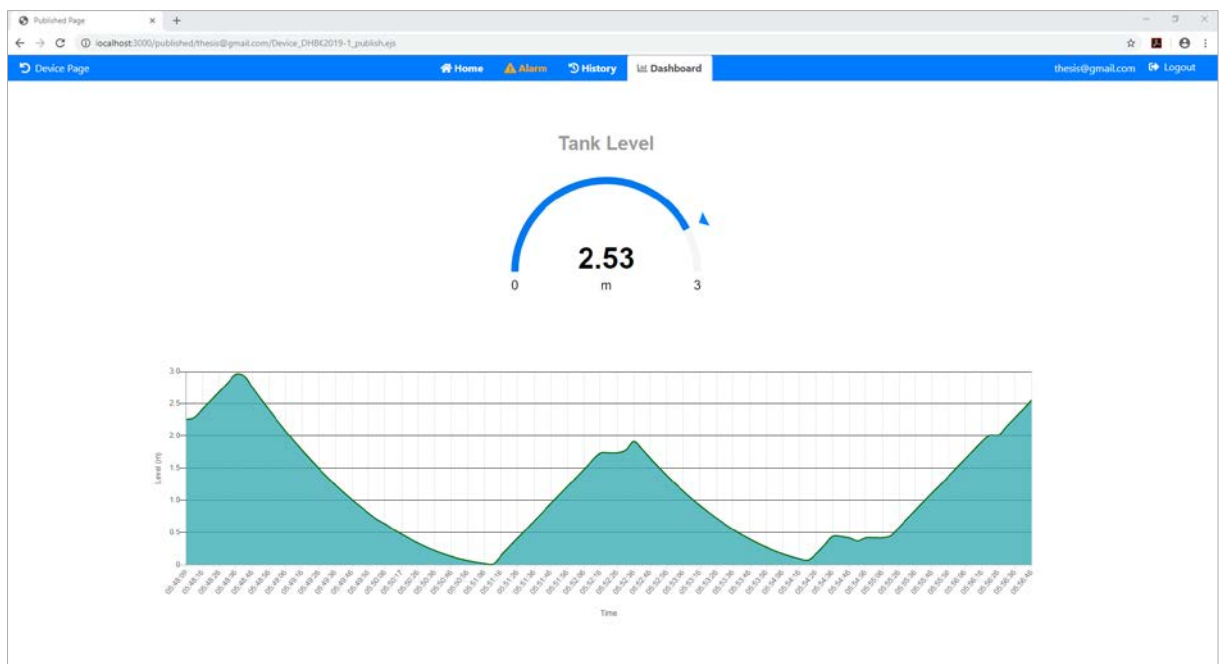
Hình 6.6. Mô hình điều khiển mức nước – chế độ Auto



Hình 6.7. Mô hình điều khiển mức nước – Chế độ Manual

Published Page							
Device Page							
Home Alarm History List Dashboard							
thesia@gmail.com Logout							
Acknowledge							
#	Date	Time	Source	Value	Message	Type	State
<input type="checkbox"/>	5/28/2019	1:40:17 PM	Level_1	2.879774570465088	Value is TOO HIGH	HIHI	UNACK
<input type="checkbox"/>	5/28/2019	1:40:12 PM	Level_1	2.74609375	Value is HIGH	HI	UNACK
<input type="checkbox"/>	5/28/2019	1:37:02 PM	Level_1	0.7622612714767456	Value is LOW	LO	UNACK
<input type="checkbox"/>	5/28/2019	1:40:07 PM	Level_1	0.36132824420928955	Value is TOO LOW	LOLO	ACKED
<input type="checkbox"/>	5/28/2019	1:40:07 PM	FillValve_1	true	Value is ON	ON	ACKED
<input type="checkbox"/>	5/28/2019	1:40:07 PM	DischargeValve_1	false	Value is OFF	OFF	ACKED
<input type="checkbox"/>	5/28/2019	1:40:07 PM	Level_1	2.2965495586395264	Value is OK	OK	ACKED
<input type="checkbox"/>	5/28/2019	1:40:07 PM	Level_1	2.6119790077209473	Value is HIGH	HI	ACKED
<input type="checkbox"/>	5/28/2019	1:40:07 PM	DischargeValve_1	true	Value is ON	ON	ACKED
<input type="checkbox"/>	5/28/2019	1:40:07 PM	FillValve_1	false	Value is OFF	OFF	ACKED
<input type="checkbox"/>	5/28/2019	1:40:07 PM	Level_1	2.8199868202209473	Value is TOO HIGH	HIHI	ACKED

Hình 6.8. Mô hình điều khiển mức nước - Alarm

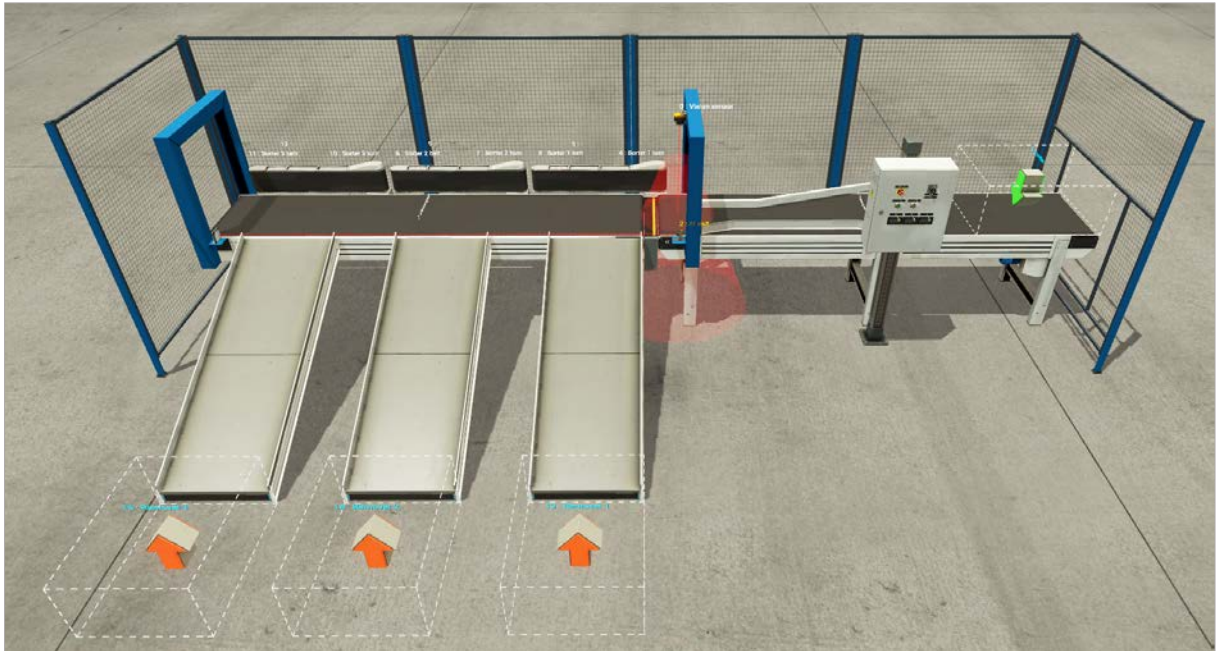


Hình 6.9. Mô hình điều khiển mức nước – Dashboard

6.2. OPC UA

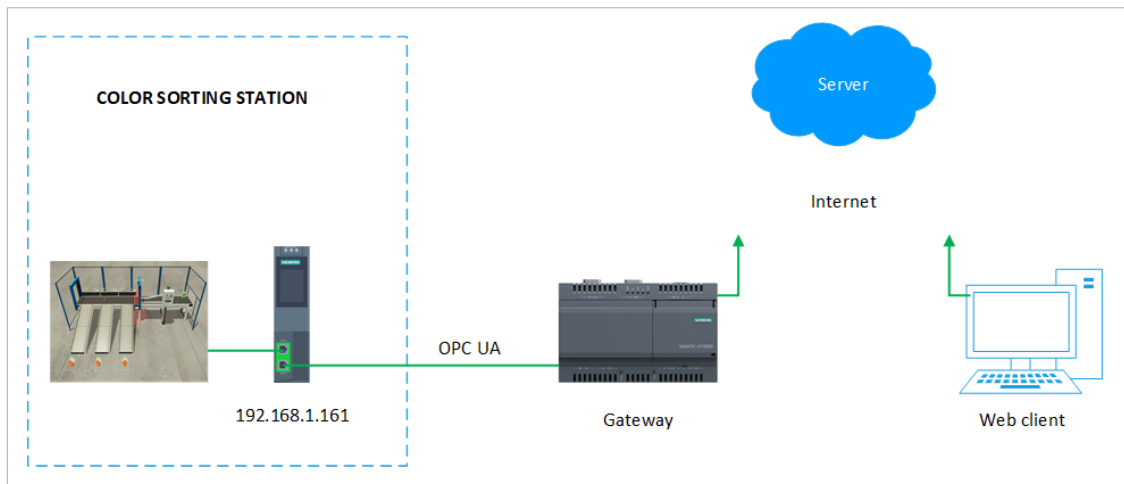
Giao thức OPC UA được kiểm nghiệm bằng mô hình phân loại sản phẩm theo màu như hình 6.10. Hệ thống gồm một băng tải đưa sản phẩm vào là Entry Conveyor, một băng

tải đưa sản phẩm ra là Exit Conveyor, ba cánh tay để cản sản phẩm, ba băng tải gắn trên cánh tay để sản phẩm đi nhanh hơn, một vision sensor để phân loại sản phẩm, một tủ điện để điều khiển và ba thanh trượt đưa sản phẩm đã phân loại đi đến dây chuyền khác.



Hình 6.10. Mô hình phân loại sản phẩm theo màu

Khi có sản phẩm đến, vision sensor phân biệt và báo về PLC. Tùy vào mã màu mà PLC ra lệnh bật/ tắt các cánh tay và băng tải tương ứng. Nếu đang có sản phẩm bên Exit Conveyor, Entry conveyor sẽ tắt khi sản phẩm đi vào vùng nhận biết của vision sensor. Chương trình điều khiển được thực hiện trên PLC S7-1500, gateway giao tiếp với PLC thông qua giao thức OPC UA (S7-1500 là OPC UA server). Sơ đồ kết nối các phần tử như hình 6.11.



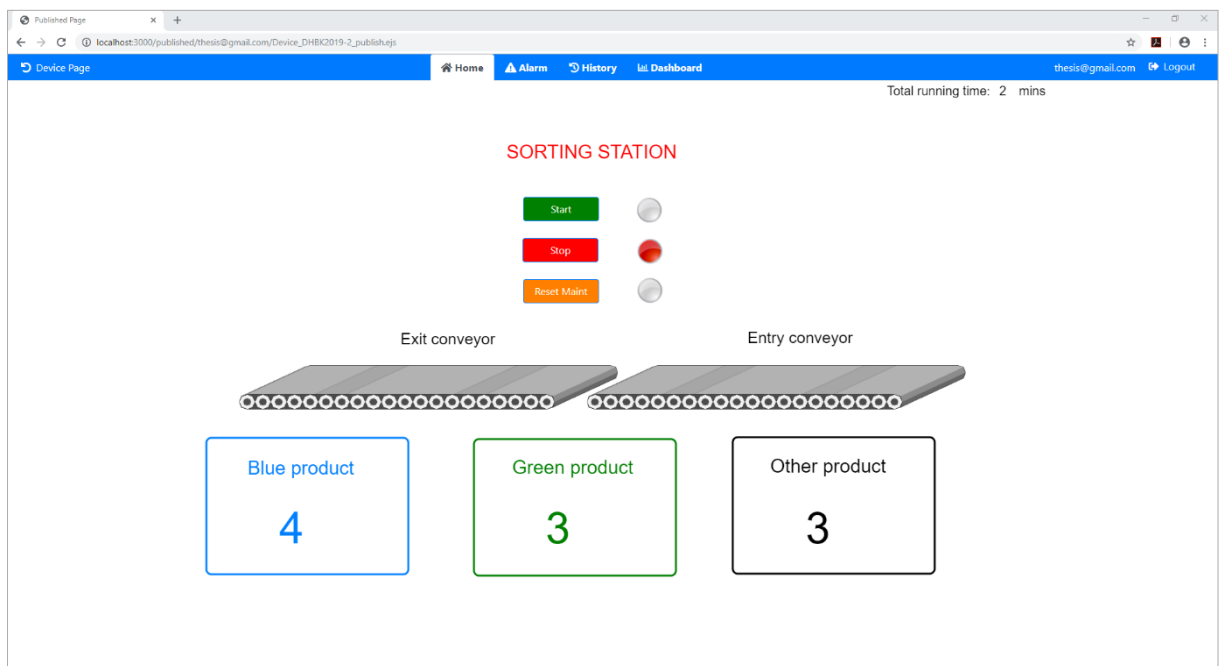
Hình 6.11. Mô hình phân loại sản phẩm theo màu

Mapping giữa PLC S7-1500 và Factory IO như hình 6.12.

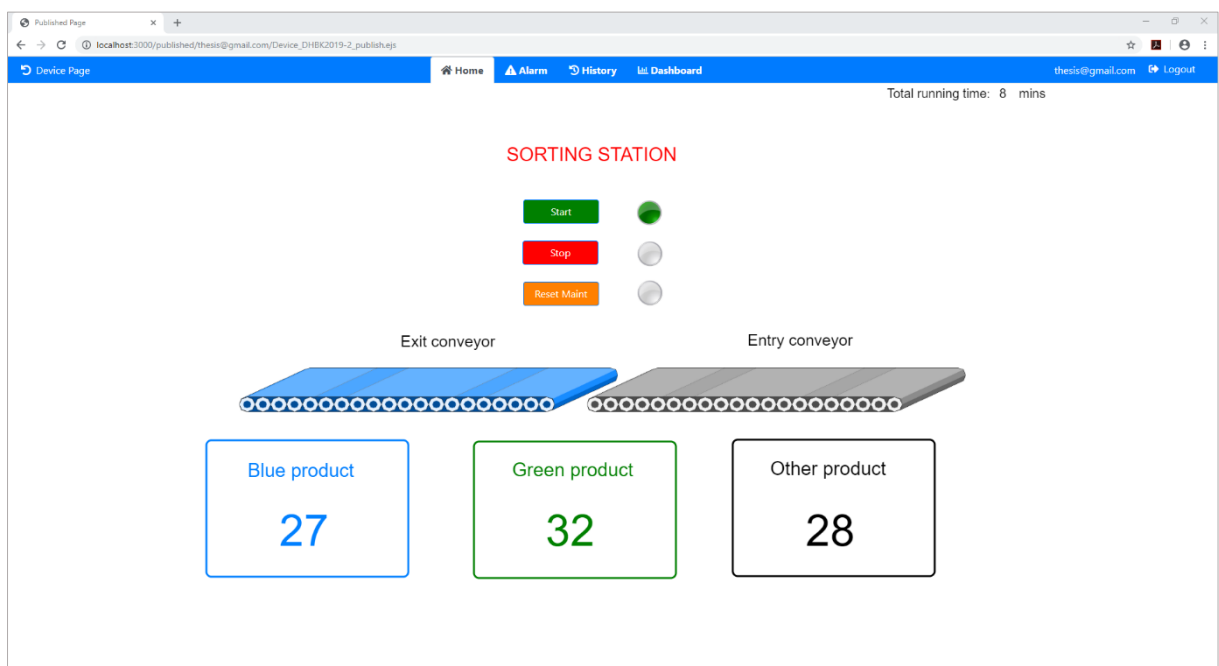
Host: 192.168.1.161			
Start	I512.0	Q0.0	Start light
Stop	I512.1	Q0.1	Reset light
	I512.2	Q0.2	
At exit	I512.3	Q0.3	Entry conveyor
	I512.4	Q0.4	Exit conveyor
	I512.5	Q0.5	Sorter 1 turn
	I512.6	Q0.6	Sorter 2 turn
	I512.7	Q0.7	Sorter 3 turn
	I513.0	Q1.0	Sorter 1 belt
	I513.1	Q1.1	Sorter 2 belt
	I513.2	Q1.2	Sorter 3 belt
	I513.3	Q1.3	
	I513.4	Q1.4	
	I513.5	Q1.5	
	I513.6	Q1.6	
	I513.7	Q1.7	
Vision sensor	IW0 (INT)	(INT) QW2	Blue
	IW2	(INT) QW4	Green
	IW4	(INT) QW6	Other

Hình 6.12. Mô hình phân loại sản phẩm theo màu – mapping

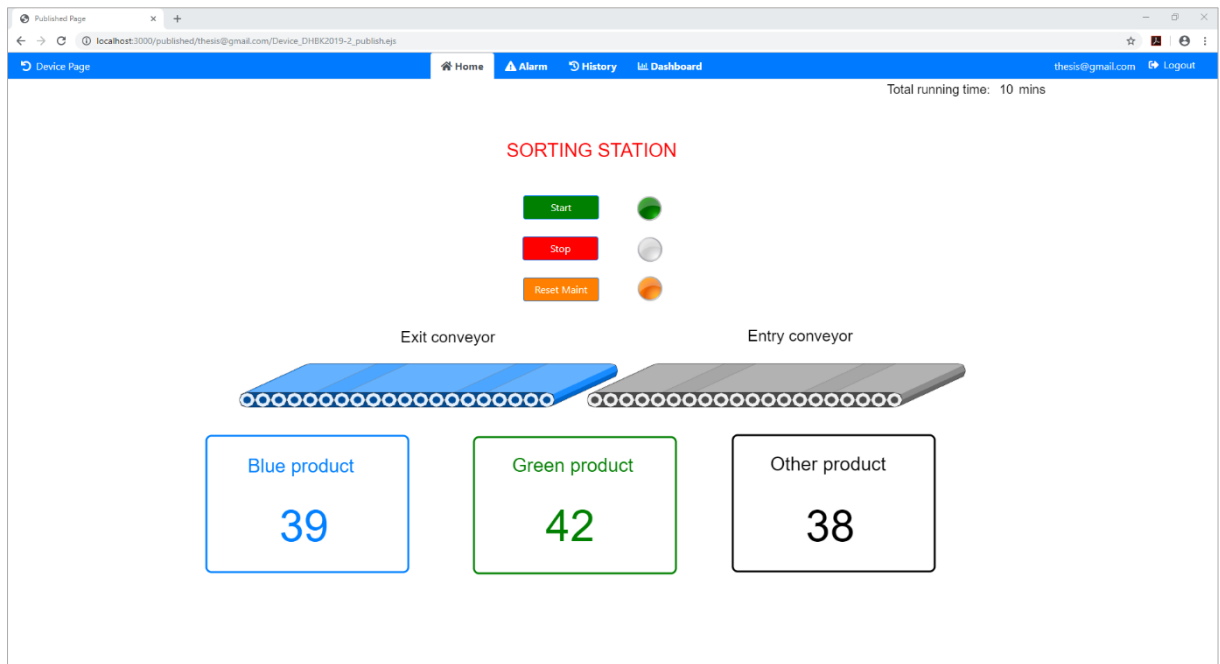
Cấu hình và các giao diện web SCADA được thể hiện từ hình 6.13 đến hình 6.16.



Hình 6.13. Trạng thái Stop



Hình 6.14. Trạng thái Run



Hình 6.15. Cảnh báo khi làm việc quá thời gian quy định

Published Page							
localhost:3000/published/thesis@gmail.com/Device_DHBK2019-2_publish.ejs							
Device Page							
Home Alarm History Dashboard							
thesis@gmail.com Logout							
Total running time: 10 mins							
SORTING STATION							
Start				●			
Stop				●			
Reset Maint				●			
Exit conveyor				Entry conveyor			
Blue product				Green product			
39				42			
Other product				38			

Hình 6.16. Màn hình cảnh báo

Chương 7. KẾT LUẬN

Luận văn đã thực hiện được nhiệm vụ đề ra là xây dựng một nền tảng SCADA trên nền web, cùng với thiết bị gateway có thể tự cấu hình. Giao diện và cách sử dụng được mô phỏng theo các phần mềm SCADA thông dụng nên người dùng có thể nhanh chóng làm quen. Quá trình giám sát và điều khiển được thực hiện hoàn toàn từ xa thông qua Internet. Sản phẩm có thể ứng dụng vào thực tế, góp phần thay đổi quan niệm về SCADA truyền thống. Tuy nhiên, do giới hạn về thời gian nên luận văn vẫn còn một số hạn chế:

- Chưa hỗ trợ chuẩn truyền thông Modbus TCP/ RTU và driver cho các dòng PLC nổi tiếng khác.
- Chưa hỗ trợ phân quyền truy cập SCADA.
- Thiếu các đối tượng điều khiển nâng cao, chưa hỗ trợ phân trang.

Hướng phát triển của luận văn là thay thế giao thức MQTT bằng một giao thức hiện đại hơn như AMQP hay OPC UA (phiên bản OPC UA thế hệ tiếp theo sẽ hỗ trợ mô hình Pub – Sub như MQTT), nhằm tăng hiệu suất và bảo mật hệ thống. Ngoài ra, gateway và server phải được kiểm nghiệm trong môi trường công nghiệp, với lượng lớn người dùng để xem xét khả năng chịu tải, lựa chọn lại phần cứng hoặc giải pháp công nghệ khi cần thiết. Bên cạnh đó, cần mở rộng khả năng giao tiếp của gateway với nhiều chuẩn truyền thông khác nhau, đặc biệt là với PLC của các hãng nổi tiếng.

TÀI LIỆU THAM KHẢO

Tài liệu trong nước

- [1] V. T. H. Phương, "Internet of Things trong công nghiệp," 2018, TPHCM, 2018.
- [2] P. H. H. Quân, "Nghiên cứu và phát triển hệ thống SCADA dựa trên điện toán đám mây," TPHCM, 2016.
- [3] Đ. N. T. Tín, "Thiết kế kiến trúc và thực thi hệ thống SCADA trên nền web," TPHCM, 2015.

Tài liệu nước ngoài

- [1] O. F. Claret, "Study of an innovative scada system," Escola, 2016.
- [2] D. Korsak, "SCADA system implementation in WWW applications," 2015.
- [3] J. M. Lynch, "An Internet based SCADA system," Southern Queensland, 2005.
- [4] H. Abbas, "Review on the Design of web based SCADA system based on OPC DA protocol," Qena Paper, Egypt, 2011.
- [5] A. N. MYINT, M. M. LATT, W. K. MOE and THEINGI, "Design and application of SCADA based control system for filling process," Mandalay Technological University, 2005.
- [6] E. H. A. A. Ahmed, "Efficient Web-Based SCADA System," Aswan, Egypt, 2011.
- [7] "W3School CSS tutorial," [Online]. Available: <https://www.w3schools.com>.
- [8] "W3School HTML tutorial," [Online]. Available: <https://www.w3schools.com>.
- [9] "W3School Bootstrap tutorial," [Online]. Available: <https://www.w3schools.com>.
- [10] "W3School Javascript tutorial," [Online]. Available: <https://www.w3schools.com>.

- [11] "W3School JQuery tutorial," [Online]. Available: <https://www.w3schools.com>.
- [12] "W3School NodeJS tutorial," [Online]. Available: <https://www.w3schools.com>.
- [13] OASIS, MQTT v3.1.1 Specifications.
- [14] "SCADA system," [Online]. Available: <https://electricalfundablog.com/scada-system-components-architecture/>.
- [15] "Mosquitto documentation," [Online]. Available: <https://mosquitto.org/api/files>.