

ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH
★ ★ ★



LUẬN VĂN TỐT NGHIỆP

**XÂY DỰNG NHÀ THÔNG MINH
TƯƠNG TÁC VỚI THIẾT BỊ DI ĐỘNG**

Giáo viên hướng dẫn: TS. Nguyễn Đức Dũng

Giáo viên phản biện: ThS. Vương Bá Thịnh

Sinh viên thực hiện:	Nguyễn Thành Tâm	51203264
	Nguyễn Thanh Tùng	51204401
	Bùi Quang Vinh	51204518

TP. HỒ CHÍ MINH, 11/2016

Tóm tắt luận văn

Nhà thông minh là khái niệm đã không còn quá xa lạ với chúng ta mà hiện tại đã trở thành xu thế công nghệ tất yếu, là tiêu chuẩn của nhà ở hiện đại. Chúng ta đang sống trong một thế giới của công nghệ. Một ngôi nhà hoàn hảo không chỉ sang trọng trong thiết kế, mà còn phải mang lại cảm giác thoải mái, tiện nghi và còn có khả năng hiểu được chủ nhân. Cuộc đua nhà thông minh đã và đang có những bước phát triển ở Việt Nam, có thể kể đến như hệ thống nhà thông minh Bkav. Tuy nhiên, hiện tại ở Việt Nam, nhà thông minh chưa được phổ biến rộng, một phần vì chi phí lắp đặt, chi phí xây dựng khá cao. Năm bắt xu hướng đó, ở luận văn này của nhóm với đề tài “Xây dựng ứng dụng nền tảng web hoặc di động tương tác với nhà thông minh qua các kịch bản” giúp chúng ta có thể tự tạo nên một hệ thống nhà thông minh nhỏ, có khả năng quản lý tự động một số thiết bị trong gia đình với mức chi phí phù hợp.

Về phương pháp thực hiện, nhóm chia việc xây dựng hệ thống thành các phần: giao diện, server và bộ phận tương tác thiết bị trên Raspberry Pi. Giao diện giúp người dùng quản lý thiết bị, kịch bản trong nhà. Phía server làm nhiệm vụ lưu trữ, kiểm tra, xử lý, vận hành kịch bản từ phía người dùng yêu cầu. Bộ phận tương tác có chức năng cung cấp cho server những dịch vụ (services) để điều khiển các thiết bị được gắn trên Raspberry Pi.

Hệ thống được thử nghiệm cùng với một số loại thiết bị như: cảm biến nhiệt độ, cảm biến ánh sáng, cảm biến vật thể chuyển động, cảm biến khí gas, bóng đèn led, còi hú. Theo đánh giá của nhóm, hệ thống có thể xử lý tốt trong một số trường hợp kịch bản đơn giản, tuy nhiên vẫn theo sát nhu cầu thực tế. Trong tương lai, hệ thống có thể tiếp tục được mở rộng. Nhóm hi vọng đề tài này có thể cung cấp những thông tin hữu ích cho các nghiên cứu liên quan sau này.

Mục lục

Tóm tắt luận văn	i
Danh mục bảng	iv
Danh mục hình ảnh	vi
1 Giới thiệu	1
1.1 Tổng quan	1
1.2 Các thành tựu, công trình liên quan nhà thông minh	1
1.3 Nhà thông minh với Raspberry Pi	2
1.4 Mục tiêu và phạm vi đề tài	2
1.5 Cấu trúc luận văn	3
2 Phân tích và thiết kế hệ thống	4
2.1 Các yêu cầu chức năng	4
2.2 Các yêu cầu phi chức năng	4
2.3 Sơ đồ usecase	4
2.4 Tổng quan hệ thống	7
3 Back-end	9
3.1 Thiết kế cơ sở dữ liệu	9
3.2 Thiết kế và kiến trúc hệ thống back-end	10
3.2.1 Spring framework	11
3.2.2 RESTful Web Service	12
3.2.3 Hibernate Framework	14
3.2.4 Kiến trúc back-end	16
4 Module điều khiển thiết bị	18
4.1 Tổng quan về Raspberry Pi	18
4.2 Điều khiển thiết bị sử dụng GPIO	20
4.3 Thiết kế chi tiết	21
4.4 Công nghệ hỗ trợ	23
5 Ứng dụng di động	25
5.1 Các chức năng của ứng dụng	25
5.2 Kiến trúc tổng quát	25

MỤC LỤC

6 Hiệu thực hệ thống	28
6.1 Hiệu thực back-end	28
6.1.1 RESTful Web Service - Cách thức giao tiếp giữa client và server	28
6.1.2 Cách thức giao tiếp với database	29
6.1.3 Giới thiệu về kịch bản(scenario)	29
6.1.4 Văn phạm (grammar) dùng tạo ra kịch bản lưu trữ	30
6.1.5 Cấu trúc dữ liệu xây dựng kịch bản hệ thống	32
6.1.6 Sơ đồ mô tả luồng dữ liệu trong hệ thống khi thêm, sửa, xóa kịch bản người dùng	34
6.1.7 Module parser chuyển đổi kịch bản lưu trữ thành kịch bản hệ thống (scenario creator)	35
6.1.8 Module hỗ trợ xây dựng kịch bản tùy ý (script builder)	36
6.1.9 Giới thiệu về tính hợp lệ của kịch bản	37
6.1.10 Module quản lý trạng thái các kịch bản (scenario runner)	41
6.1.11 Module hỗ trợ bảo mật, xác thực và phân quyền (authorization and authentication)	42
6.2 Xây dựng module điều khiển thiết bị	43
6.2.1 Lắp đặt thiết bị	43
6.2.2 Hiện thực module	46
6.2.3 Dánh giá	51
6.3 Phát triển ứng dụng di động	52
6.3.1 Hướng phát triển ứng dụng di động	52
6.3.2 Các công nghệ sử dụng	52
6.3.3 Hiện thực ứng dụng di động	55
7 Thí nghiệm và đánh giá hệ thống	69
7.1 Phương pháp thực nghiệm	69
7.2 Kết quả và đánh giá	70
8 Thảo luận	72
8.1 Giải pháp xử lý các kịch bản có khả năng mâu thuẫn	72
8.1.1 Độ ưu tiên mặc định cho một số loại kịch bản	72
8.1.2 Cách xử lý một số trường hợp trùng độ ưu tiên	73
8.2 Giải pháp tối ưu việc quản lý, vận hành các kịch bản dựa trên kĩ thuật multi-threading	73
8.3 Giải pháp xây dựng hệ thống ứng dụng điện toán đám mây phục vụ nhiều người dùng	74
9 Tổng kết	76
Tài liệu tham khảo	78
Phụ lục	80

Danh mục bảng

3.1	Chi tiết về chức năng các bảng được thiết kế trong database ứng dụng	11
4.1	Một vài thông số kỹ thuật của Raspberry Pi 3 Model B	19
6.1	So sánh điểm mạnh, điểm yếu của Phát triển ứng dụng di động thuần túy và Phát triển ứng dụng di động lai [4]	53

Danh mục hình ảnh

2.1	Sơ đồ usecase của hệ thống	5
2.2	Tổng quan hệ thống	8
3.1	Thiết kế database cho ứng dụng	9
3.2	Tổng quan về kiến trúc Spring Framework	12
3.3	Sơ đồ giao tiếp giữa Client và máy chủ RESTful Web Service	13
3.4	Cấu trúc Hibernate	15
3.5	Kiến trúc hệ thống ở back-end	17
4.1	Máy tính Raspberry Pi 3 Model B	18
4.2	Siêu máy tính được chế tạo từ các con Raspberry Pi [2]	19
4.3	Vị trí đặt các cổng GPIO của Raspberry Pi 3 Model B	20
4.4	Vị trí phân bố các cổng GPIO	21
4.5	Vị trí phân bố các cổng GPIO	22
4.6	Sơ đồ tổng quan của module điều khiển thiết bị	23
4.7	Thiết kế chi tiết module điều khiển thiết bị	24
5.1	Sơ đồ luồng Giao diện của ứng dụng di động	26
6.1	Spring MVC RESTful Web services workflow	28
6.2	Tổ chức của tầng truy xuất dữ liệu (DAO)	29
6.3	Văn phạm kịch bản lưu trữ	31
6.4	Cấu trúc dữ liệu kịch bản hệ thống	32
6.5	Sơ đồ luồng dữ liệu khi thêm/sửa/xóa kịch bản	34
6.6	Văn phạm kịch bản tùy biến	37
6.7	Flowchart thể hiện cách kiểm tra kịch bản hợp lệ	39
6.8	Flowchart thể hiện cách kiểm tra kịch bản mâu thuẫn	40
6.9	Flowchart thể hiện cách quản lý trạng thái các kịch bản	41
6.10	Thông tin các chân của đèn led [1]	43
6.11	Module còi gồm 3 chân: VCC, GND và Signal (nằm giữa) [2]	43
6.12	Raspberry Pi được lắp đặt camera chuyên dụng	44
6.13	Module cảm biến khí gas MQ-2 [4]	44
6.14	Module cảm biến ánh sáng quang trở được gắn vào Raspberry Pi [6]	45
6.15	Cảm biến chuyển động PIR	45
6.16	Cách mắc dây cho cảm biến nhiệt độ DS18B20 [11]	46
6.17	Sơ đồ lớp (class diagram) của module điều khiển thiết bị	47
6.18	Quy tắc đánh số của Pi4J cho Raspberry Pi 3 Model B	48
6.19	Bảng các nền tảng Apache Cordova hỗ trợ	55
6.20	Thanh công cụ (Navbar)	57

DANH MỤC HÌNH ẢNH

6.21	Bảng hộp thoại cập nhật chế độ	57
6.22	Thẻ nhà	58
6.23	Bảng hộp thoại cập nhật Ngôi nhà	58
6.24	Một số Thẻ kiểu thiết bị	59
6.25	Thẻ Thiết bị	60
6.26	Bảng hộp thoại cập nhật Thiết bị	60
6.27	Một số Thẻ Kịch bản khi/thì	61
6.28	Thẻ Kịch bản từ/đến	61
6.29	Thẻ Kịch bản tự tạo	62
6.30	Bảng hộp thoại cập nhật Kịch bản tự tạo	62
6.31	Trang đăng nhập	63
6.32	Trang đăng ký	63
6.33	Trang danh sách các ngôi nhà	64
6.34	Bảng hộp thoại tạo ngôi nhà mới	64
6.35	Trang danh sách các kiểu thiết bị	65
6.36	Bảng hộp thoại tạo chế độ mới	65
6.37	Trang danh sách các thiết bị	66
6.38	Bảng hộp thoại tạo thiết bị mới	66
6.39	Trang danh sách các kịch bản tự tạo	67
6.40	Bảng hộp thoại tạo kịch bản tự tạo mới	67
8.1	Cách thức giao tiếp giữa client, server và Raspberry Pi	74

Chương 1

Giới thiệu

1.1 Tổng quan

Trong khoảng những năm trở lại đây, chắc hẳn khái niệm IoT (Internet of Things) đã không còn xa lạ và mới mẻ với chúng ta. Ý tưởng về một thế giới mà mọi thứ trong cuộc sống được kết nối với Internet để truyền tải, trao đổi dữ liệu, từ đó người dùng có thể tương tác, điều khiển và kiểm soát mọi hoạt động trong cuộc sống thông qua những thiết bị thông minh như điện thoại hoặc máy tính bảng chỉ qua vài cái chạm tay đơn giản.

Hãy tưởng tượng một ngôi nhà trong đó có thể kết nối và tự động hóa một số thiết bị điện tử, điện gia dụng, như đèn chiếu sáng, còi báo động,... hay đến thiết bị an ninh như camera chả hạn. Ta sẽ dễ dàng kiểm soát hoạt động của ngôi nhà ngay cả khi vắng mặt từ bất cứ đâu với máy tính hay smartphone, giảm thiểu những rủi ro ngoài ý muốn. Những kịch bản tự động hóa hoạt động của các thiết bị theo ngữ cảnh đem lại tiện nghi, thoải mái và an toàn cho cuộc sống [5].

Theo Gartner, IoT dự kiến sẽ có khoảng 26 tỷ thiết bị vào năm 2020 [3]. Tuy nhiên ở thời điểm hiện tại, IoT vẫn còn là khái niệm khá mới mẻ đối với các quốc gia đang trên đà phát triển như Việt Nam ta. Để ngôi nhà của mình sở hữu những tính năng thông minh kể trên, chắc hẳn chi phí cũng không hề nhỏ. Và đúng như thế, chi phí là một trong những rào cản lớn với sự phát triển IoT ở các nước đang phát triển, trong đó có cả Việt Nam.

1.2 Các thành tựu, công trình liên quan nhà thông minh

Nếu xếp hạng tốc độ đổi mới của các ngành phục vụ cho nhu cầu cuộc sống thì chắc chắn công nghệ luôn là một trong những vị trí dẫn đầu. Ngày nay rất nhiều ngành nghề đã ứng dụng công nghệ cao vào sản phẩm của mình, trong đó có mô hình nhà thông minh. Trong những năm gần đây nhà thông minh đang dần trở nên phổ biến hơn với người dân Việt Nam [4]. Nhiều sản phẩm nhà thông minh được đánh giá cao ở nước ngoài có thể kể tới như Moni, Crestron, Control4,... và ở Việt Nam như Bkav, Vinteli Home,... [10]. Những ngôi nhà thông minh này được tích hợp công nghệ và nhiều tiện ích đi kèm. Sau đây là một số hướng phát triển nhà thông minh chính ở thời điểm hiện tại:

- Xu hướng nhà thông minh với các cơ chế bảo mật.

CHƯƠNG 1. GIỚI THIỆU

- Xu hướng nhà thông minh tiết kiệm năng lượng và kiểm soát môi trường.
- Hệ thống ánh sáng thông minh.
- Xu hướng nhà thông minh sử dụng công nghệ không dây.
- Xu hướng tự động hóa, toàn quyền kiểm soát nhà thông minh [9].

Trong luận văn này, hệ thống nhóm sắp xây dựng cũng đi theo xu hướng tự động hóa nhà thông minh. Tuy nhiên, một điểm khác biệt đó là hệ thống cho phép người dùng có thể “lập trình” cho nhà thông minh hoạt động tự động dựa theo các kịch bản. Đây không hoàn toàn là xu hướng mới với các nước phát triển, nhưng ở một số nước đang phát triển như Việt Nam, giải pháp này có thể giúp cho một số hộ gia đình được tiếp cận gần hơn xu hướng nhà thông minh với mức chi phí phù hợp, có khả năng đáp ứng một số nhu cầu thực tế đơn giản.

1.3 Nhà thông minh với Raspberry Pi

Có rất nhiều tùy chọn trong việc tiếp cận với IoT nhưng không có gì tốt hơn ngoài những trải nghiệm thực tế. Một trong những nền tảng quan trọng cho việc học IoT là mạch tính toán nhỏ và đơn giản, nhắm đến khả năng tạo một chiếc máy tính nhỏ gọn, giá rẻ phục vụ cho công tác học tập, nghiên cứu và thử nghiệm. Raspberry Pi không chỉ là một nền tảng phần cứng thú vị để ứng dụng cho các dự án IoT mà còn là công cụ giúp các nhà phát triển học hỏi và hoàn thiện kỹ năng Internet kết nối vạn vật. Một trong những ứng dụng tốt nhất của Raspberry Pi chính là trở thành trung tâm điều khiển dành cho các thiết bị sử dụng nguồn điện áp thấp.

1.4 Mục tiêu và phạm vi đề tài

Từ những nhu cầu thực tế phát sinh, nhóm mong muốn được thực hiện một hệ thống nhà thông minh được quản lý tự động bởi các kịch bản định sẵn, tương tác với ứng dụng nền tảng web hay ứng dụng thiết bị di động. Trong đó, Raspberry Pi sẽ đóng vai trò là “bộ não” trung tâm, quản lý thiết bị trong nhà, đồng thời cũng là nơi vận hành, xử lý các kịch bản người dùng.

Mục tiêu của đề tài trong việc xây dựng hệ thống này đó là:

- Có giao diện ứng dụng nền tảng web hoặc di động để người dùng thao tác, quản lý ngôi nhà của riêng mình.
- Xây dựng hệ thống (phía back-end) giúp cho việc lưu trữ, xử lý, vận hành các kịch bản người dùng định nghĩa trên ứng dụng.
- Các thao tác, điều khiển các thiết bị gắn trên Raspberry Pi hoạt động theo ý muốn của kịch bản người dùng đã định.

CHƯƠNG 1. GIỚI THIỆU

Vì giới hạn thời gian, hệ thống chưa thực sự hoàn thiện và tối ưu. Tuy nhiên, hệ thống vẫn có thể hoạt động tốt trên một số kịch bản đáp ứng được với thực tế và có độ phức tạp thấp. Và ở đề tài này, nhóm chỉ tập trung vào phát triển, xây dựng ứng dụng nền tảng Web, cũng như ứng dụng trên thiết bị di động, các thiết bị phần cứng chỉ là hỗ trợ, phục vụ cho thí nghiệm và đánh giá hệ thống.

1.5 Cấu trúc luận văn

Toàn bộ nội dung luận văn được nhóm trình bày trong bảy chương. Các chương này lần lượt nêu lên những kiến thức cần thiết, chi tiết cách hiện thực để xây dựng và hoàn thiện hệ thống. Ở chương cuối, nhóm đưa ra kết quả tổng kết về những vấn đề đã được giải quyết ở đề tài này, song song đó là những hạn chế còn tồn đọng cũng như là hướng phát triển trong tương lai.

Đầu tiên ta đến với chương **Tổng quan hệ thống**, người đọc sẽ có cái nhìn toàn cảnh về lý do nhóm tiến hành thực hiện đề tài, vai trò và vị trí của đề tài trong xu thế phát triển nhà thông minh ở Việt Nam cũng như phạm vi của luận án này. Kế đến chương **Phân tích và thiết kế hệ thống** sẽ nêu lên những yêu cầu được đặt ra cho hệ thống, từ đó nhóm đề xuất những phương pháp thiết kế phù hợp. Sau khi có được cái nhìn tổng thể về hệ thống, chương **Tổng quan về hệ thống back-end** giúp người đọc hiểu thêm về cách xây dựng, tổ chức, thiết kế ở phía back-end. Đồng thời các công nghệ sử dụng để xây dựng back-end cũng sẽ được giới thiệu. Đồng thời để hiểu thêm về cách mà back-end tương tác với Raspberry Pi, cũng như việc điều khiển các thiết bị gắn trên nó, ta sẽ đến với chương **Module điều khiển thiết bị**. Qua đó, một dịch vụ (service) được xây dựng để back-end có thể tương tác được với các thiết bị. Tiếp đến chương **Tổng quan giao diện ứng dụng** sẽ giới thiệu cho ta về mặt giao diện ứng dụng của hệ thống đã thiết kế, cũng như cách tổ chức và cách sử dụng cho việc quản lý nhà, chế độ, thiết bị và kịch bản người dùng. Các công nghệ sử dụng để xây dựng giao diện cũng sẽ được giới thiệu tại đây. Sau khi trình bày tổng quan hệ thống, chúng ta đi đến với chi tiết hiện thực back-end, giao diện ứng dụng, module điều khiển thiết bị. Những thông tin này sẽ được trình bày rõ trong chương kế tiếp **Hiện thực và đánh giá**. Cuối cùng, chương **Tổng kết** sẽ tóm tắt lại các kết quả đã đạt được sau quá trình thực hiện luận án, cũng như hạn chế còn chưa giải quyết, từ đó đề xuất hướng phát triển mở rộng trong tương lai.

Chương 2

Phân tích và thiết kế hệ thống

2.1 Các yêu cầu chức năng

Từ các mục tiêu đề tài đã được đặt ra ở phần 1.4 về việc xây dựng một hệ thống nhà thông minh cho phép người dùng quản lý các thiết bị bằng kịch bản mong muốn bên trong một hoặc nhiều ngôi nhà của mình, các yêu cầu chức năng chính được xác định cho hệ thống chúng tôi phát triển sẽ bao gồm các máy tính kích thước nhỏ Raspberry Pi ứng với từng ngôi nhà được lắp đặt hệ thống nơi người dùng sẽ kết nối các thiết bị trong nhà thông qua các chân cắm GPIO. Một ứng dụng di động nơi người dùng sẽ đăng ký cũng như chỉnh sửa thông tin các ngôi nhà, khai báo và quản lý thông tin các thiết bị bên trong ngôi nhà. Ứng dụng cần cung cấp cho người dùng một cơ chế kịch bản hoạt động thiết bị vừa đơn giản để người dùng có thể thêm mới, thiết lập dễ dàng đồng thời vừa có khả năng tùy biến cao khi người dùng muốn kết hợp hoạt động của nhiều thiết bị trong những ngữ cảnh phức tạp. Ngoài ra, ứng dụng còn cần cho phép người dùng tạo nên các chế độ hoạt động khác nhau của ngôi nhà ứng với từng trường hợp thực tế như vắng nhà, tiệc tùng, trời mưa... Trong đó, từng chế độ là từng nhóm các kịch bản riêng biệt để người dùng có thể dễ dàng chuyển đổi qua lại một cách nhanh chóng và thuận tiện khi cần thiết.

2.2 Các yêu cầu phi chức năng

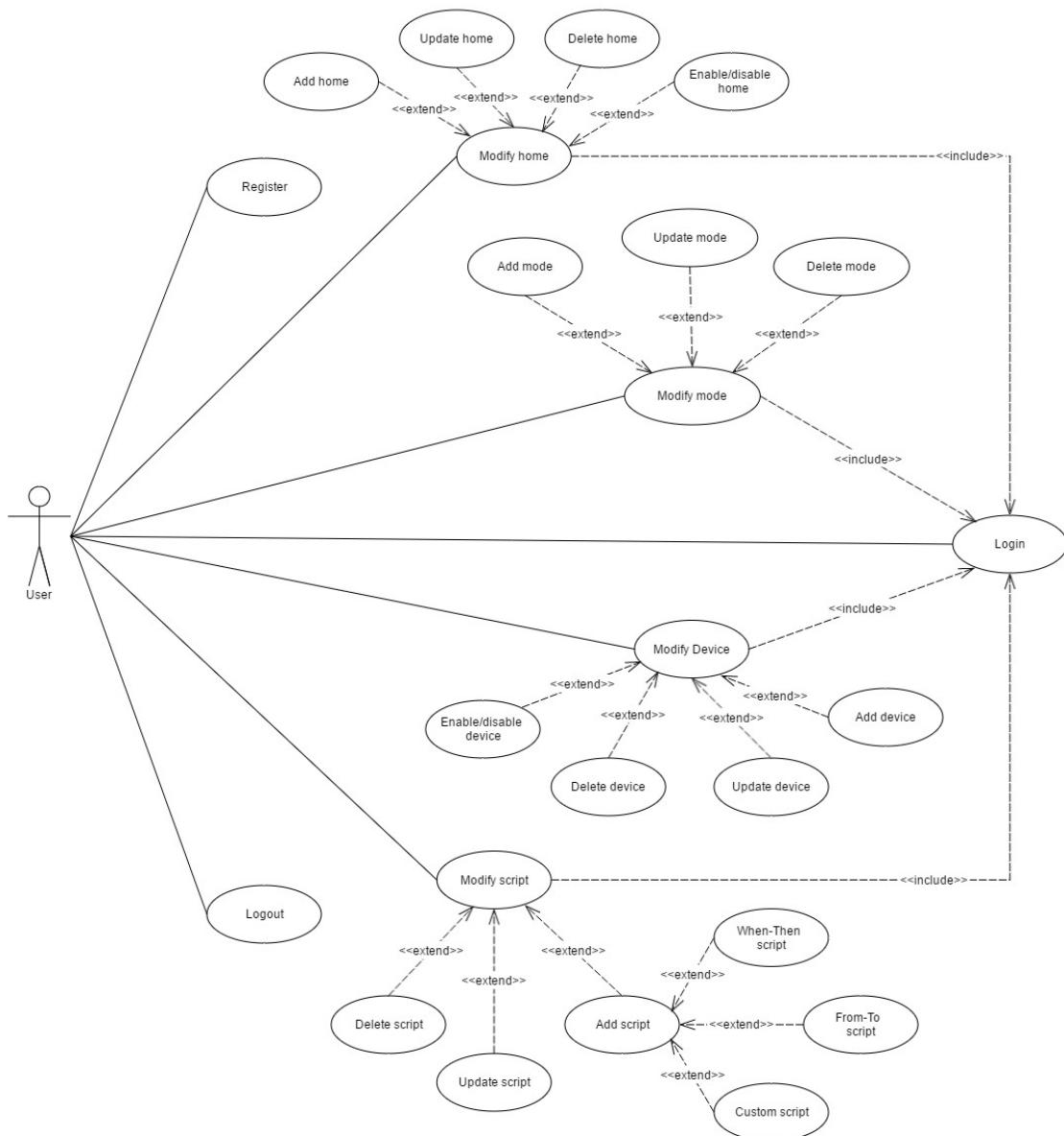
Ngoài những yêu cầu chức năng đã được nêu, hệ thống cũng cần có những yêu cầu phi chức năng như ứng dụng cần có giao diện thu hút, đơn giản, dễ sử dụng, không quá phức tạp và quá nhiều chức năng. Ngoài ra, bảo mật (security) cũng là một yếu tố không thể trong các ứng dụng nhà thông minh. Giao tiếp giữa client và server phải thông qua HTTPS (HTTP Secure) để tránh bị đánh cắp thông tin trên đường truyền mạng, mật khẩu người dùng phải được mã hóa trước khi lưu vào database để tài khoản khách hàng không bị đánh cắp.

2.3 Sơ đồ usecase

Từ những yêu cầu chức năng và phi chức năng đã được phân tích, chúng tôi thiết kế sơ đồ usecase như hình 2.1. Sơ đồ gồm có 7 usecase chính, chi tiết cụ thể từng usecase được mô tả như sau:

Đăng ký (Register): Chức năng đăng ký cho phép người dùng đăng ký một tài khoản để có thể sử dụng hệ thống. Người dùng cần nhập những thông tin như họ tên, tên đăng

CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG



Hình 2.1: Sơ đồ usecase của hệ thống

nhập, địa chỉ, email và mật khẩu. Sau khi đã đăng ký xong, người dùng sẽ sử dụng tài khoản này để đăng nhập và tiến hành tạo nhà, thêm các thiết bị, tạo kịch bản,...

Đăng nhập (Login): Đăng nhập là bước bắt buộc để có thể sử dụng các dịch vụ của hệ thống. Người dùng cần nhập đúng tên đăng nhập và mật khẩu đã đăng ký, nếu thành công, hệ thống sẽ hiển thị danh sách các ngôi nhà có thể truy cập, nếu đăng nhập thất bại, hệ thống cần hiển thị thông báo lỗi để người sử dụng thử lại.

CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

Chỉnh sửa nhà (Modify home): Chỉnh sửa nhà bao gồm 3 thao tác con là thêm (add), cập nhật (update), xóa (delete) và kích hoạt/vô hiệu hóa (enable/disable). Khi một người sử dụng hệ thống lần đầu vừa hoàn tất việc đăng ký và đăng nhập thành công vào hệ thống, điều đầu tiên họ cần làm là tiến hành tạo nhà. Khi tạo nhà, người dùng cần nhập các thông tin cần thiết như tên nhà, địa chỉ, ... Do một người có thể quản lý nhiều căn nhà nên người sử dụng có thể tạo nhiều nhà khác nhau. Sau khi tạo, nếu cảm thấy có sai sót, người dùng có thể cập nhật lại thông tin của ngôi nhà đó, hoặc nếu không còn quản lý một ngôi nhà nữa, người dùng có thể tiến hành xóa ngôi nhà đó ra khỏi danh sách những ngôi nhà đang được quản lý của mình. Nếu muốn hệ thống trong nhà ngừng hoạt động trong một khoảng thời gian nhất định, người dùng có thể sử dụng chức năng vô hiệu hóa nhà (disable home), khi đó, tất cả các kịch bản (script) trong nhà sẽ dừng tạm thời. Nếu muốn trở lại bình thường, người dùng chỉ cần kích hoạt lại ngôi nhà (enable home).

Chỉnh sửa chế độ (Modify mode): Một ngôi nhà có nhiều chế độ (mode) khác nhau như chế độ vắng nhà, chế độ có nhà, chế độ đi chơi,... Tại mỗi chế độ sẽ có những kịch bản khác nhau hoạt động, như ở chế độ có nhà, đèn cần được bật từ 18h tối đến 6h sáng, ngược lại, ở chế độ vắng nhà thì không cần bật đèn vào buổi tối mà cần kích hoạt cảm biến chuyển động được đặt trước cửa để chống trộm. Vì vậy, hệ thống cần cung cấp chức năng thêm chế độ (add mode) để người dùng có thể tạo ra những chế độ của riêng mình, sau khi thêm, người dùng sẽ có nhu cầu chỉnh sửa hoặc xóa đi những chế độ không cần thiết, vì vậy, hệ thống cũng cần cung cấp thêm tính năng cập nhật (update mode) và xóa (delete mode).

Chỉnh sửa thiết bị (Modify device): Sau khi đã tạo nhà, tạo chế độ cho một căn nhà, điều tiếp theo mà người sử dụng cần là thêm các thiết bị vào ngôi nhà của mình (add device). Chỉ sau khi thêm thiết bị thành công, người dùng mới có thể thêm kịch bản cho từng thiết bị của mình. Để thêm một thiết bị, người dùng cần nhập đầy đủ thông tin về thiết bị đó như tên thiết bị, loại thiết bị, mô tả, ... Bên cạnh đó, hệ thống cần cung cấp chức năng cập nhật thông tin thiết bị (update device) và xóa (delete device) nếu người dùng cảm thấy thiết bị đó không còn được dùng nữa. Hệ thống cũng cần cung cấp chức năng vô hiệu hóa thiết bị (disable device) trong những trường hợp người dùng muốn dừng các kịch bản đang hoạt động trên thiết bị này một cách tạm thời, sau đó, người dùng có thể kích hoạt lên lại bất kỳ khi nào họ muốn (enable device).

Chỉnh sửa kịch bản (Modify script): Để thiết bị hoạt động theo một kịch bản được xác định trước, hệ thống cung cấp cho người dùng chức năng thêm kịch bản (add script). Có ba loại kịch bản chính: kịch bản When-Then (When-Then script), kịch bản From-To (From-To script) và kịch bản tùy chỉnh (Custom script). Người dùng có thể truy cập vào từng thiết bị và thêm những kịch bản mà mình mong muốn. Sau khi thêm, người dùng có thể chỉnh sửa (update script) hoặc xóa (delete script) những kịch bản không sử dụng.

Đăng xuất (Logout): Sau khi hoàn tất phiên làm việc, người dùng có thể đăng xuất khỏi hệ thống. Điều này đảm bảo tài khoản người dùng được bảo mật, không bị sử dụng

CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

bởi những người dùng chung thiết bị. Sau khi đã đăng xuất, người dùng phải nhập lại tên đăng nhập và mật khẩu để truy cập lại vào ngôi nhà của mình.

Với sơ đồ usecase đã trình bày, hệ thống đã đáp ứng được những yêu cầu cơ bản của một ứng dụng nhà thông minh như đăng nhập, tạo nhà, tạo chế độ, thiết bị, ... Ngoài ra, hệ thống còn cung cấp chức năng thêm mới một kịch bản, đây là phần cốt lõi của ứng dụng, cho phép người dùng có thể tùy chỉnh cách hoạt động của các thiết bị trong nhà, tạo ra những kịch bản của riêng họ giống như lập trình cho một ngôi nhà.

2.4 Tổng quan hệ thống



Hình 2.2: Tổng quan hệ thống

Để đáp ứng mục tiêu và yêu cầu đặt ra của đề tài về việc xây dựng hệ thống điều khiển các thiết bị thông dụng trong nhà thông qua các kịch bản, nhóm chúng tôi quyết định thiết kế hệ thống với 3 bộ phận chính: Module điều khiển thiết bị (Module Controller) đảm nhiệm việc kết nối và điều khiển các thiết bị được gắn với máy tính siêu nhỏ Raspberry Pi, máy chủ hệ thống (Server) với vai trò trung tâm xử lý các nghiệp vụ liên quan đến quản lý nhà, thiết bị, kịch bản trong hệ thống và ứng dụng di động (Mobile Application) cung cấp giao diện để người dùng truy cập và sử dụng hệ thống. Mỗi liên hệ của 3 bộ phận này

CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

được thể hiện ở hình 2.2.

Module điều khiển thiết bị (Module Controller): Bộ phận này là một bộ điều khiển liên kết trực tiếp với máy tính siêu nhỏ Raspberry Pi nhằm gửi tín hiệu điều khiển đến các thiết bị phần cứng được kết nối cũng như nhận tín hiệu thông tin trả về từ các thiết bị này. Tùy vào chủng loại thiết bị phần cứng mà bộ điều khiển sẽ có cơ chế xử lý thông tin riêng. Module điều khiển thiết bị cung cấp API cho phía máy chủ hệ thống để máy chủ có thể gián tiếp truy cập, gửi nhận thông tin và điều khiển các thiết bị phần cứng một cách dễ dàng.

Máy chủ hệ thống (Server): Bộ phận trung tâm của hệ thống, thông qua cơ sở dữ liệu xử lý các thao tác nghiệp vụ cốt lõi thêm mới, xóa, sửa các thông tin nhà, thiết bị, chế độ và vận hành các kịch bản của người dùng tạo ra. Từ đó, bộ phận này cung cấp một danh sách API để giao tiếp, truyền nhận thông tin với phía ứng dụng di động.

Ứng dụng di động (Mobile Application): Người dùng truy cập và sử dụng hệ thống thông qua bộ phận này. Các giao diện về đăng ký, đăng nhập vào hệ thống, xem thông tin và quản lý danh sách nhà, thiết bị, chế độ và kịch bản được bộ phận này cung cấp.

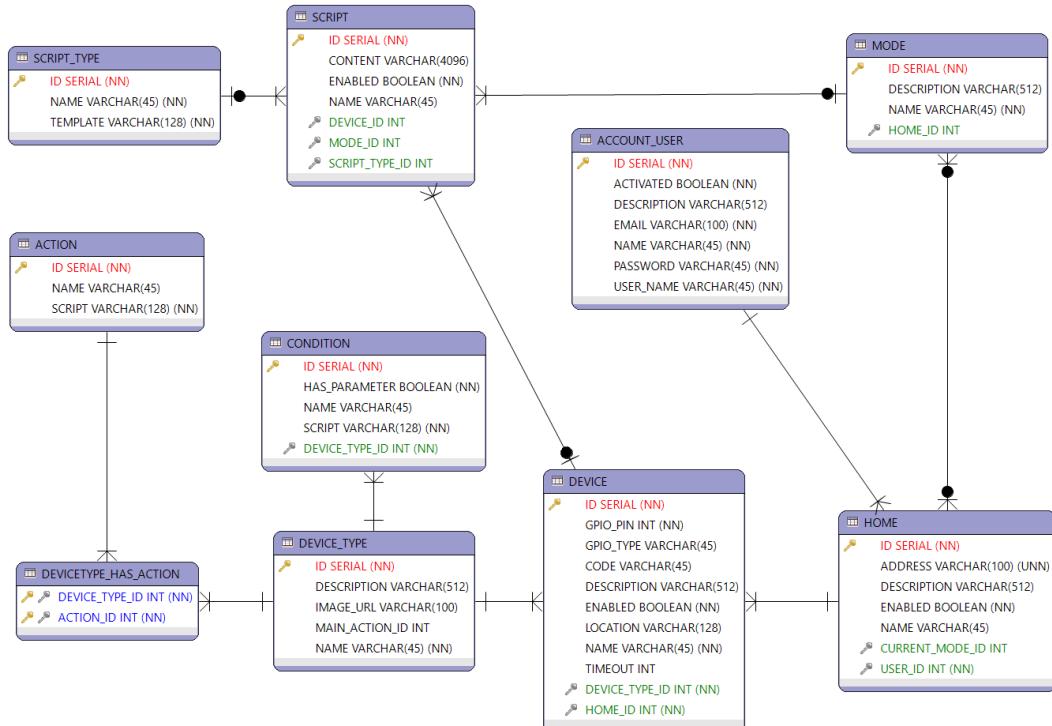
Với 3 bộ phận như trên, hệ thống được chia tách thành các thành phần độc lập nhưng có mối liên hệ chặt chẽ với nhau nhằm xử lý từng yêu cầu riêng biệt cũng như để đạt được mục tiêu chung của đề tài. Chi tiết về thiết kế và hiện thực của từng bộ phận được trình bày tại các chương 4, 5 và 6.

Chương 3

Back-end

Ở mục này, nhóm sẽ tập trung vào phần thiết kế database cho ứng dụng. Tiếp đến, nhóm sẽ đi vào giới thiệu tổng quan về các công nghệ mà nhóm đã tìm kiếm và ứng dụng trong việc xây dựng hệ thống này. Cuối cùng, dựa trên những phân tích về yêu cầu hệ thống, nhóm đề xuất và thiết kế ra mô hình kiến trúc của back-end để đáp ứng các nhu cầu đó.

3.1 Thiết kế cơ sở dữ liệu



Hình 3.1: Thiết kế database cho ứng dụng

Mục tiêu của ứng dụng đó là tạo ra và lưu trữ các kịch bản của người dùng. Do đó, việc thiết kế database đóng một vai trò quan trọng trong việc xây dựng ứng dụng này. Ở phần thiết kế database, nhóm sử dụng PostgreSQL để hiện thực trên Raspberry Pi.

CHƯƠNG 3. BACK-END

Mô hình quan hệ giữa các bảng trong phần thiết kế database được trình bày như ở hình 3.1.

Trở lại với yêu cầu được đặt ra từ hệ thống, người dùng có thể có nhiều ngôi nhà. Trong mỗi ngôi nhà sẽ điều khiển nhiều thiết bị khác nhau thuộc nhiều loại khác nhau. Mỗi thiết bị có thể có nhiều kịch bản để tự động hóa chúng. Kịch bản được đặc tả theo dạng **<điều kiện – hành động>**, nghĩa là dưới điều kiện nào đó do người dùng quyết định thì sẽ có những hành động tương ứng xảy ra. Những kịch bản sẽ hoặc thuộc dạng đơn giản (bao gồm 1 điều kiện, 1 hành động), hoặc dạng phức tạp mà người dùng có thể tự tạo theo ý muốn riêng của mình. Hơn nữa, với mỗi ngôi nhà có thể có nhiều chế độ quản lý nhưng tại 1 thời điểm chỉ có 1 chế độ được kích hoạt, ví dụ như chế độ đi vắng thì sẽ có những kịch bản riêng, còn với chế độ ở nhà sẽ là bộ kịch bản khác nhằm giúp người dùng tiện lợi trong việc quản lý căn nhà của mình ở nhiều hoàn cảnh khác nhau.

Với cách thiết kế như trên , hệ thống có những đặc điểm

- Tính thích ứng cao khi có thay đổi yêu cầu.
- Đơn giản trong việc bảo trì và cập nhật (ví dụ như thêm thiết bị hay loại thiết bị mới).

Chi tiết về chức năng các bảng trong thiết kế được mô tả ở bảng 3.1. Các bảng quan trọng trong hệ thống có thể kể đến là Account_User, Home, Mode, Device, Script dùng để chứa dữ liệu của người dùng. Còn những bảng còn lại chủ yếu chứa dữ liệu của hệ thống (master data).

3.2 Thiết kế và kiến trúc hệ thống back-end

Hệ thống back-end được hiện thực hoàn toàn dựa trên ngôn ngữ Java và tận dụng sức mạnh từ Spring, một trong những framework được sử dụng nhiều nhất trong Java EE framework.

Spring có thể giúp chúng ta xây dựng ứng dụng một cách nhanh chóng, tốn ít thời gian để làm quen. Spring cho phép các thư viện bên ngoài (third party service) tích hợp vào ứng dụng dễ dàng. Ví dụ như ứng dụng ta có thể kết nối đến nhiều loại database khác nhau với chi phí thay đổi rất thấp hay 1 ứng dụng Spring MVC có thể chuyển thành dịch vụ cung cấp tài nguyên thông qua REST API. Một điểm cộng nữa cho Spring đó là các công nghệ front-end đều có thể làm việc cùng với nó. Với sự phổ biến của Spring trong cộng đồng phát triển ứng dụng Java nền web cùng nhiều đặc điểm nổi trội so với các frameworks khác như: Struts, Vaadin,... nhóm đã chọn Spring để phát triển hệ thống back-end này. Phần 3.2.1 sẽ cho ta cái nhìn tổng quan về Spring framework, cùng những tính năng, ưu điểm của nó.

CHƯƠNG 3. BACK-END

Bảng 3.1: Chi tiết về chức năng các bảng được thiết kế trong database ứng dụng

STT	Tên bảng	Chức năng
1	Account_User	Thông tin liên quan đến tài khoản người dùng, đã kích hoạt hay chưa, một vài thông tin cá nhân cơ bản
2	Home	Thông tin liên quan đến nhà , có đang “active” hay không, nhà đang ở chế độ nào
3	Mode	Thông tin liên quan chế độ được người dùng định nghĩa cho ngôi nhà của mình
4	Device	Thông tin các thiết bị cho từng nhà mà hệ thống quản lý như cổng GPIO, có đang ”active” không,...
5	Device_Type	Loại thiết bị hệ thống có thể quản lý như đèn , còi , cảm biến,...
6	Script	Quản lý các thông tin liên quan đến kịch bản, kịch bản người dùng được lưu xuống theo 1 cú pháp định sẵn
7	Script_Type	Phân loại kịch bản đơn giản, được cung cấp sẵn hay phức tạp
8	Condition	Những điều kiện hợp lệ gắn với từng loại thiết bị để sử dụng khi định nghĩa kịch bản
9	Action	Những hành động hợp lệ gắn với từng loại thiết bị để sử dụng khi định nghĩa kịch bản
10	Device_Has_Action	Mô tả mối quan hệ giữa loại thiết bị và hành động của chúng

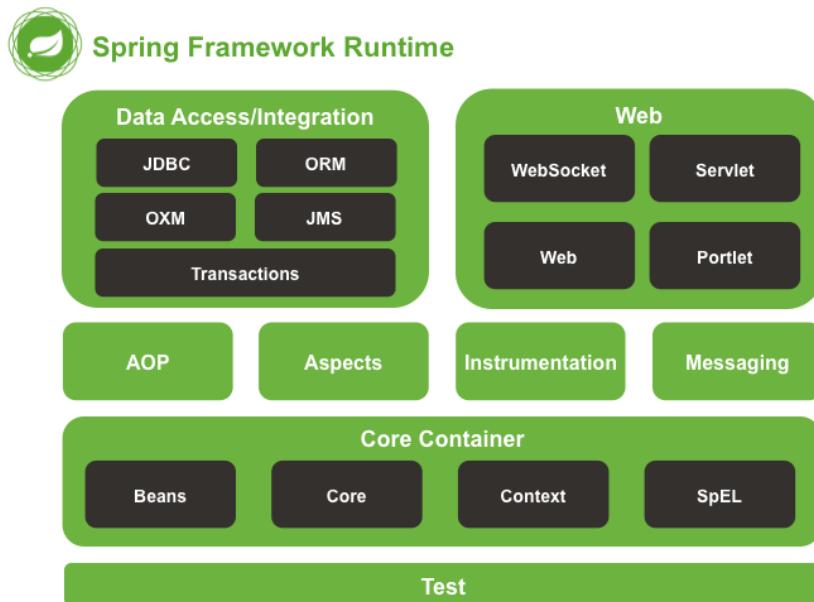
3.2.1 Spring framework

Hai khái niệm chính của Spring framework core là “Dependency Injection - DI” và “Aspect Oriented Programming - AOP”. Spring framework được sử dụng như là ứng dụng java cơ bản để đạt được kỹ thuật “loose coupling” giữa các components khác nhau bằng cách sử dụng kỹ thuật DI và hỗ trợ việc thực hiện chéo những task vụ như logging, authentication,... theo kỹ thuật AOP [8][12].

Spring framework cung cấp khá nhiều tính năng khác và số lượng lớn các module cho các mục đích cụ thể, ví dụ như web có Spring MVC, hỗ trợ security có Spring Security, tương tác với datababse có Spring JDBC, và nhiều thứ khác nữa. Ngoài ra, nó còn là một dự án open source với rất nhiều cộng đồng sử dụng, tài liệu tham khảo.

Hình 3.2 cho thấy kiến trúc tổng quan của Spring framework với nhiều module thiết kế tách biệt phục vụ cho nhiều mục đích khác nhau. Một số tính năng cũng như ưu điểm có thể kể đến của Spring:

- Dependency Injection hoặc Inversion of Control được sử dụng để giúp các component tách rời, độc lập với nhau. Spring container sẽ giúp gắn kết những components này lại với nhau theo đặc tả business của bạn.



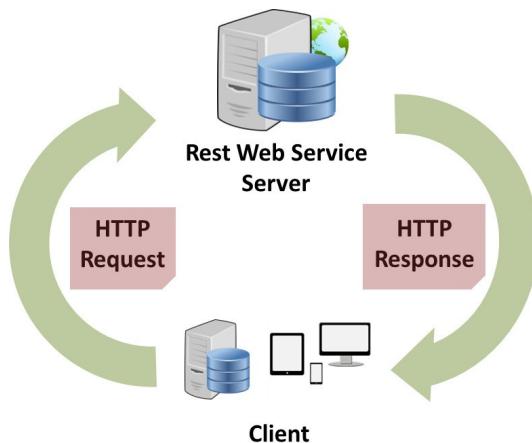
Hình 3.2: Tổng quan về kiến trúc Spring Framework

- Spring MVC framework được sử dụng cho phát triển ứng dụng web rất dễ dàng với việc hỗ trợ rất tốt các tính năng web services, json,... (như RESTful web service framework).
- Hỗ trợ quản lý transaction, JDBC operations, File uploading, Exception Handling,... rất dễ dàng bằng cách cấu hình được rút gọn, thay vào đó là sử dụng annotation.
- Làm giảm đi khối lượng code rất nhiều, chẳng hạn như việc khởi tạo đối tượng, open/close các resources,...

Với đặc trưng của ứng dụng hiện tại là dùng những kịch bản quản lý tự động thiết bị trong nhà, không cần real time (như ứng dụng chat, stream video,...) và chỉ gửi yêu cầu tại một số thời điểm, với tần suất nhỏ nên nhóm lựa chọn dùng RESTful web service, xây dựng những API cho client có thể giao tiếp và truy xuất tài nguyên từ server một cách thuận tiện. Hơn thế nữa, tận dụng sức mạnh của Spring framework, việc xây dựng các RESTful web service trở nên dễ dàng và tiết kiệm công sức hơn rất nhiều.

3.2.2 RESTful Web Service

Representational State Transfer (REST) là một kiểu kiến trúc được sử dụng trong việc giao tiếp giữa các máy tính (máy tính cá nhân và máy chủ của trang web) trong việc quản lý các tài nguyên trên internet. Hình 3.3 là sơ đồ thể hiện cách giao tiếp giữa client và máy chủ RESTful web service thông qua giao thức HTTP [11].



Hình 3.3: Sơ đồ giao tiếp giữa Client và máy chủ RESTful Web Service

Những nguyên tắc cơ bản của một RESTful Web Service:

- Sử dụng các phương thức HTTP một cách rõ ràng.
- Phi trạng thái.
- Hiển thị cấu trúc thư mục như URIs.

Sử dụng các phương thức HTTP một cách rõ ràng

Một đặc tính quan trọng của dịch vụ RESTful là sử dụng một cách rõ ràng các phương thức HTTP theo cách một giao thức được xác định bởi RFC 2616.

REST yêu cầu các nhà phát triển sử dụng phương thức HTTP một cách rõ ràng theo cách tương thích với giao thức chuẩn. Nguyên lý thiết kế REST cơ bản này thiết lập một ánh xạ 1-1 giữa các hành động tạo, đọc, cập nhật và xoá (CRUD) các quá trình vận hành và các phương thức HTTP. Theo cách ánh xạ này thì:

- Để tạo một tài nguyên trên máy chủ, ta cần sử dụng phương thức POST.
- Để truy xuất một tài nguyên, ta sử dụng GET.
- Để thay đổi trạng thái một tài nguyên hoặc để cập nhật nó, ta sử dụng PUT.
- Để huỷ bỏ hoặc xoá một tài nguyên, ta sử dụng DELETE [11].

Phi trạng thái

Một yêu cầu hoàn chỉnh, độc lập không đòi hỏi máy chủ thu thập được bất kỳ ngữ cảnh hoặc trạng thái của ứng dụng nào trong lúc xử lý yêu cầu. Một ứng dụng (hoặc máy khách) Web service REST chứa ở phần đầu và phần thân trang HTTP của một yêu cầu tất cả các tham số, ngữ cảnh và dữ liệu cần thiết bởi thành phần bên ngoài máy chủ để đưa ra một phản hồi. Phi trạng thái theo nghĩa này nâng cao tính hiệu quả của dịch vụ Web, đơn

CHƯƠNG 3. BACK-END

giản hoá thiết kế và sự thi hành của các thành phần của máy chủ vì khi máy chủ không có trạng thái sẽ huỷ bỏ nhu cầu để đồng bộ hoá các mảng dữ liệu với một ứng dụng bên ngoài [11].

Hiển thị cấu trúc thư mục như URIs

Các địa chỉ Web service REST nên có tính hiện thực theo nghĩa rằng chúng dễ dàng đổi với người dùng. Có thể nghĩ rằng một địa chỉ đường dẫn như là giao diện tự đóng gói mà đòi hỏi ít lý giải hay tham chiếu, nếu có, đối với một nhà phát triển để hiểu nó nhằm đến điểm gì và phân phối tài nguyên liên quan. Cuối cùng, cấu trúc của địa chỉ nên rõ ràng, có thể đoán được và dễ hiểu. Một cách để đạt được mức độ sử dụng này là xác định cấu trúc thư mục giống URIs. Loại URI này có thứ bậc, có điểm khởi nguồn tại một đường dẫn đơn giản, và có nhánh đi ra là các nhánh phụ thể hiện các vùng chính của dịch vụ. Theo định nghĩa này, một URI không chỉ là một chuỗi bị cắt không giới hạn, mà còn là một cây với các nhánh chính và nhánh dọc nối nhau tại các nút. Ví dụ: <http://www.myservice.org/discussion/topics/{topic}>.

Các địa chỉ URIs nên giữ nguyên để khi tài nguyên thay đổi hoặc khi tiến hành thay đổi dịch vụ, đường liên kết cũng sẽ giữ nguyên. Việc này cho phép đánh dấu lại vị trí đang đọc. Nó cũng rất quan trọng vì mối liên quan giữa các tài nguyên mà được mã hoá trong các địa chỉ được giữ nguyên độc lập với các mối liên quan đại diện khi chúng được lưu trữ [11].

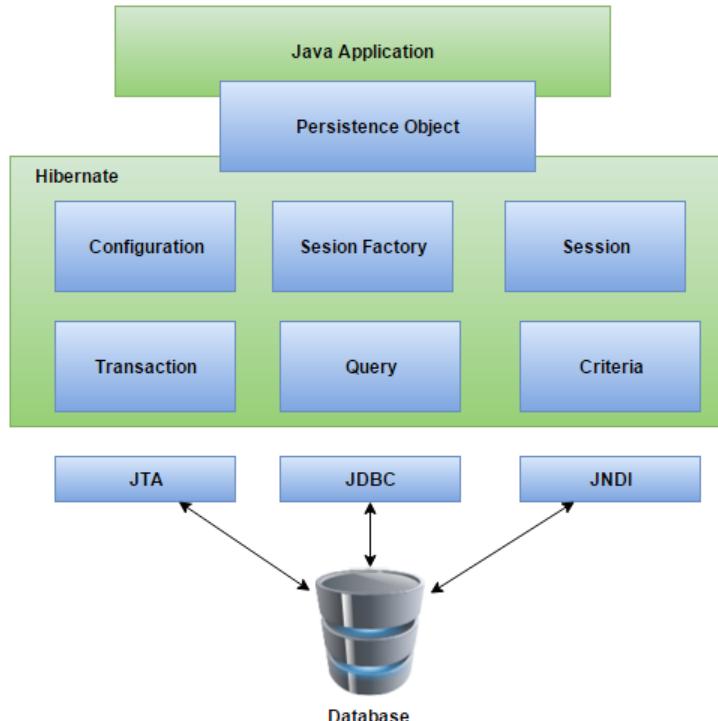
Với thiết kế database của nhóm, giữa các thực thể có mối quan hệ, liên kết với nhau chặt chẽ. Việc quản lý và thao tác với dữ liệu trực tiếp trên các database thường là công việc nặng nhọc và dễ phát sinh lỗi với các lập trình viên. Hibernate framework ra đời, giúp ta giải quyết những khó khăn trên. Ta có thể mô tả các đối tượng Java (Java objects) đại diện tương ứng cho mối quan hệ giữa các thực thể ở database và Hibernate sẽ làm công việc chuyển ánh xạ đó. Hibernate còn giúp quản lý cả kết nối (connection) với database, quản lý session,... Với những ưu điểm đó, nhóm chọn Hibernate để đơn giản hóa việc quản lý, truy xuất dữ liệu từ database. Hơn nữa, Hibernate cũng nhận được sự hỗ trợ từ Spring không nhỏ, tiết kiệm chi phí thời gian lẫn công sức trong việc xây dựng hệ thống. Phần 3.2.3 sẽ giới thiệu tổng quan về kiến trúc cũng như tính năng, điểm mạnh của Hibernate.

3.2.3 Hibernate Framework

Hibernate Framework là một công cụ mã nguồn mở, dung lượng nhỏ (lightweight) và ORM (Object Relational Mapping) giúp đơn giản hóa việc phát triển ứng dụng Java để tương tác với cơ sở dữ liệu. Do Hibernate Framework là một ORM framework cho persistence layer nên khi phát triển ứng dụng, lập trình viên chỉ cần tập trung vào những layer khác (như tầng ứng dụng-business) mà không cần xem xét nhiều về persistence layer, dẫn đến tránh thao tác nhiều với database.

CHƯƠNG 3. BACK-END

Cấu trúc Hibernate được thể hiện qua hình 3.4. Hibernate sử dụng nhiều API của Java như JDBC, Java Transaction, Java Naming and Directory Interface. JDBC cho phép bất kỳ cơ sở dữ liệu nào với một trình điều khiển JDBC đều được hỗ trợ bởi Hibernate.



Hình 3.4: Cấu trúc Hibernate

Dưới đây là một vài mô tả ngắn gọn về các thành phần trong cấu trúc Hibernate

- Cấu hình đối tượng (Configuration): nó đại diện cho một tập tin cấu hình, cung cấp thông tin về database muốn kết nối đến. Đây cũng là thành phần tạo ra sự kết nối giữa các Java class và các bảng cơ sở dữ liệu.
- SessionFactory: đối tượng này được tạo ra trong quá trình ứng dụng khởi động. Mỗi database sử dụng một tập tin cấu hình riêng biệt và chỉ có 1 đối tượng Session-Factory duy nhất. Đối tượng này có thể được truy cập đồng thời bởi nhiều thread nhưng vẫn đảm bảo tính an toàn dữ liệu (thread-safe).
- Session: đối tượng này được ứng dụng dùng để giao tiếp với database. Các đối tượng Session không nên giữ mở trong thời gian dài vì không an toàn (not thread-safe).
- Transaction: đối tượng này đại diện cho công việc nhỏ (ví dụ như cập nhật, lưu giá trị). Một session thường bao gồm nhiều transaction.
- Query: đối tượng truy vấn sử dụng SQL hoặc Hibernate Query Language (HQL) để lấy dữ liệu từ cơ sở dữ liệu.

CHƯƠNG 3. BACK-END

- Criteria: kết hợp một hay nhiều tiêu chí để truy xuất một thực thể từ database thỏa mãn.

Những lợi ích mà Hibernate đem lại

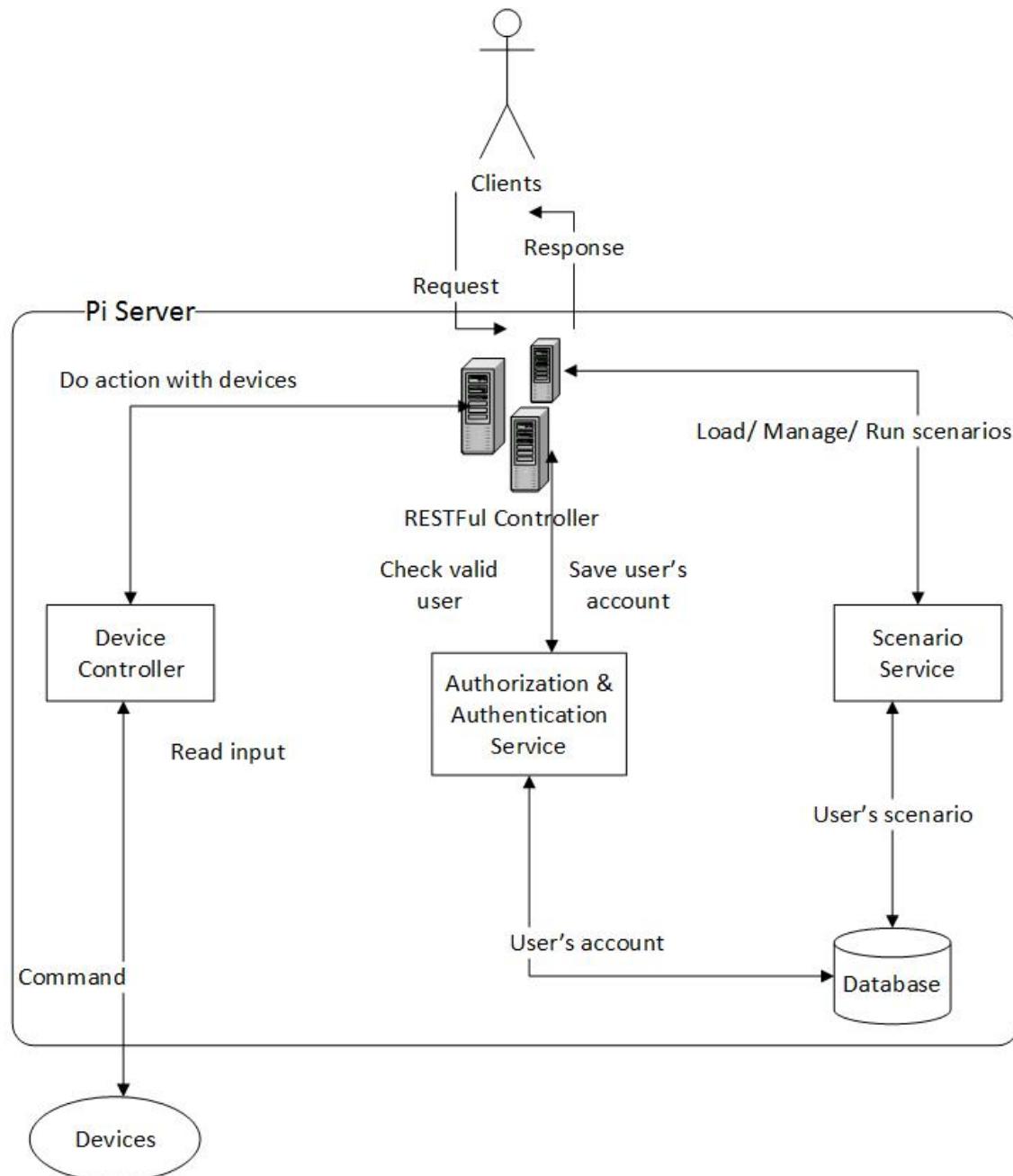
- Hibernate Framework là mã nguồn mở theo LGPL licence và dung lượng nhỏ.
- Đơn giản hóa việc truy nhập, kết nối
- Hibernate Framework cung cấp các thiết bị để tạo ra các bảng tự động
- Hỗ trợ hầu hết các loại database management system thông dụng hiện nay
- Cung cấp cơ chế tự động quản lý cache, cache cấp 1 và cấp 2, giúp tối ưu hóa việc truy xuất dữ liệu.

3.2.4 Kiến trúc back-end

Hình 5 mô tả kiến trúc hệ thống ở back-end, bao gồm các module nhỏ như sau:

- Authorization và Authentication service: phục vụ mục đích bảo mật hệ thống, chỉ những người dùng hợp lệ (có tài khoản hợp lệ, có quyền truy xuất với tài nguyên yêu cầu), quản lý token.
- Scenario Service: quản lý trạng thái các kịch bản (có đang được kích hoạt chạy hay không) hay có thay đổi từ nhà hoặc thiết bị ảnh hưởng đến trạng thái kịch bản; quản lý việc thực thi các kịch bản một cách tự động; kiểm tra tính hợp lệ của kịch bản, xác định xem kịch bản có bị mâu thuẫn với chính nó hay với những kịch bản đã tồn tại hay không; cho phép truy xuất, tạo mới, cập nhật kịch bản.
- Device Service: các kịch bản khi ở trong trạng thái kích hoạt, và thỏa 1 điều kiện định trước do người dùng định nghĩa thì nó sẽ thực thi những hành động tương ứng. Và module này đóng vai trò trung gian trong việc tương tác với thiết bị thật gắn trên Raspberry Pi ở hệ thống back-end, cụ thể là các kịch bản đang chạy.

Chi tiết về hiện thực các module này sẽ nằm trong mục 6.1



Hình 3.5: Kiến trúc hệ thống ở back-end

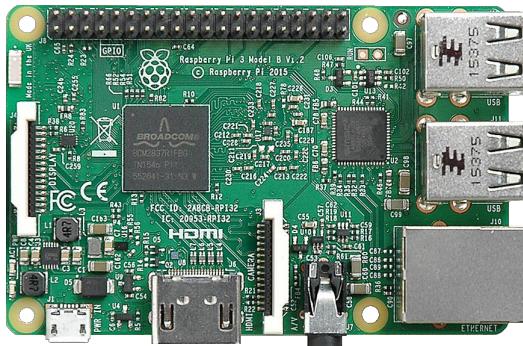
Chương 4

Module điều khiển thiết bị

Điều khiển các thiết bị điện trong nhà là một phần không thể thiếu trong hệ thống nhà thông minh. Server cần một bộ điều khiển có thể lấy được dữ liệu từ các cảm biến đặt tại các nơi khác nhau trong nhà, chụp ảnh từ các camera, bật hoặc tắt các thiết bị điện,... Trong chương này, chúng tôi đưa ra các công trình liên quan trong việc sử dụng máy tính siêu nhỏ Raspberry Pi để giải quyết nhiều vấn đề khác nhau trong cuộc sống. Tiếp đến, nhóm trình bày tổng quan thiết kế của module điều khiển thiết bị sử dụng Raspberry Pi và các công nghệ có thể hỗ trợ để hiện thực module.

4.1 Tổng quan về Raspberry Pi

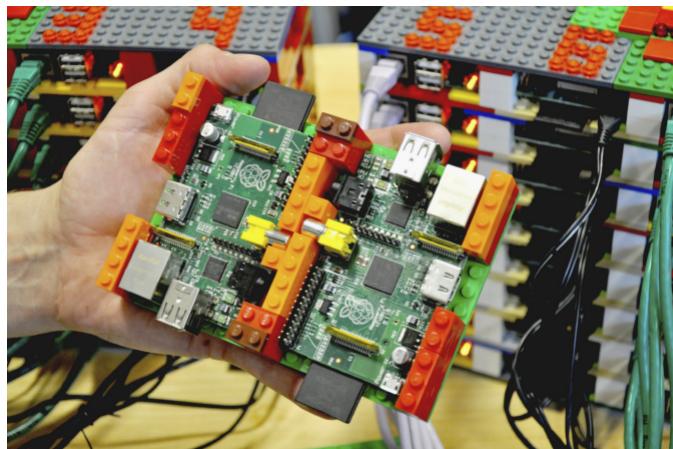
Raspberry Pi được phát triển tại anh bởi Raspberry Pi Foundation với mục đích giúp sinh viên có thể thực hành nhiều hơn trong môn Khoa học máy tính. Nó được thiết kế nhỏ gọn và phải đủ bền để sinh viên có thể giữ trong cặp. Hình 4.1 mô tả cấu trúc bên ngoài của máy tính Raspberry Pi 3 Model B [7][15].



Hình 4.1: Máy tính Raspberry Pi 3 Model B

Giá thành của một con Raspberry Pi khá rẻ, phù hợp với túi tiền của tất cả mọi người. Trên thế giới, đã có rất nhiều sản phẩm hữu ích phục vụ cho đời sống và công việc hằng ngày ra đời nhờ sử dụng máy tính Raspberry Pi. Giáo sư Simon Cox và các đồng nghiệp đã lắp ghép 60 con Raspberry Pi bằng các viên gạch Lego để làm thành một chiếc siêu máy tính. Các máy tính Raspberry Pi này hoạt động song song với nhau và cùng giải quyết một vấn đề. Hình 4.2 chụp lại hai trong số 60 con Raspberry Pi làm ra chiếc siêu máy tính [7].

CHƯƠNG 4. MODULE ĐIỀU KHIỂN THIẾT BỊ



Hình 4.2: Siêu máy tính được chế tạo từ các con Raspberry Pi [2]

Bảng 4.1: Một vài thông số kỹ thuật của Raspberry Pi 3 Model B

Vi xử lý	ARMv8
Xung nhịp	1.2GHz
RAM	1 GB SDRAM
Bộ nhớ trong	MicroSD
Nguồn điện	5V
GPIO	40 pin
Video output	HDMI
USB 2.0	4 cổng
Cổng Ethernet	Có hỗ trợ
Wireless	802.11n / Bluetooth 4.1

Một sở thú ở London, Anh cũng sử dụng Raspberry Pi để sáng tạo ra một công cụ gọi là EyesPi, cho phép chụp ảnh các hoạt động thường ngày của các loài động vật trong sở thú. Một người tên Dave Akerman đã cùng những người đồng nghiệp lắp đặt Raspberry Pi trên một khinh khí cầu, sau đó, họ thả bay khinh khí cầu và chụp những bức ảnh về Trái đất khi nhìn từ trên cao [7].

Các phiên bản: đã có nhiều thế hệ Raspberry Pi được đưa ra thị trường, phiên bản đầu tiên (Raspberry Pi 1 Model B) được ra mắt vào tháng 2 năm 2012. Từ năm 2012 cho đến nay đã có rất nhiều phiên bản Raspberry Pi ra đời với nhiều cải tiến. Raspberry Pi 3 Model B là phiên bản mới nhất tính đến thời điểm hiện tại, được phát hành vào tháng 2 năm 2016, tích hợp thêm bộ thu phát sóng WiFi và Bluetooth so với phiên bản trước. Đây cũng chính là phiên bản được nhóm lựa chọn sử dụng, thông số kỹ thuật được mô tả tại bảng 4.1 [15].

Hệ điều hành: Raspberry Pi đa phần sử dụng một hệ điều hành chuyên dụng là Raspbian. Đây là hệ điều hành mã nguồn mở, được xây dựng dựa trên nền tảng hệ điều hành Debian,

CHƯƠNG 4. MODULE ĐIỀU KHIỂN THIẾT BỊ

được tạo ra bởi một nhóm nhỏ những người hâm mộ Raspberry Pi với mục đích tối ưu hóa cho việc sử dụng phần cứng của Raspberry Pi. Raspbian có khoảng 35000 packages cũng như những phần mềm biên dịch sẵn đính kèm, tiện cho việc cài đặt sử dụng [15][14].

Ngoài hệ điều hành Raspbian, Raspberry Pi còn có thể chạy được Windows 10 IoT Core, một hệ điều hành được Microsoft thiết kế để sử dụng riêng cho hệ thống nhúng, và các hệ điều hành có nhân Linux khác như Ubuntu MATE, Xbian, CentOS,... Trong quá trình phát triển ứng dụng, nhóm quyết định cài đặt hệ điều hành Raspbian trên Raspberry Pi vì đây vẫn là hệ điều hành được sử dụng và hỗ trợ nhiều nhất cho Raspberry Pi.

4.2 Điều khiển thiết bị sử dụng GPIO

Một trong những yếu tố quan trọng của Raspberry Pi là các cổng GPIO (General Purpose Input/Output) được bố trí dọc theo một cạnh của nó. Hình 4.3 miêu tả vị trí đặt các cổng GPIO của Raspberry Pi 3 Model B [2].



Hình 4.3: Vị trí đặt các cổng GPIO của Raspberry Pi 3 Model B

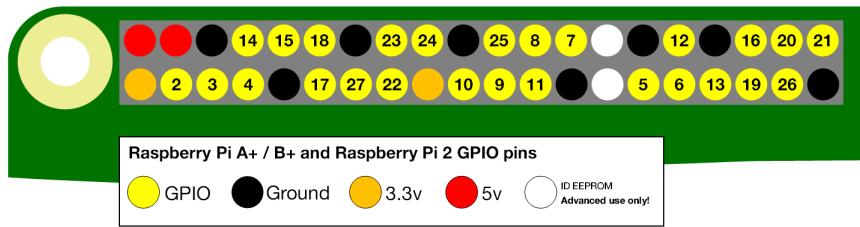
Mỗi cổng GPIO đều có hai trạng thái là High và Low tương ứng với mức điện áp lần lượt là 3.3V và 0V. Có thể xem các cổng GPIO như một cầu nối giữa Raspberry Pi và thế giới bên ngoài, mỗi cổng cũng có thể là input, cũng có thể là output cho tín hiệu số. Nói một cách đơn giản, mỗi cổng GPIO như một công tắc, người bên ngoài có thể tắt hoặc bật, tức thay đổi trạng thái cổng GPIO, lúc này, GPIO thực hiện chức năng là một cổng input. Các chương trình trên Raspberry Pi cũng có thể thay đổi trạng thái các cổng GPIO, GPIO lúc này thực hiện chức năng là một cổng output, truyền tín hiệu từ các chương trình ra ngoài [2].

Trong số 40 cổng của Raspberry Pi 3 Model B, có 26 cổng GPIO, 4 cổng nguồn có điện áp luôn luôn là 3.3V hoặc 5V, 8 cổng đất và 2 cổng ID EEPROM được dùng trong các mục đích đặc biệt. Hình 4.4 mô tả chi tiết phân bố của các cổng trên Raspberry Pi.

Trong số 26 cổng GPIO, một vài cổng có thêm những chức năng khác ngoài việc thay đổi trạng thái giữa High và Low. Những cổng có thêm những chức năng đặc biệt này được gắn thêm một nhãn tên phía sau như mô tả trong hình 4.5:

- UART (Universal Asynchronous Receiver/Transmitter) Serial Bus: Giả sử như chúng ta cắm một thiết bị vào mà nó có khả năng hiển thị dữ liệu lên thì khi cắm vào hai chân này, ta có thể đọc được những thông báo từ kernel của Raspberry Pi. Ví dụ như Pi không thể khởi động lên được, ta dùng cách này như một công cụ chẩn đoán lỗi. Có hai GPIO hỗ trợ chức năng này là GPIO 15 và 16.

CHƯƠNG 4. MODULE ĐIỀU KHIỂN THIẾT BỊ



Hình 4.4: Vị trí phân bố các cổng GPIO

- I2C (Inter-Integrated Circuit) Bus: cung cấp phương thức giao tiếp giữa nhiều ICs với nhau. Đối với Pi, một IC có thể nhắc đến ngay đó là mạch vi xử lý của nó. I2C bus có thể được truy cập từ GPIO 8 (SDA – Serial Data Line) và GPIO 9 (SCL – Serial Clock). Nó được dùng khá phổ biến để kết nối đến các thiết bị ngoại vi như cảm biến nhiệt độ hay màn hình LCD.
- SPI (Serial Peripheral Interface) Bus: phục vụ cho in-system programming (ISP). Các GPIO sau hỗ trợ SPI : GPIO 10 ,11, 12, 13, 14.

4.3 Thiết kế chi tiết

Vai trò chính của module điều khiển thiết bị là nhận yêu cầu từ server gửi đến và thực hiện đúng hành động mà server mong muốn. Để hiện thực chức năng này, các thiết bị được gắn trực tiếp vào một máy tính siêu nhỏ gọi là Raspberry Pi, mỗi thiết bị được điều khiển thông qua các chân cắm gọi là GPIO nằm dọc một cạnh của Raspberry Pi. Hình 4.6 mô tả sự tương tác giữa server, module điều khiển thiết bị và Raspberry Pi.

Ví dụ, khi server muốn thực hiện hành động tắt đèn phòng khách, server sẽ gửi yêu cầu đến module điều khiển thiết bị kèm theo thông tin về thiết bị mà server cần điều khiển, trong trường hợp này là thông tin về chiếc đèn phòng khách. Trong những thông tin mà server gửi đến module điều khiển thiết bị thì số thứ tự cổng GPIO mà thiết bị này kết nối vào là thông tin quan trọng. Vì mỗi thiết bị sẽ được điều khiển bởi một GPIO, module điều khiển thiết bị cần biết chính xác chiếc đèn phòng khách được điều khiển bởi GPIO nào để thực hiện đúng hành động và đúng mục tiêu.

Module điều khiển thiết bị được thiết kế gồm hai phần chính, được trình bày chi tiết tại hình 4.7, bao gồm: bộ điều khiển tổng quát (General controller) và bộ cung cấp cổng GPIO (GPIO provider).

Bộ điều khiển tổng quát (General controller): vai trò của bộ điều khiển tổng quát là nhận tất cả yêu cầu được gửi đến từ server và xử lý tất cả yêu cầu này, xác định xem đây là yêu cầu dành cho loại thiết bị nào và sử dụng đúng bộ điều khiển dành cho loại thiết bị đó. Trong bộ điều khiển tổng quát có chứa nhiều bộ điều khiển con dành cho từng loại thiết bị, tùy vào yêu cầu của server mà bộ điều khiển tổng quát sẽ quyết định sử dụng

CHƯƠNG 4. MODULE ĐIỀU KHIỂN THIẾT BỊ

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I2C)		DC Power 5v	04
05	GPIO03 (SCL1 , I2C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)		(I2C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

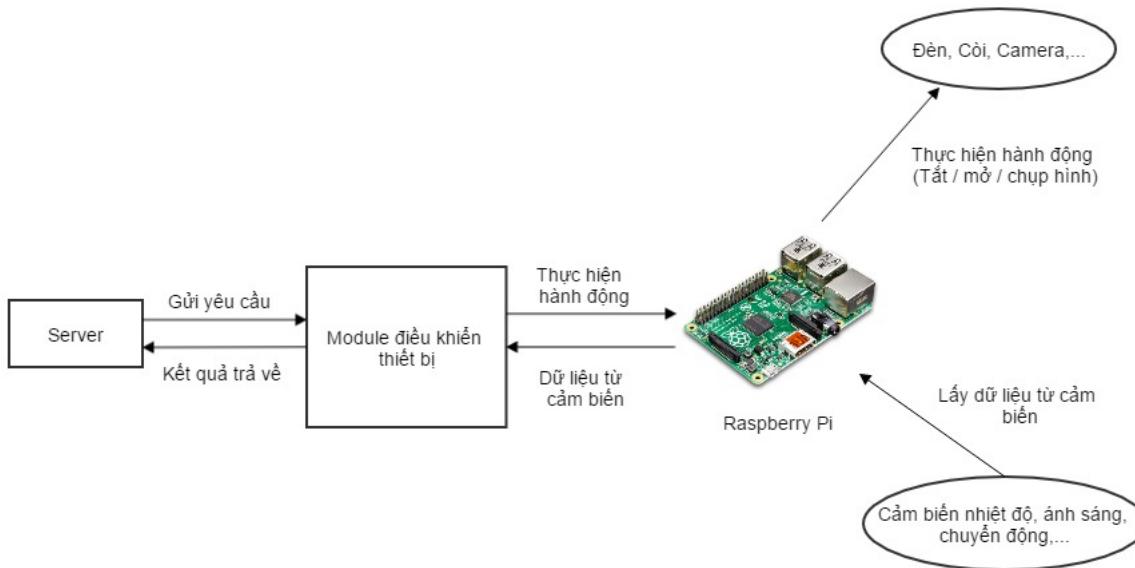
Hình 4.5: Vị trí phân bố các cổng GPIO

bộ điều khiển con nào.

Bộ điều khiển camera (Camera controller) được dùng để chụp ảnh với camera chuyên dụng của Raspberry Pi. Với sự hỗ trợ của thư viện JRPiCam, bộ điều khiển camera có thể dễ dàng điều chỉnh các thông số từ chất lượng ảnh cho đến độ sáng, độ nét và độ tương phản. Các bộ điều khiển cảm biến (Gas sensor controller, Light sensor controller,...) lấy dữ liệu từ các cảm biến thông qua các cổng GPIO. Dữ liệu trả về có thể thuộc dạng số thực, như nhiệt độ hoặc dạng đúng/sai như có gas hay không, trời sáng hay không,...?

Hai bộ điều khiển còn lại là bộ điều khiển đèn (Lightbulb controller) và bộ điều khiển còi (Buzzer controller) có vai trò như một công tắc đóng/mở. Mỗi thiết bị đèn hoặc còi sẽ được điều khiển bởi một cổng GPIO. Như đã trình bày, mỗi cổng GPIO đều có hai trạng thái là Low và High, tùy thuộc vào người thiết kế hệ thống mà từng trạng thái của GPIO sẽ tương ứng với bật hay tắt thiết bị. Trong dự án của mình, nhóm sử dụng trạng thái Low là bật và trạng thái High là tắt.

Bộ cung cấp GPIO (GPIO provider): bộ cung cấp GPIO có vai trò khai báo và cung cấp chính xác loại cổng mà bộ điều khiển thiết bị cần. Với các bộ điều khiển thiết bị dành



Hình 4.6: Sơ đồ tổng quan của module điều khiển thiết bị

cho cảm biến, bộ cung cấp GPIO cần khởi tạo cổng nhập (input pin) có chức năng nhận dữ liệu từ cảm biến. Đối với bộ điều khiển thiết bị cho đèn hoặc còi, bộ cung cấp GPIO cần khởi tạo cổng xuất (output pin) sử dụng cho việc điều khiển trạng thái bật/tắt.

4.4 Công nghệ hỗ trợ

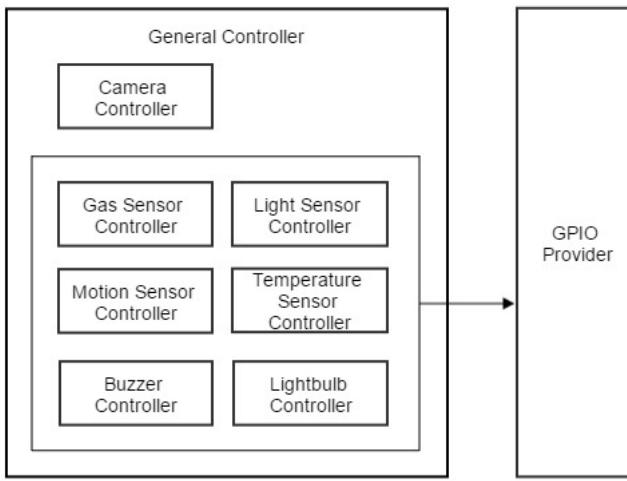
Có rất nhiều thư viện được viết bằng nhiều ngôn ngữ khác nhau hỗ trợ việc điều khiển các cổng GPIO của Raspberry Pi. Bên dưới là một vài công nghệ đang được sử dụng phổ biến hiện nay.

Thư viện Pi4J: Pi4J là một thư viện được viết bằng ngôn ngữ Java với mục đích cung cấp các API (Application Programming Interface) cho người lập trình có thể điều khiển dễ dàng các cổng GPIO mà không cần quan tâm đến cấu trúc bên dưới của Raspberry Pi.

Thư viện WiringPi: cũng là một thư viện được sử dụng để điều khiển các cổng GPIO, thư viện WiringPi được viết bằng ngôn ngữ C, là một thư viện hiệu quả đối với những chương trình viết bằng C hoặc C++. Tuy nhiên, đã có nhiều lập trình viên tạo thêm những phiên bản phù hợp với những loại ngôn ngữ khác như Java, Python, Ruby,...

Thư viện RPi.GPIO: là một thư viện được viết bằng ngôn ngữ Python, RPi.GPIO được tạo ra với mục đích cung cấp những API cần thiết cho người lập trình Python có thể điều khiển các cổng GPIO của Raspberry Pi một cách dễ dàng.

CHƯƠNG 4. MODULE ĐIỀU KHIỂN THIẾT BỊ



Hình 4.7: Thiết kế chi tiết module điều khiển thiết bị

Thư viện JRPiCam: được tạo ra với mục đích cung cấp các API dành cho việc sử dụng camera chuyên dụng của Raspberry Pi, giúp người lập trình dễ dàng hơn trong việc sử dụng và cấu hình camera. JRPiCam được viết bằng ngôn ngữ Java, mã nguồn được công bố công khai tại GitHub.

Thư viện Picamera: giống như JRPiCam, Picamera cung cấp các API sử dụng cho việc điều khiển và cấu hình camera của Raspberry. Tuy nhiên, Picamera được viết bằng ngôn ngữ Python.

Sau khi tìm hiểu các công nghệ có thể hỗ trợ, chúng tôi quyết định chọn thư viện Pi4J sử dụng cho việc lấy dữ liệu và điều khiển trạng thái các cổng GPIO, thư viện JRPiCam sẽ được dùng để hiện thực bộ điều khiển camera. Lý do nhóm quyết định lựa chọn hai thư viện này vì đây đều là những thư viện mã nguồn mở, được công bố công khai tại GitHub. Vì vậy, nhóm có thể tùy chỉnh lại mã nguồn nếu cần thiết để tăng hiệu suất của hệ thống. Bên cạnh đó, phương pháp lập trình nhóm hướng đến là lập trình hướng đối tượng, nên chúng tôi ưu tiên sử dụng những thư viện được viết bằng ngôn ngữ Java.

Với thiết kế của module điều khiển thiết bị đã được nêu ở những phần trên, tuy không thể điều khiển được tất cả các thiết bị có trong nhà, nhưng đủ để nhóm có thể tạo ra một mô hình thí nghiệm với đầy đủ các thiết bị cần thiết như các cảm biến, đèn, còi và camera. Server được cung cấp đầy đủ các API để điều khiển tất cả các thiết bị hiện có phục vụ cho các mục đích khác nhau như bật/tắt đèn hoặc còi, lấy nhiệt độ, kiểm tra khí gas,... Chi tiết hiện thực module điều khiển thiết bị sẽ được trình bày tại phần 6.2.

Chương 5

Ứng dụng di động

Để có thể tương tác với hệ thống cũng như quản lý các thiết bị, kịch bản trong nhà thông minh, một ứng dụng di động với một giao diện đơn giản, dễ dùng dành cho người dùng là một điều tất yếu cần có. Trong chương này, chúng tôi sẽ giới thiệu tổng quan về các chức năng của ứng dụng di động, cũng như kiến trúc khái quát và các thành phần bên trong nó.

5.1 Các chức năng của ứng dụng

Với mục tiêu hỗ trợ người dùng dễ dàng quản lý các ngôi nhà thông minh của mình, cùng với các thiết bị và kịch bản của ngôi nhà, ứng dụng di động được thiết kế để cho phép người dùng đăng ký và đăng nhập vào hệ thống, quản lý và chỉnh sửa thông tin các ngôi nhà được lắp đặt hệ thống và các thiết bị bên trong. Đồng thời, ứng dụng còn cho phép người dùng tạo ra nhiều chế độ khác nhau ứng với từng ngữ cảnh khác nhau (như vắng nhà, trời mưa, trời nóng...) với một bộ các kịch bản định sẵn hay tự tạo riêng biệt và chúng có thể được chuyển đổi qua lại một cách dễ dàng.

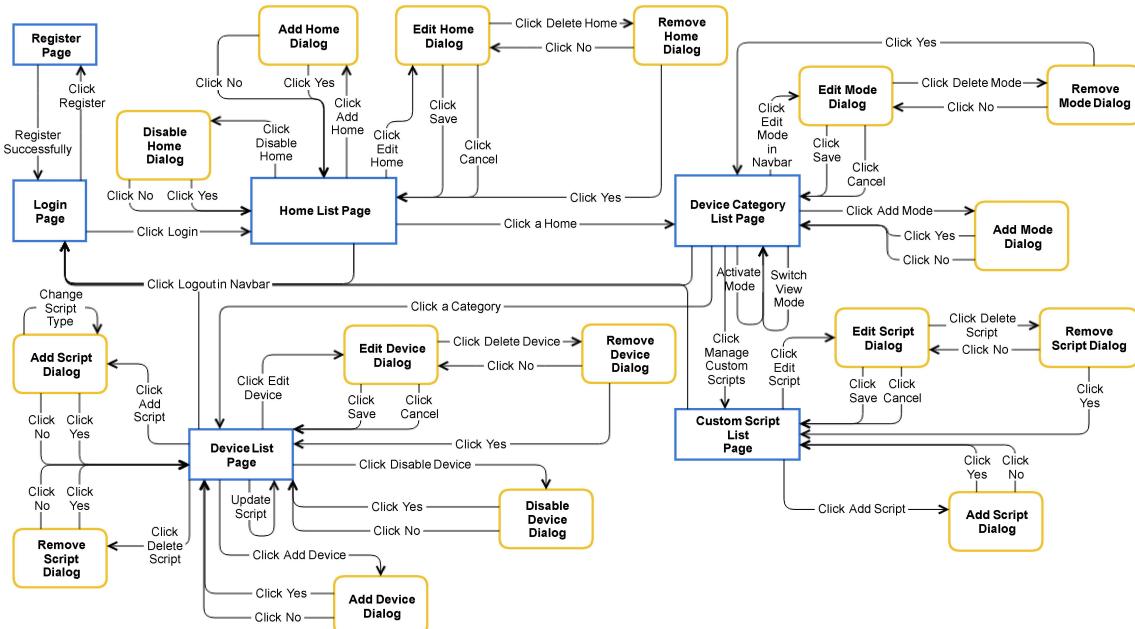
5.2 Kiến trúc tổng quát

Ứng dụng có kiến trúc bao gồm 6 trang giao diện (Page) chính là trang đăng nhập, trang đăng ký, trang danh sách các ngôi nhà, trang danh sách các kiểu thiết bị, trang danh sách các thiết bị và trang danh sách các kịch bản tự tạo. Trong mỗi trang có các bảng hộp thoại (Dialog) hỗ trợ người dùng thực hiện các chức năng của ứng dụng. Hình 5.1 thể hiện luồng chạy và mối liên hệ giữ các trang và bảng hộp thoại trong ứng dụng.

Giao diện Đăng ký (Register Page): Giao diện cho phép người dùng đăng ký tài khoản hệ thống với các thông tin là *Tên đầy đủ*, *Tên đăng nhập*, *Mật khẩu*, *Xác nhận mật khẩu* và *Địa chỉ Email*. Sau khi người dùng nhập đầy đủ các thông tin và gửi lên hệ thống, một Email sẽ được gửi tới địa chỉ Email của người dùng để xác nhận các thông tin đăng ký. Giao diện này không có bảng hộp thoại nào. Từ giao diện này, người dùng có thể đến giao diện đăng nhập sau khi đã gửi thông tin đăng ký.

Giao diện Đăng nhập (Login Page): Giao diện cho phép người dùng đăng nhập vào hệ thống với tên tài khoản và mật khẩu mà người dùng đã đăng ký. Giao diện này không

CHƯƠNG 5. ỨNG DỤNG DI ĐỘNG



Hình 5.1: Sơ đồ luồng Giao diện của ứng dụng di động

có bảng hộp thoại nào. Từ giao diện này người dùng có thể đi đến giao diện đăng ký bằng cách nhấn nút đăng ký (Register) hoặc đến giao diện danh sách các ngôi nhà sau khi đã đăng nhập thành công.

Giao diện Danh sách các ngôi nhà (Home List Page): Danh sách tên các ngôi nhà được lắp đặt hệ thống của người dùng được thể hiện ở Giao diện này. Với mỗi ngôi nhà, khi người dùng nhấn nút chỉnh sửa, bảng hộp thoại chỉnh sửa ngôi nhà sẽ hiện ra, tại đây người dùng có thể xem đầy đủ cũng như cập nhật lại thông tin của ngôi nhà, bao gồm *Tên*, *Địa chỉ* và *Mô tả*. Để xóa một ngôi nhà, người dùng có thể nhấn nút xóa (Remove Home) tại bảng hộp thoại này, một bảng hộp thoại khác hiện lên yêu cầu người dùng xác nhận thao tác xóa. Để thêm mới một ngôi nhà, người dùng có thể nhấn nút thêm mới (Add Home) tại giao diện chính để mở bảng hộp thoại thêm mới yêu cầu người dùng nhập đầy đủ các thông tin về nhà mới. Ngoài ra, người dùng còn có thể dừng hoạt động của hệ thống hoặc cho phép hệ thống hoạt động trở lại tại mỗi ngôi nhà thông qua việc nhấn nút tắt/mở (Disable/Enable). Từ giao diện này, khi nhấn đăng xuất (Logout) từ thanh công cụ, người dùng sẽ về giao diện Đăng nhập hoặc khi nhấn vào một ngôi nhà, người dùng sẽ tới giao diện Danh sách các kiểu thiết bị.

Giao diện Danh sách các kiểu thiết bị (Device Category List Page): Danh sách tên các kiểu thiết bị và danh sách tên các chế độ thuộc ngôi nhà được chọn sẽ được thể hiện ở Giao diện này. Tại đây, người dùng có thể chuyển đổi chế độ (Mode) thông qua hộp

CHƯƠNG 5. ỨNG DỤNG DI ĐỘNG

trình đơn thả xuống (Dropdown box) hoặc kích hoạt (Activate) một chế độ bất kì bằng cách nhấn vào tên chế độ đó tại thanh công cụ. Để chỉnh sửa thông tin hoặc xóa một chế độ nào đó, người dùng nhấn vào nút chỉnh sửa cạnh tên chế độ đó tại thanh công cụ. Để thêm mới một chế độ, người dùng có thể nhấn nút thêm mới (Add Mode) tại giao diện chính để mở bảng hộp thoại thêm mới yêu cầu người dùng nhập đầy đủ các thông tin về chế độ mới. Từ giao diện này, khi nhấn đăng xuất (Logout) từ thanh công cụ, người dùng sẽ về giao diện đăng nhập, khi nhấn vào một thiết bị, người dùng sẽ tới giao diện danh sách các thiết bị hoặc khi nhấn vào nút quản lý kịch bản tự tạo (Manage Custom Scripts), người dùng sẽ tới giao diện danh sách các kịch bản tự tạo.

Giao diện Danh sách các thiết bị (Device List Page): Danh sách tên, chân GPIO các thiết bị thuộc kiểu thiết bị được chọn cũng như danh sách các kịch bản định sẵn thuộc chế độ được chọn của từng thiết bị được thể hiện ở giao diện này. Với mỗi thiết bị, khi người dùng nhấn nút chỉnh sửa, bảng hộp thoại chỉnh sửa thiết bị sẽ hiện ra, tại đây người dùng có thể xem đầy đủ cũng như cập nhật lại thông tin của thiết bị, bao gồm *Tên*, *Mô tả* và *Vị trí*. Để xóa một thiết bị, người dùng có thể nhấn nút xóa (Remove Device) tại bảng hộp thoại này, một bảng hộp thoại khác hiện lên yêu cầu người dùng xác nhận thao tác xóa. Để thêm mới một thiết bị, người dùng có thể nhấn nút thêm mới (Add Device) tại giao diện chính để mở bảng hộp thoại thêm mới yêu cầu người dùng nhập đầy đủ các thông tin về thiết bị mới. Danh sách các kịch bản thuộc một thiết bị sẽ được thể hiện khi người dùng nhấn vào thiết bị đó, người dùng có thể xóa Kịch bản bằng cách nhấn nút xóa ở đầu mỗi kịch bản hoặc cập nhật lại thông tin kịch bản như *Điều kiện*, *Hành động* hay *Thời gian bắt đầu*, *Thời gian kết thúc* thông qua các hộp trình đơn thả xuống. Ngoài ra để thêm mới một kịch bản, người dùng có thể nhấn nút thêm mới Kịch bản (Add Script), bảng hộp thoại thêm mới kịch bản hiện ra, người dùng cần chọn loại kịch bản muốn thêm và xác định nội dung cho kịch bản đó. Từ giao diện này, khi nhấn đăng xuất (Logout) từ thanh công cụ, người dùng sẽ về giao diện đăng nhập hoặc khi nhấn vào biểu tượng ngôi nhà (Home) trên thanh công cụ, người dùng sẽ quay lại giao diện danh sách các thiết bị.

Giao diện Danh sách các Kịch bản tự tạo (Custom Script List Page): Danh sách tên các kịch bản người dùng tự tạo thuộc chế độ được chọn sẽ được thể hiện ở giao diện này. Với mỗi kịch bản tự tạo, khi người dùng nhấn nút chỉnh sửa, bảng hộp thoại chỉnh sửa kịch bản tự tạo sẽ hiện ra, tại đây người dùng có thể xem cũng như cập nhật lại nội dung và tên của kịch bản. Để thêm mới một kịch bản tự tạo, người dùng có thể nhấn nút thêm mới kịch bản tự tạo (Add Custom Script), bảng hộp thoại thêm mới kịch bản tự tạo hiện ra yêu cầu người dùng xác định tên và nội dung cho kịch bản mới. Từ giao diện này, khi nhấn đăng xuất (Logout) từ thanh công cụ, người dùng sẽ về giao diện đăng nhập hoặc khi nhấn vào biểu tượng ngôi nhà (Home) trên thanh công cụ, người dùng sẽ quay lại giao diện danh sách các thiết bị.

Với cách thiết kế như trên, kiến trúc ứng dụng di động cung cấp một giao diện tương đối đơn giản nhưng vẫn đáp ứng được các nhu cầu cơ bản về việc quản lý các ngôi nhà, thiết bị, chế độ và các kịch bản điều khiển thiết bị của người dùng.

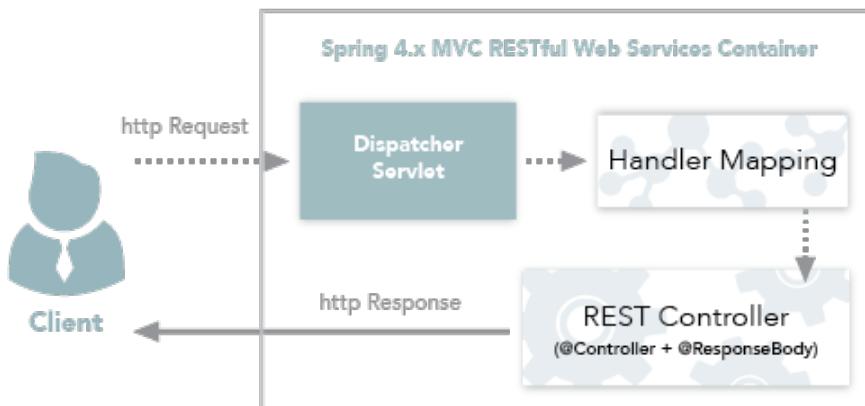
Chương 6

Hiện thực hệ thống

6.1 Hiện thực back-end

6.1.1 RESTful Web Service - Cách thức giao tiếp giữa client và server

Như đã đề cập ở phần Thiết kế back-end, nhóm đã chọn dùng RESTful web service làm cách giao tiếp chính giữa client và server. Một tiện ích khi sử dụng Spring framework đó là nó có hỗ trợ sẵn @RestController (là 1 annotation hỗ trợ bởi Spring framework), đơn giản hóa việc tạo ra các RESTful web services.



Hình 6.1: Spring MVC RESTful Web services workflow

Hình 6.1 diễn tả luồng thực thi của Spring MVC REST, bao gồm các bước sau:

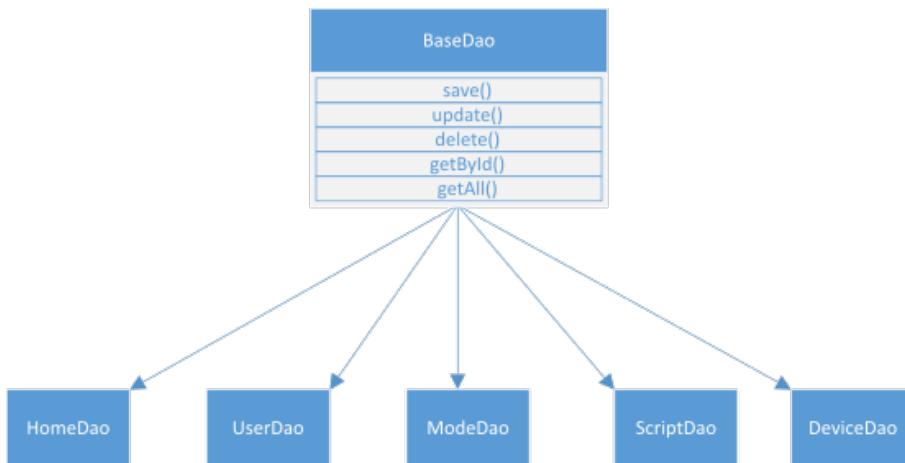
- Client gửi yêu cầu đến web service theo như một định dạng URI nào đó có sẵn và hợp lệ
- Yêu cầu đi qua Servlet Dispatcher đầu tiên và nó sẽ tìm ra 1 controller phù hợp nhất để xử lý yêu cầu đó
- Yêu cầu sau khi được xử lý bởi controller sẽ được gửi trả về client dưới định dạng JSON [13].

Danh sách API có thể được tham khảo thêm ở mục **Phụ lục**.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

6.1.2 Cách thức giao tiếp với database

Như đã đề cập ở mục thiết kế hệ thống back-end bên trên, nhóm sử dụng Hibernate framework để hỗ trợ cho các thao tác liên quan đến database. Hibernate cung cấp sẵn các hàm giúp truy xuất, lưu, cập nhật, xóa thực thể liên quan. Hình 6.2 thể hiện tầng thao tác dữ liệu (DAO) trong ứng dụng mà nhóm đã thiết kế và hiện thực.



Hình 6.2: Tổ chức của tầng truy xuất dữ liệu (DAO)

Thiết kế này giúp tăng khả năng tái sử dụng (reuse), cũng như việc quản lý, bảo trì, mở rộng hệ thống được dễ dàng hơn trong tương lai. Ý tưởng cơ bản là có 1 class `BaseDao`, được hiện thực đầy đủ các hàm `save`, `update`, `delete`,... còn các thực thể khác (như Home, User, Mode, Device, Script) thì thừa kế class `BaseDao` này và hiện thực thêm một số phương thức khác tùy theo nhu cầu.

Nếu những thao tác với database gây ra lỗi, dữ liệu sẽ được rollback ngay thời điểm đó (ví dụ như vi phạm ràng buộc, khóa ngoại,...), nhằm đảm bảo tính nhất quán của dữ liệu.

6.1.3 Giới thiệu về kịch bản(scenario)

Kịch bản là một bản phác thảo, diễn tả những hành vi mình mong muốn thiết bị trong nhà sẽ tự động thực hiện trong hoàn cảnh nhất định hay điều kiện nào đó thỏa mãn.

Kịch bản người dùng

Kịch bản người dùng sử dụng ngôn ngữ tự nhiên để đặc tả. Lấy ví dụ như “Trong khoảng thời gian từ 18h tối đến 22h tối thì bật đèn ở hành lang lên”. Vẽ đầu “trong khoảng thời gian từ 18h tối đến 22h tối” đặc tả điều kiện, vẽ sau “bật đèn hành lang” nêu ra hành động mong muốn khi mà điều kiện trên thỏa mãn.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

Kịch bản lưu trữ (script)

Kịch bản lưu trữ có tác dụng như một bản thảo ngắn gọn của kịch bản người dùng, phục vụ mục đích lưu trữ dưới cơ sở dữ liệu là chính. Kịch bản lưu trữ được đặc tả bởi văn phạm riêng, được giới thiệu ở mục 6.1.4 sắp tới.

Kịch bản hệ thống (scenario)

Kịch bản hệ thống là dạng kịch bản mà hệ thống có khả năng “đọc”, “hiểu” và xử lý. Hệ thống đọc kịch bản lưu trữ và dùng 1 cấu trúc dữ liệu riêng (sẽ được giới thiệu ở mục 6.1.5) để mô tả nó và xây dựng lên thành kịch bản hệ thống (lưu trên bộ nhớ máy tính).

6.1.4 Văn phạm (grammar) dùng tạo ra kịch bản lưu trữ

Kịch bản của người dùng thường được đặc tả bởi ngôn ngữ tự nhiên. Với hệ thống hiện tại của nhóm, tính năng xử lý ngôn ngữ tự nhiên không được hỗ trợ, do đó vấn đề cấp thiết đặt ra đầu tiên và cũng không kém phần quan trọng, chính là đặc tả văn phạm cho kịch bản. Đặc tả văn phạm kịch bản nhằm một mặt giúp hệ thống có thể lưu trữ, mặt khác là phân định được kịch bản nào là hợp lệ và kịch bản nào không hợp lệ. Hơn thế nữa, văn phạm còn giúp hệ thống có thể “đọc”, “hiểu” và xử lý kịch bản. Tuy nhiên, công việc khó khăn là làm sao văn phạm đặc tả chính xác được kịch bản mà vẫn giữ đúng ý nghĩa của nó. Sau thời gian nghiên cứu, nhóm quyết định chọn BNF (Backus-Naur form), gồm những ký hiệu toán học để đặc tả văn phạm cho ngôn ngữ phi ngôn ngữ cảnh, thường dùng để xây dựng cú pháp các ngôn ngữ trong ngành máy tính, ví dụ như ngôn ngữ lập trình, tập lệnh... áp dụng vào việc xây dựng văn phạm kịch bản hệ thống.

Hình 6.3 là mô tả văn phạm mà nhóm dùng để mô tả những kịch bản lưu trữ, bao gồm từ khóa (đặt trong cặp ngoặc kép), ký hiệu, phép toán, cấu trúc (đặt trong <>).

Nhóm đã chọn đặc tả văn phạm kịch bản với cú pháp nêu trên, nhằm mục đích:

- Có thể dễ dàng lưu trữ kịch bản.
- Cú pháp tương tự như dạng JSON, nhóm có thể dùng parser JSON để phân tích và chuyển đổi thành kịch bản hệ thống dễ dàng hơn.

Tuy nhiên, với cú pháp trên cũng có mặt hạn chế với cú pháp như việc tạo ra kịch bản lưu trữ cần phải được xử lý cẩn thận. Nếu khâu đầu vào có sai sót thì toàn bộ khâu còn lại, như đọc và xử lý sẽ gặp vấn đề. Để khắc phục hạn chế này, nhóm đã xây dựng 1 module nhằm tạo ra kịch bản hệ thống theo cú pháp nhất định (sẽ được giới thiệu trong mục 6.1.8).

Để dễ hình dung, sau đây là một ví dụ trong thực tế về 1 kịch bản mà ta đang muốn hệ thống đọc, hiểu và xử lý:

- “Nếu cảm biến nhiệt phát hiện nhiệt độ nằm trong khoảng 40 đến 50 độ C thì bật còi hú ở phòng khách”.
- “Trong khoảng thời gian từ 18h tối đến 22h tối thì bật đèn ở hành lang lên”.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

```
<Scenario> ::= "[" <ControlBlock> "] | "[" <Scenario> "," <ControlBlock> "]"
| "[" <SimpleAction> "]" | "[" <Scenario> "," <SimpleAction> "]"

<ControlBlock> ::= "[If , " <Condition> "," <Action> "]"
| "[If , " <Condition> "," <Action> "," <Action> "]"
| <FromToBlock>

<Condition> ::= "[" <DeviceName> "," <RelationalOperator> "," <Value> "]"

<Action> ::= <Scenario>

<FromToBlock> ::= "[FromTo," Time "," Time "," <Action> "]"

<RelationalOperator> ::= <Equal> | <NotEqual> | <GreaterThan >
| <GreaterThanOrEqualTo> | <LessThan> | <LessThanOrEqualTo>

<SimpleAction> ::= "[TurnOn," <DeviceName> "]"
| "[TurnOff," <DeviceName> "]"
| "[Toggle," <DeviceName> "]"
| "[TakePicture," <DeviceName> "]"

<DeviceName> ::= String

<Value> ::= Long | Boolean

<Equal> ::= "="

<NotEqual> ::= "!="

<GreaterThan> ::= ">"

<GreaterThanOrEqualTo> ::= ">="

<LessThan> ::= "<"

<LessThanOrEqualTo> ::= "<="
```

Hình 6.3: Văn phạm kịch bản lưu trữ

Sử dụng văn phạm đã có, kịch bản lưu trữ dùng mô tả các kịch bản trên sẽ là:

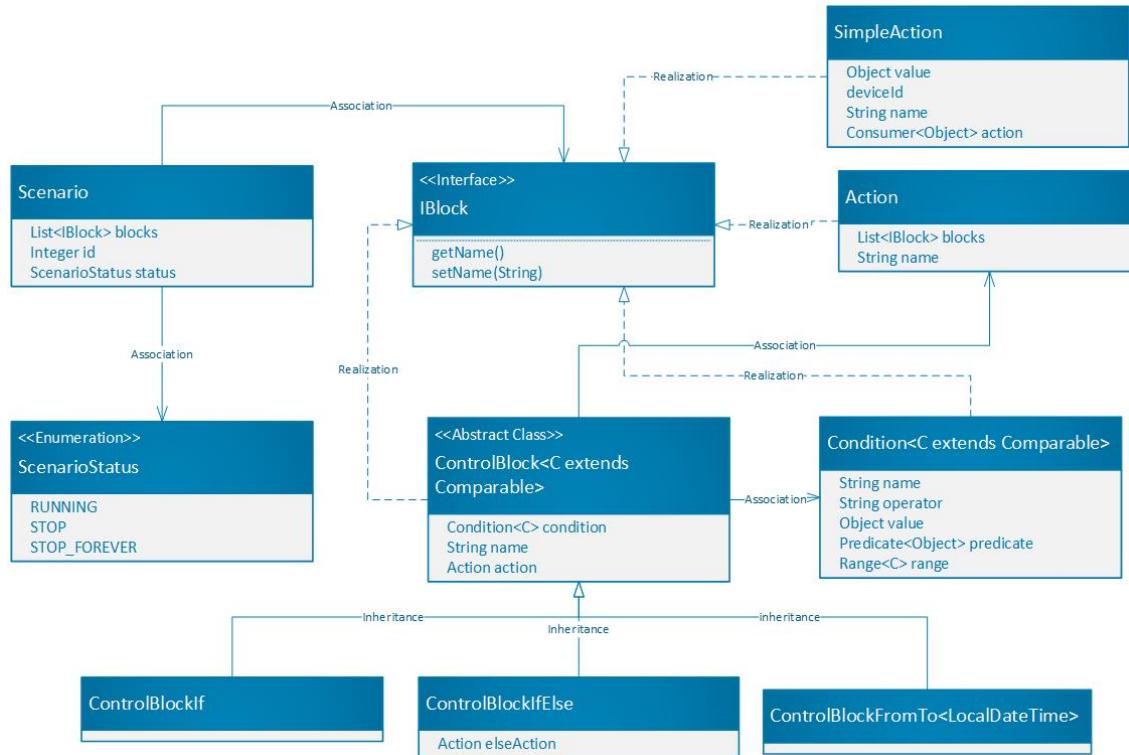
- “[[‘If’, [‘temperature_sensor’,’>’, 40], [[‘If’, [‘temperature_sensor’,’<’, 50], [[‘TurnOn’, ‘buzzer_living_room’]]]]]”
- “[[‘FromTo’, ’18:00’, ’20:00’, [[‘TurnOn’, ‘light_lobby’]]]]”

Với văn phạm đặc tả trên, hệ thống đã có thể phân định được kịch bản hợp lệ và không hợp lệ. Vì lý do thời gian nên nhóm chưa thể hỗ trợ nhiều dạng kịch bản hơn, nhưng việc

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

mở rộng là hoàn toàn khả thi. Nhóm cũng đã liệt kê một số kịch bản thông dụng trên thực tế mà nhóm đã sưu tập và lặp nêu (tham khảo thêm ở phần **Phụ lục**).

6.1.5 Cấu trúc dữ liệu xây dựng kịch bản hệ thống



Hình 6.4: Cấu trúc dữ liệu kịch bản hệ thống

Kịch bản hệ thống sẽ được lưu trữ trong bộ nhớ hệ thống khi chạy. Hình 6.4 cho ta thấy cấu trúc dữ liệu của kịch bản hệ thống (scenario) bao gồm:

- Thông tin trạng thái của nó, được định nghĩa bằng 1 enum ScenarioStatus là
 - RUNNING: đang được thực thi
 - STOP: đã dừng lại (tạm thời và sau đó có thể được khởi động lại)
 - STOP_FOREVER: dừng lại vĩnh viễn, trạng thái này dùng để kiểm soát những kịch bản trong hàng chờ được gỡ bỏ
- Tập hợp các khối (block) hiện thực từ interface IBlock, các khối hợp lệ và được phép nằm trong kịch bản sẽ là
 - ControlBlock (sẽ chứa tham khảo đến Condition)
 - Action
 - SimpleAction

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

Class SimpleAction

Class SimpleAction đại diện cho những hành động nhỏ nhất, đơn giản nhất, có thể tương tác với thiết bị. Lấy ví dụ như hành động “Tắt đèn 1” hay “Bật còi hú ở bếp” là những hành động đơn giản.

Thuộc tính “action” có kiểu Consumer<Object> nhằm để mình truyền vào 1 biểu thức dạng Lamda Expression, cụ thể nó là 1 hàm gọi tới Device controller , tương tác với thiết bị thật. Chi tiết về phần hiện thực sẽ được giải thích trong mục Module Script Creator.

Class Action

Class Action là tập hợp các hành động mong muốn thực hiện, có thể xem như là 1 kịch bản hệ thống “con” trong kịch bản hệ thống lớn bên ngoài và chứa nó. Vì thế mà cấu trúc của nó gần tương tự như là 1 kịch bản hệ thống vậy.

Class Condition

Class Condition đại diện cho 1 điều kiện nào đó. Lấy 1 ví dụ như “Nhiệt độ lớn hơn 40 độ C thì bật còi hú 1”. Phân tích kịch bản trên thì điều kiện ở đây chính là “Nhiệt độ lớn hơn 40 độ C”. Khi đó, thuộc tính “operator” sẽ có giá trị “>”, thuộc tính “value” có giá trị “40”. Một thuộc tính đặc biệt là range, kiểu Range<C> mang giá trị dãy số (40, +infinity), phục vụ cho mục đích kiểm tra kịch bản mâu thuẫn (giới thiệu ở phần module ScenarioConflictValidation).

Thuộc tính predicate có kiểu Predicate<Object> sẽ chứa 1 biểu thức Lamda Expression, tương ứng với điều kiện mà mình mô tả trong kịch bản. Chi tiết về hiện thực sẽ nằm trong mục Module Script Creator.

Class ControlBlock và các class kế thừa từ nó

Class ControlBlock được sinh ra nhằm mục đích kết nối 2 class: Action và Condition lại với nhau. Class ControlBlock này có 3 class kế thừa từ nó là

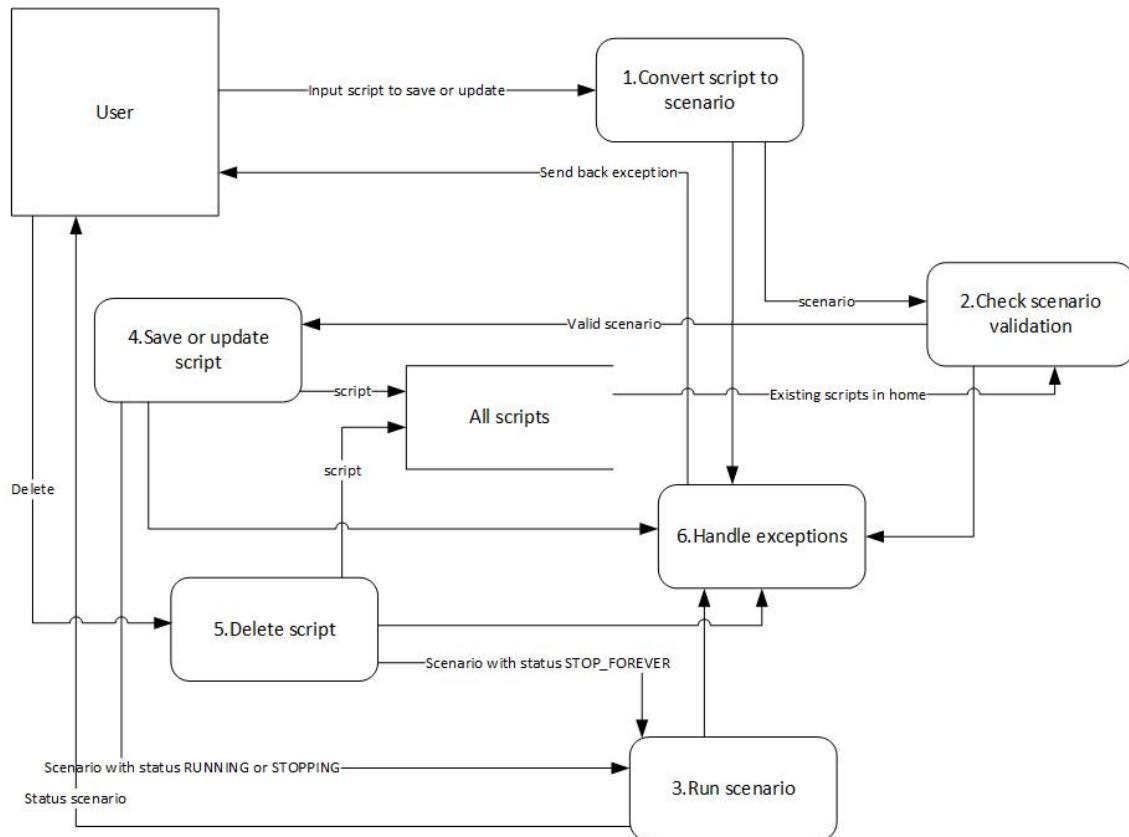
- ControlBlockIf: ngữ nghĩa cơ bản là “Nếu điều kiện A xảy ra thì mình sẽ thực hiện hành động B”.
- ControlBlockIfElse: nó mang nghĩa “Nếu điều kiện A xảy ra thì mình sẽ thực hiện hành động B còn không thì sẽ thực hiện hành động C”. Vì thế mà nó có thêm 1 thuộc tính là “elseAction”, diễn tả hành động mong muốn được thực thi nếu mệnh đề điều kiện không thỏa mãn.
- ControlBlockFromTo: đây là khối điều khiển chuyên dụng cho các điều kiện liên quan tới thời gian, hiểu là “Trong khoảng thời gian từ X đến Y thì mình sẽ thực hiện hành động A”. Khoảng thời gian này có thể là trong cùng một ngày, hay kéo

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

dài từ ngày này sang ngày tới. Ví dụ “Từ 18h00 đến 22h00 thì tắt đèn 1” hay “Từ 19h00 đến 01h00 sáng hôm sau thì bật đèn hành lang”.

Nếu như sau này ứng dụng mở rộng và hỗ trợ thêm nhiều khối điều khiển khác, ta có thể kế thừa từ class ControlBlock này và tiếp tục hiện thực nó một cách dễ dàng.

6.1.6 Sơ đồ mô tả luồng dữ liệu trong hệ thống khi thêm, sửa, xóa kịch bản người dùng



Hình 6.5: Sơ đồ luồng dữ liệu khi thêm/sửa/xóa kịch bản

Một trong những chức năng chính của hệ thống đó là giúp người dùng quản lý thiết bị trong nhà một cách tự động, theo kịch bản được định sẵn. Hình 9 mô tả sơ đồ luồng dữ liệu của hệ thống back-end khi mà người dùng thêm, sửa hay xóa một kịch bản.

Các tiến trình xử lý lần lượt theo các bước sau:

- Khi người dùng thêm, sửa 1 kịch bản qua giao diện, phía client sẽ gửi về cho server thông tin kịch bản đó dưới dạng kịch bản lưu trữ.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

- Kịch bản lưu trữ được chuyển đổi thành kịch bản hệ thống, được lưu trong bộ nhớ dưới 1 cấu trúc dữ liệu riêng bởi module **1.Convert script to scenario** (sẽ đi vào chi tiết ở mục module scenario creator).
- Kịch bản hệ thống trên được kiểm tra tính hợp lệ, cũng như đảm bảo không mâu thuẫn (conflict) với những kịch bản khác hiện có trong ngôi nhà ấy bởi module **2.Check scenario validation** (sẽ đi vào chi tiết ở mục module Scenario Validation). Nếu như kịch bản không hợp lệ, thông báo lỗi sẽ được gửi trả về ngay cho người dùng.
- Sau khi kịch bản được kiểm tra tính hợp lệ, nó sẽ được lưu hay cập nhật vào cơ sở dữ liệu (mình sẽ lưu ở dạng kịch bản lưu trữ).
- Việc cuối cùng là hệ thống sẽ “đọc” và xử lý kịch bản ấy. Việc quản lý đó được module **3.Run scenario** đảm nhiệm (sẽ đi vào chi tiết ở mục module Scenario Runner). Trạng thái của kịch bản mới sẽ được quyết định bởi người dùng, nó có thể là chưa được chạy (STOP) hay được chạy (RUNNING). Thông thường khi 1 kịch bản mới được thêm vào hệ thống sẽ mang trạng thái được chạy.
- Trong trường hợp mà người dùng muốn xóa một kịch bản, hệ thống back-end kiểm tra dưới cơ sở dữ liệu và gửi truy vấn xóa. Sau đó, hệ thống cập nhật lại trạng thái của kịch bản ấy là đã dừng vĩnh viễn (STOP_FOREVER) và từ đó kịch bản này không còn có ảnh hưởng tới các thiết bị trong nhà nữa.

6.1.7 Module parser chuyển đổi kịch bản lưu trữ thành kịch bản hệ thống (scenario creator)

Kịch bản lưu trữ chỉ phục vụ cho mục đích lưu dữ liệu. Khi hệ thống muốn đọc, hiểu, xử lý và chạy những kịch bản ấy thì nó cần được chuyển sang dạng kịch bản hệ thống. Để phục vụ cho nhu cầu đó, module scenario creator được sinh ra. Đầu vào của nó là một kịch bản lưu trữ và đầu ra sẽ là kịch bản hệ thống. Kịch bản hệ thống sẽ được dùng trong phần kiểm tra tính hợp lệ và chạy kịch bản như đã đề cập ở mục tổng quan phía trên.

Kịch bản lưu trữ được thể hiện dưới dạng mảng JSON. Sở dĩ nhóm chọn cách thể hiện này là vì thư viện parse JSON đã có sẵn, việc tận dụng nó sẽ giúp giảm thời gian hiện thực hệ thống này. Mỗi một mảng JSON tương đương với 1 khối: điều kiện (condition) hay hành động (action) hay hành động đơn giản (simple action). Và cũng trong module này, nhóm đã áp dụng nhiều kĩ thuật mang tên Lamda Expression được hỗ trợ trong Java 8 để hiện thực việc xử lý 1 điều kiện hay là 1 hành động đơn giản tương tác với thiết bị thật.

Khi xử lý 1 điều kiện, nhóm dùng Predicate class để hiện thực nó. Predicate như một mệnh đề luân lý trả về giá trị hoặc đúng, hoặc sai, rất phù hợp cho việc kiểm tra điều kiện. Một vài điều kiện cơ bản mà hệ thống hiện có đó là: kiểm tra xem đèn có được bật hay không, kiểm tra ngày hay đêm từ cảm biến ánh sáng, thực hiện so sánh nhiệt độ thu được từ cảm biến nhiệt với một giá trị người dùng mong muốn... Mỗi predicate sẽ nhận việc gọi đến

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

device controller và kiểm tra giá trị trả về từ controller ấy. Nói cách khác thì khi hệ thống kiểm tra một điều kiện nào đó thì sẽ chỉ cần kiểm tra thuộc tính predicate bên trong cấu trúc kịch bản mà không phải gọi riêng lẻ một device controller nào. Nó giúp cho việc code được ngắn gọn và “sạch sẽ” hơn. Tương tự như việc xử lý 1 hành động đơn giản, nó có 1 thuộc tính “action” kiểu Consumer. Khi mà muốn thực hiện hành động, mình chỉ cần gọi phương thức trên biến “action” đó là đủ.

6.1.8 Module hỗ trợ xây dựng kịch bản tùy ý (script builder)

Hệ thống không chỉ cung cấp cho người dùng những kịch bản theo mẫu định sẵn, mà còn hỗ trợ cả kịch bản tùy biến. Để dễ dàng hơn cho người dùng định nghĩa 1 kịch bản tùy biến, nhóm tạo ra 1 module là Script Builder. Cú pháp của kịch bản người dùng tùy biến gần giống như ngôn ngữ tự nhiên, nhưng lại có khả năng chuyển đổi sang dạng kịch bản lưu trữ dễ dàng. Hình 6.6 mô tả cú pháp của văn phạm kịch bản tùy biến.

Lấy ví dụ về 1 kịch bản người dùng là: “Nếu nhiệt độ thu được từ cảm biến nhiệt gần cửa sổ lớn hơn 40 độ thì bật đèn phòng và bật đèn nhà bếp”.

Kịch bản người dùng tùy biến được viết lại theo văn phạm trên sẽ là:

```
If('temp_sensor_near_window', '>', '40')
.action('TurnOn', 'light_room')
.action('TurnOn', 'light_kitchen')
.endIf()
```

Nhiệm vụ của client là sẽ mang kịch bản người dùng tùy biến trên gửi lên server, server sẽ dùng module ScriptBuilder để mà chuyển đổi sang kịch bản lưu trữ. Đi vào chi tiết module ScriptBuilder thì nó gồm 1 bộ biên dịch, nhằm thực thi một đoạn code được lưu dưới dạng chuỗi. Sở dĩ nhóm đề xuất văn phạm, cú pháp trên cho kịch bản tùy biến vì nó chính là đoạn code thu nhỏ, sử dụng ngôn ngữ Java. Nhiệm vụ của bộ biên dịch là đọc kịch bản đó giống như đọc và thực thi 1 đoạn code Java. Kết quả trả về từ đoạn code (hay kịch bản tùy biến) sẽ là một kịch bản lưu trữ. Bằng cách này, nhóm có thể cung cấp cho người dùng những cú pháp khác linh hoạt hơn và tiện lợi hơn (syntactic sugar), giúp người dùng định nghĩa một kịch bản tùy biến dễ dàng và thuận tiện nhất.

Trước khi hiện thực module ScriptBuilder này, nhóm cũng đã tìm hiểu sơ qua các Rule Engine hỗ trợ xây dựng kịch bản. Theo tìm hiểu của nhóm, Rule Engine sẽ hỗ trợ cho ta định nghĩa ra các quy tắc cứng, đã được định nghĩa sẵn (thông thường là business rule) dành cho hệ thống trong một số trường hợp cụ thể nào đó. Khi ta muốn thay đổi các quy tắc ấy thì cần khởi chạy lại hệ thống để có hiệu lực. Chính vì thế mà Rule Engine không phù hợp với ứng dụng nhóm muốn phát triển. Có thể lý giải rằng các kịch bản người dùng đặt ra không phải là các business rule của hệ thống. Hơn nữa, những kịch bản ấy có tính linh hoạt, người dùng có thể cập nhật nội dung mới và nó tự động có hiệu lực ngay sau đó.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

```
<Scenario> ::= "[" <ControlBlock> "] | ["<Scenario> "," <ControlBlock> "]"
| "[" <SimpleAction> "]" | "[" <Scenario> "," <SimpleAction> "]"

<ControlBlock> ::= "If" <Condition> "." <Action> ".endIf()"
| "If" <Condition> "." <Action> ".Else()." <Action> ".endIf()"
| <FromToBlock>

<Condition> ::= "(" <DeviceName> "," <RelationalOperator> "," <Value> ")"

<Action> ::= <Scenario>

<FromToBlock> ::= "FromTo("Time","Time")." <Action> ".endFromTo()"

<RelationalOperator> ::= <Equal> | <NotEqual> | <GreaterThan>
| <GreaterThanOrEqual> | <LessThan> | <LessThanOrEqual>

<SimpleAction> ::= "action( 'TurnOn'," <DeviceName> ")"
| "action( 'TurnOff'," <DeviceName> ")"
| "action( 'Toggle'," <DeviceName>)"
| "action( 'TakePicture'," <DeviceName>)"

<DeviceName> ::= String

<Value> ::= Long | Boolean

<Equal> ::= "="

<NotEqual> ::= "!="

<GreaterThan> ::= ">"

<GreaterThanOrEqual> ::= ">="

<LessThan> ::= "<"

<LessThanOrEqual> ::= "<="
```

Hình 6.6: Văn phạm kịch bản tùy biến

6.1.9 Giới thiệu về tính hợp lệ của kịch bản

Thế nào là kịch bản tự mâu thuẫn (self-conflict script)

Kịch bản tự mâu thuẫn là kịch bản chứa các điều kiện mâu thuẫn trong nội tại chính nó. Lấy ví dụ: Nếu như nhiệt độ thu được từ cảm biến nhiệt ở phòng khách lớn hơn 40 độ và bé hơn 30 độ thì bật đèn 1. Dễ dàng nhận thấy rằng 2 điều kiện “nhiệt độ lớn hơn 40 độ” và “nhiệt độ bé hơn 30 độ” mâu thuẫn lẫn nhau khi giá trị đó cùng thu thập được từ cùng 1 thiết bị. Vì thế, chúng ta có thể kết luận rằng kịch bản nêu trên là kịch bản tự mâu

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

thuẫn. Nhưng nếu điều kiện nhiệt độ bé hơn 30 độ thu được từ 1 cảm biến nhiệt khác thì kịch bản trên không được gọi là tự mâu thuẫn.

Thế nào là kịch bản mâu thuẫn (conflict) ?

Kịch bản này được gọi là mâu thuẫn với kịch bản kia nếu như cả 2 kịch bản có tồn tại 2 hành động đơn giản (simple action) trái ngược nhau (counter-action) nhưng điều kiện xảy ra hành động trên lại giống nhau. Hai hành động được gọi là trái ngược nhau nếu hành vi của chúng ngược với nhau và chúng cùng là hành động đơn giản (simple action). Lấy ví dụ, ta có 2 kịch bản sau

- Kịch bản 1: Nếu đèn 1 bật thì đèn 2 tắt.
- Kịch bản 1: Nếu đèn 1 bật thì đèn 2 bật.

Ta nhận thấy, 2 hành động “đèn 2 bật” và “đèn 2 tắt” là 2 hành động trái ngược nhau. Hơn nữa, điều kiện xảy ra hành động trái ngược nhau trên là “đèn 1 bật”. Có thể kết luận rằng 2 kịch bản trên là mâu thuẫn với nhau.

Thế nào là kịch bản có khả năng mâu thuẫn (potential conflict)?

Hai kịch bản được gọi là có khả năng mâu thuẫn nếu như giữa 2 kịch bản ấy tồn tại ít nhất một mâu thuẫn (theo khái niệm từ mục 6.1.9) và mâu thuẫn đó chỉ xảy ra trong một số điều kiện nhất định. Sau đây là ví dụ về kịch bản có khả năng mâu thuẫn.

- Giả sử ta có kịch bản là “từ (12h40, 15h30) thì tắt đèn 1” và một kịch bản khác yêu cầu là bật đèn 1 nhưng có thời gian thực hiện không xác định (ví dụ: khi có gas, khi trời sáng, khi có người, khi nhiệt độ $>$, $<$, ...). Hai kịch bản trên vẫn có khả năng mâu thuẫn vì giữa chúng tồn tại sự mâu thuẫn giữa hành động “bật” và “tắt” đèn 1 và thời gian thực hiện kịch bản thứ hai là không xác định, nó có thể là thời điểm nào đó trong ngày và cùng lúc điều kiện kịch bản 1 đang thực thi thì điều kiện kịch bản 2 cũng được thỏa mãn.
- Ví dụ như ta có kịch bản 1 là “Nếu nhiệt độ lớn hơn 35 độ và nếu đèn 1 tắt thì đèn 2 bật” và kịch bản 2 là “Nếu đèn 1 tắt thì đèn 2 tắt”. Ta nhận thấy ở kịch bản 1 có cặp \langle điều kiện, hành động \rangle là \langle khi đèn 1 tắt, đèn 2 bật \rangle mâu thuẫn với \langle khi đèn 1 tắt, đèn 2 tắt \rangle ở kịch bản 2. Nhưng thực sự chỉ khi nhiệt độ lớn hơn 35 độ thì sự mâu thuẫn mới xảy ra, bình thường thì ta nói 2 kịch bản trên có khả năng mâu thuẫn mà thôi.
- Trường hợp điều kiện cùng khoảng giá trị cũng là một dạng có khả năng mâu thuẫn. Lấy 1 ví dụ như “nhiệt độ trong khoảng (30,40) độ C thì thực hiện bật đèn 1” và “nhiệt độ trong khoảng (35,45) độ C thì tắt đèn 1”. Ta thấy là 2 khoảng giá trị trên có trùng lặp lẫn nhau và 2 hành động lại đối nghịch nhau. Hai kịch bản trên có khả năng mâu thuẫn.

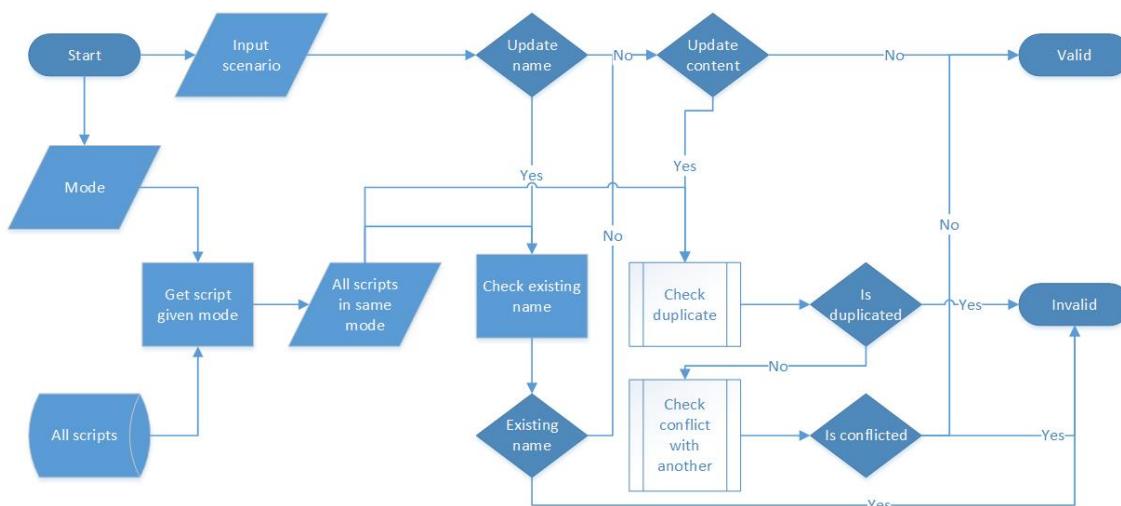
CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

- Hoặc trùng khoảng thời gian: tương tự như khoảng giá trị trên, còn bây giờ là khoảng thời gian. Ví dụ “từ (12h40, 15h30) thì bật đèn 1” và “từ (15h, 16h) tắt đèn 1” cũng là những kịch bản có khả năng mâu thuẫn.

Định nghĩa về kịch bản hợp lệ

Kịch bản hợp lệ là kịch bản không trùng tên hay trùng nội dung hoàn toàn với một trong những kịch bản, không tự mâu thuẫn với chính nó, cũng như không mâu thuẫn hay có khả năng mâu thuẫn với các kịch bản khác đã có ở cùng chế độ (mode) của ngôi nhà đang xét. Trong tương lai, nhóm đề xuất kịch bản có khả năng mâu thuẫn vẫn là kịch bản hợp lệ và chúng sẽ được quản lý bởi độ ưu tiên riêng biệt. Ở mục thảo luận, nhóm xin đề xuất phương pháp quản lý kịch bản có khả năng mâu thuẫn.

Thuật toán kiểm tra kịch bản hợp lệ



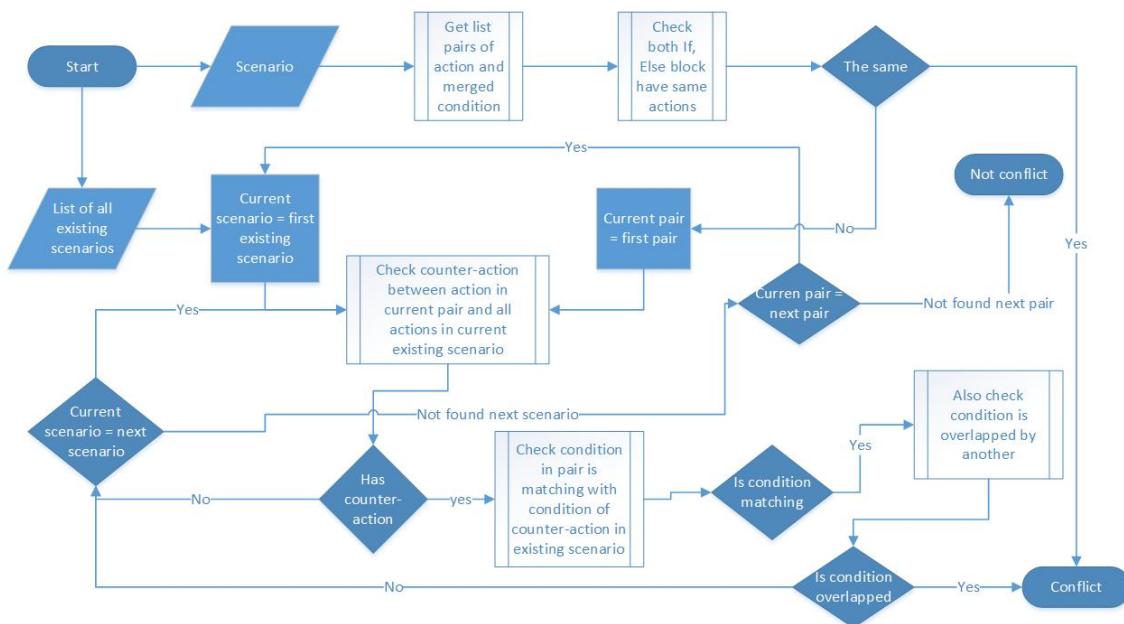
Hình 6.7: Flowchart thể hiện cách kiểm tra kịch bản hợp lệ

Đầu vào thuật toán nhận một kịch bản cần kiểm tra tính hợp lệ (gọi là “input scenario”), còn đầu ra của thuật toán sẽ là giá trị luận lý đúng hoặc sai, tương ứng kịch bản hợp lệ hoặc không hợp lệ. Hình 6.7 thể hiện thuật toán kiểm tra tính hợp lệ của kịch bản. Dựa trên thông tin về chế độ (mode) khi kịch bản được thêm vào hệ thống thì ta sẽ lấy tất cả kịch bản trong cùng chế độ ấy lên từ cơ sở dữ liệu. Mục đích của công việc này là kiểm tra xem tên của kịch bản mới (nếu có) có trùng với những kịch bản có sẵn hay không. Nếu trùng tên xảy ra thì kết luận rằng kịch bản đầu vào không hợp lệ. Kế tiếp, ta cần kiểm tra rằng nội dung kịch bản có bị trùng với kịch bản nào có sẵn chưa (xét trùng về mặt ngữ nghĩa chứ không chỉ là về mặt chuỗi lưu trữ kịch bản có giống nhau hoàn toàn hay không). Cuối cùng, thuật toán kiểm tra kịch bản mâu thuẫn sẽ được thực thi (chi tiết thuật toán

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

được mô tả ở mục sau). Nếu kịch bản đầu vào không xảy ra mâu thuẫn với các kịch bản sẵn có thì kết quả trả về từ thuật toán là hợp lệ, trường hợp khác sẽ là không hợp lệ.

Thuật toán kiểm tra kịch bản mâu thuẫn



Hình 6.8: Flowchart thể hiện cách kiểm tra kịch bản mâu thuẫn

Như đã đề cập về khái niệm kịch bản mâu thuẫn, trong thực tế nếu sự mâu thuẫn giữa các kịch bản tồn tại, hệ thống sẽ gặp nhiều rắc rối. Do đó, việc xử lý, ngăn chặn kịch bản mâu thuẫn là thiết yếu phải có của hệ thống. Hình 6.8 mô tả tổng quan về thuật toán kiểm tra kịch bản mâu thuẫn. Đầu vào của thuật toán là kịch bản hệ thống muốn kiểm tra, cùng với danh sách kịch bản đã tồn tại trong cùng chế độ.

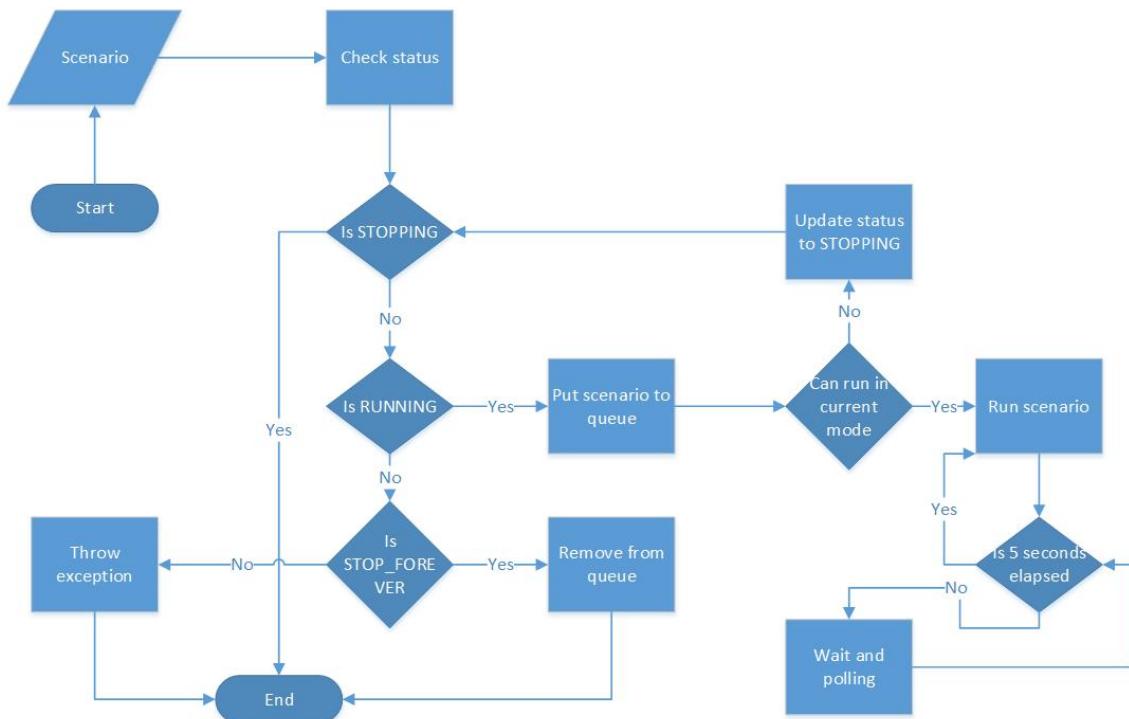
Bước đầu, ta cần trích xuất ra từ kịch bản đầu vào những cặp <điều kiện gộp, hành động đơn giản> (<merged condition, simple action>) giúp cho việc so sánh. Nếu như kịch bản có tất cả 3 hành động đơn giản (simple action) thì ta sẽ có ít nhất 3 cặp <điều kiện gộp, hành động đơn giản> như trên. Một lưu ý nhỏ là <điều kiện gộp> có thể không tồn tại. Một điều kiện bên ngoài có thể gộp được với điều kiện lồng trong nó khi và chỉ khi 2 điều kiện đó cùng thuộc về 1 thiết bị nào đó, hoặc là điều kiện về thời gian. Lấy ví dụ:

- Kịch bản 1: “Nếu nhiệt độ thu được từ cảm biến nhiệt gần cửa sổ lớn hơn 40 độ và bé hơn 50 độ thì bật đèn phòng”. Điều kiện gộp ở đây sẽ là nhiệt độ nằm trong khoảng 40 độ và 50 độ.
- Kịch bản 2: “Nếu thời gian từ 0h đến 6h thì tắt đèn phòng, đồng thời trong khoảng thời gian đó, nếu thời gian từ 4h đến 5h thì bật thêm đèn hành lang”. Điều kiện gộp ở đây cho hành động bật đèn hành lang sẽ là “từ 4h đến 5h”.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

Tiếp theo, thuật toán còn kiểm tra cả việc nếu như kịch bản có dạng “If/Then/Else” thì hành động ở khối “Then” và khối “Else” không được giống nhau (giống về mặt ngữ nghĩa). Nếu không, thuật toán sẽ xếp loại kịch bản này là mâu thuẫn. Sau các bước trên, ta sẽ phải lặp trên tất cả kịch bản hiện tại, với mỗi kịch bản, ta so sánh các hành động đơn giản trong kịch bản ấy với từng hành động trong cặp <điều kiện gộp, hành động đơn giản>. Nếu 2 điều kiện đó là trái ngược nhau về mặt ngữ nghĩa (ví dụ một hành động kêu bật đèn phòng và một hành động khác kêu tắt đèn phòng) thì ta xét tiếp về điều kiện gộp của chúng có “trùng nhau” hay không. Khái niệm xét “trùng” điều kiện đã được nêu rõ ở mục **6.1.9 Thế nào là kịch bản mâu thuẫn**. Nếu duyệt đến hết danh sách kịch bản hiện tại và không tìm thấy dấu hiệu kịch bản mâu thuẫn, thuật toán trả về kết quả kịch bản không mâu thuẫn. Trường hợp ngược lại thì sẽ trả về là kịch bản mâu thuẫn và lỗi sẽ được thông báo cho người dùng trên giao diện.

6.1.10 Module quản lý trạng thái các kịch bản (scenario runner)



Hình 6.9: Flowchart thể hiện cách quản lý trạng thái các kịch bản

Việc xây dựng kịch bản, cách thức tổ chức, cấu trúc dữ liệu kịch bản, ... đều là tiền đề cho việc thực thi kịch bản đó. Nhóm đã phải đắn đo trong việc chọn cách thiết kế cấu trúc dữ liệu để việc quản lý, cũng như thực thi kịch bản dễ dàng và suôn sẻ hơn. Dáp ứng cho nhu cầu quản lý ấy, nhóm tạo ra module Scenario Runner. Đầu vào của module này sẽ là một kịch bản hệ thống với mong muốn là hệ thống sẽ thực thi, “chạy” kịch bản ấy. Hình

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

6.9 mô tả thuật toán quản lý trạng thái các kịch bản trong hệ thống. Đầu vào thuật toán nhận kịch bản hệ thống với trạng thái xác định. Nếu trạng thái là dừng (STOP), module lập tức cập nhật ngay trạng thái đó vào kịch bản tương ứng trong hàng chờ hệ thống và làm nó dừng thật sự. Nếu trạng thái là chạy (RUNNING), ta cần kiểm tra thêm 1 điều kiện đó là kịch bản này có thể chạy ở chế độ hiện tại hay không. Nếu có thì cứ sau khoảng thời gian mặc định, tầm 5 giây, kịch bản được kích hoạt hàm chạy một lần. Và ngược lại, nếu kịch bản không thuộc về chế độ hiện tại của nhà này, trạng thái kịch bản sẽ cập nhật thành dừng và nằm trong hàng chờ hệ thống. Trường hợp mà người dùng mong muốn xóa đi kịch bản, trạng thái của nó sẽ trở thành dừng vĩnh viễn và được loại bỏ khỏi hàng chờ.

Module Scenario Runner quản lý mỗi kịch bản bởi 1 thread khác nhau, nói cách khác là chịu trách nhiệm xử lý multi-thread. Mỗi kịch bản cứ sau khoảng 5 giây sẽ được chạy lại một lần, nhằm đảm bảo mọi cập nhật mới nhất với kịch bản ấy có hiệu lực gần như lập tức. Kịch bản thì có chứa thông tin về trạng thái: chạy, dừng, dừng vĩnh viễn. Nhiệm vụ module này cũng là kiểm soát trạng thái các kịch bản ấy.

Sau khi một kịch bản mới được thêm vào hệ thống, trạng thái của nó sẽ là chạy (RUNNING). Khi kịch bản ấy được cập nhật, trạng thái của nó cũng được cập nhật lại. Nếu như người dùng muốn cho phép (enable) hoặc vô hiệu hóa (disable) ngôi nhà, hay thiết bị có liên quan kịch bản ấy thì thao tác đó cũng làm ảnh hưởng đến trạng thái của kịch bản. Ví dụ người dùng cho phép dùng (enable) thiết bị ấy thì các kịch bản liên quan thiết bị đó sẽ mang trạng thái đang chạy (RUNNING). Khi người dùng vô hiệu hóa thiết bị, các kịch bản liên quan sẽ chuyển sang trạng thái dừng (STOP). Tương tự với việc cho phép sử dụng nhà hay vô hiệu hóa nhà thì kịch bản thuộc nhà ấy cũng có trạng thái chạy hay dừng tương ứng.

Tuy nhiên, vấn đề quản lý multi-thread hiệu quả khi số lượng kịch bản tăng lên cũng là một khó khăn mà nhóm đang đối mặt và sẽ được đề cập tới trong mục thảo luận.

6.1.11 Module hỗ trợ bảo mật, xác thực và phân quyền (authorization and authentication)

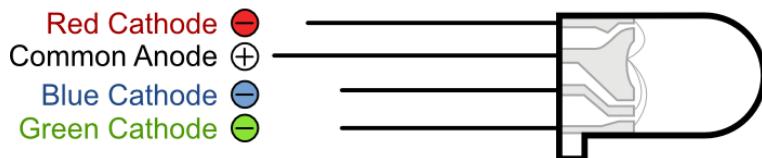
> Wating <

Kết thúc chương này, ta có được một cái nhìn chi tiết hơn, sâu hơn về phần thiết kế hệ thống back-end. Back-end đóng vai trò là trung tâm, là “bộ não” của toàn hệ thống, lưu trữ các dữ liệu quan trọng của người dùng,... cho nên khâu thiết kế, xây dựng và hiện thực cần tiến hành kĩ lưỡng, cân nhắc đến nhiều vấn đề liên quan. Hệ thống back-end được xây dựng theo hướng module hóa, mỗi module đảm nhận nhiệm vụ chuyên biệt. Trong đó, quan trọng nhất chính là các module chuyển đổi sang kịch bản hệ thống, module kiểm tra tính hợp lệ và module quản lý, khởi chạy các kịch bản. Tuy rằng hệ thống chưa thực sự hoàn thiện, việc quản lý tài nguyên, quản lý kịch bản chưa được tối ưu, dạng kịch bản chưa phong phú, đa dạng,... nhưng với bước khởi điểm này cùng những định hướng mở rộng trong tương lai thì hệ thống sẽ có tính khả thi và thực tiễn cao hơn.

6.2 Xây dựng module điều khiển thiết bị

6.2.1 Lắp đặt thiết bị

Để Raspberry Pi có thể điều khiển được các thiết bị thông qua các chân GPIO, đầu tiên, chúng ta phải lắp đặt đúng cách thiết bị đó vào Raspberry Pi. Nhằm mục đích đảm bảo an toàn và thuận tiện trong việc tiến hành thí nghiệm, chúng tôi ưu tiên sử dụng những thiết bị điện yêu cầu hiệu điện thế thấp (từ 5V trở xuống).



Hình 6.10: Thông tin các chân của đèn led [1]

Lắp đặt đèn: Đèn được sử dụng trong thí nghiệm là loại đèn led bốn chân tích cực mức cao. Một chân của đèn led sẽ luôn luôn được nối vào cổng Ground của Raspberry Pi, một chân còn lại (chân red, green hoặc blue) sẽ được nối vào một GPIO. Khi GPIO ở trạng thái High, tức mức điện áp đưa vào là 3.3V, đèn led sẽ sáng, ngược lại, đèn sẽ tắt. Hình 6.10 trình bày thông tin các chân của đèn led. Ngoài led ba màu, nhóm còn sử dụng thêm loại led chỉ có hai chân được sử dụng rộng rãi, cách lắp đặt tương tự như đèn led bốn chân.

Lắp đặt còi: Còi là một thiết bị không thể thiếu trong các căn nhà thông minh, mục đích chủ yếu là để cảnh báo chủ nhà đang có nguy hiểm xảy ra. Loại còi được nhóm sử dụng trong mô hình nhà thông minh gồm có ba chân: VCC, GND và Signal. Hình 6.11 mô tả vị trí các chân nằm trên module còi. Hai chân VCC và GND sẽ lần lượt được gắn vào hai cổng luôn luôn có hiệu điện thế lần lượt là 3.3V và 0V, chân Signal sẽ được gắn vào một cổng GPIO có nhiệm vụ điều khiển. Loại còi mà nhóm sử dụng thuộc loại tích cực mức thấp nên còi sẽ bắt đầu hú khi trạng thái của GPIO điều khiển là Low.



Hình 6.11: Module còi gồm 3 chân: VCC, GND và Signal (nằm giữa) [2]

Lắp đặt camera: Nhóm sử dụng camera chuyên dụng của Raspberry Pi để dùng cho việc chụp ảnh. Cách lắp đặt camera cũng khá đơn giản, trên Raspberry Pi có một cổng cắm camera nằm giữa cổng HDMI và cổng Ethernet. Hình 6.12 mô tả vị trí cổng cắm camera. Chỉ cần gắn dây camera vào đúng cổng trên Raspberry Pi là việc lắp đặt hoàn tất.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG



Hình 6.12: Raspberry Pi được lắp đặt camera chuyên dụng

Lắp đặt cảm biến khí gas: Với các đặc tính như giá thành rẻ, độ nhạy cao và phạm vi phát hiện rộng, cảm biến khí gas MQ-2 được nhóm sử dụng để phát hiện sự xuất hiện của các khí dễ cháy như gas, cồn và khói. Cảm biến gồm có bốn chân: VCC, GND, DOUT và AOUT, hình 6.13 mô tả cấu trúc bên ngoài của một cảm biến khí gas. Các chân VCC và GND lần lượt sẽ được nối vào các cổng có hiệu điện thế là 3.3V và 0V. Khi phát hiện khí gas, cổng AOUT sẽ xuất ra tín hiệu tương tự (analog) còn cổng DOUT sẽ xuất ra tín hiệu số (digital).



Hình 6.13: Module cảm biến khí gas MQ-2 [4]

Do đặc tính các chân GPIO của Raspberry Pi chỉ nhận được tín hiệu số, nên chúng tôi chỉ sử dụng chân DOUT của cảm biến để phát hiện khí gas. Chân DOUT sẽ được đấu nối với một cổng GPIO của Raspberry PI. Bình thường, chân DOUT của cảm biến sẽ ở trạng thái High, tức có mức điện áp là 3.3V, khi phát hiện khí gas, chân DOUT sẽ ngay lập tức chuyển từ trạng thái High thành Low (0V), và trạng thái này kéo dài cho đến khi cảm biến không còn phát hiện khí gas trong không khí. Độ nhạy của cảm biến sẽ được điều chỉnh bằng tay thông qua bộ điều chỉnh màu xanh đặc bên dưới cảm biến [3].

Lắp đặt cảm biến ánh sáng: Module cảm biến ánh sáng sử dụng quang trở được lựa chọn sử dụng để phát hiện thời gian hiện tại là ban ngày hay ban đêm. Khi cường độ ánh sáng tăng, điện trở của quang trở sẽ giảm và ngược lại, khi cường độ ánh sáng giảm, điện trở sẽ tăng. Giống module cảm biến khí gas MQ-2, module cảm biến ánh sáng sử dụng quang trở cũng có ba chân: VCC, GND và Signal. VCC và GND cũng sẽ được nối vào hai cổng có hiệu điện thế lần lượt là 3.3V và 0V, chân Signal được nối vào một cổng GPIO của Raspberry Pi.

Khi cường độ ánh sáng thấp hơn giá trị ngưỡng, cổng Signal sẽ ở trạng thái High, tức có

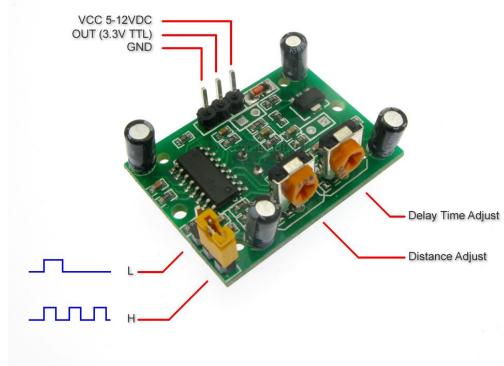
CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

hiệu điện thế là 3.3V. Khi cường độ ánh sáng vượt quá giá trị ngưỡng, cổng Signal sẽ ở trạng thái Low, tức có hiệu điện thế là 0V. Giá trị ngưỡng được điều chỉnh bằng tay sử dụng nút vặn màu xanh nằm phía dưới cảm biến. Hình 6.14 mô tả một con Raspberry Pi đã được kết nối với module cảm biến ánh sáng quang trở [5][6].



Hình 6.14: Module cảm biến ánh sáng quang trở được gán vào Raspberry Pi [6]

Lắp đặt cảm biến chuyển động: Chức năng chống trộm cũng là một chức năng không thể thiếu của một ngôi nhà thông minh. Nhóm sử dụng module cảm biến chuyển động PIR (Passive InfraRed) để nhận biết có người hay có trộm vừa đi ngang qua. Nguyên lý hoạt động của cảm biến là dựa vào bức xạ nhiệt mà con người phát ra. Mỗi người trong chúng ta đều có thân nhiệt lúc bình thường khoảng 37 độ C và đều phát ra các bức xạ nhiệt. Khi một người đi qua, các tia bức xạ nhiệt phát ra từ người đó sẽ kích hoạt cảm biến. Hình 6.15 mô tả cấu trúc bên ngoài cảm biến chuyển động PIR [7].



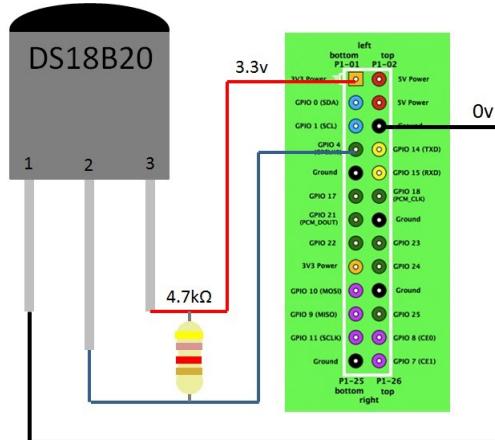
Hình 6.15: Cảm biến chuyển động PIR

Cảm biến có ba chân: VCC, GND và OUT. Cổng VCC và GND sẽ lần lượt nối vào các cổng có hiệu điện thế lần lượt là 5V và 0V trên Raspberry Pi. Cổng OUT sẽ được nối vào một

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

chân GPIO để ứng dụng trên Raspberry Pi có thể lấy dữ liệu từ cảm biến. Bình thường, khi không có người đi ngang qua, cổng OUT sẽ có trạng thái là Low, khi phát hiện người, trạng thái của cổng OUT là High. Khoảng cách phát hiện người và độ trễ của cảm biến có thể được điều chỉnh bằng tay qua hai bộ điều chỉnh màu cam đặt ở một cạnh của cảm biến.

Lắp đặt cảm biến nhiệt độ: Nhóm sử dụng cảm biến nhiệt độ DS18B20 để lấy giữ liệu về nhiệt độ trong hệ thống nhà thông minh. Cách lắp đặt cảm biến nhiệt độ này có chút phức tạp hơn so với các cảm biến khác. Cảm biến cũng gồm có ba chân: VCC, GND và OUT. Giống như các cảm biến khác, cổng VCC và GND cũng được lần lượt gắn vào các cổng có hiệu điện thế lần lượt là 3.3V và 0V. Tuy nhiên, khác với các cảm biến khác, cổng OUT của cảm biến nhiệt độ DS18B20 phải được gắn vào GPIO số 4 nằm ở vị trí số 7 trên Raspberry Pi. Bên cạnh đó, phải có một điện trở 4.7 ôm nối giữa VCC và OUT. Đầu màu vàng của cảm biến điện trở sẽ được nối vào dây VCC của cảm biến nhiệt độ. Hình 6.16 mô tả cách lắp ghép cảm biến nhiệt độ DS18B20 vào Raspberry Pi.



Hình 6.16: Cách mắc dây cho cảm biến nhiệt độ DS18B20 [11]

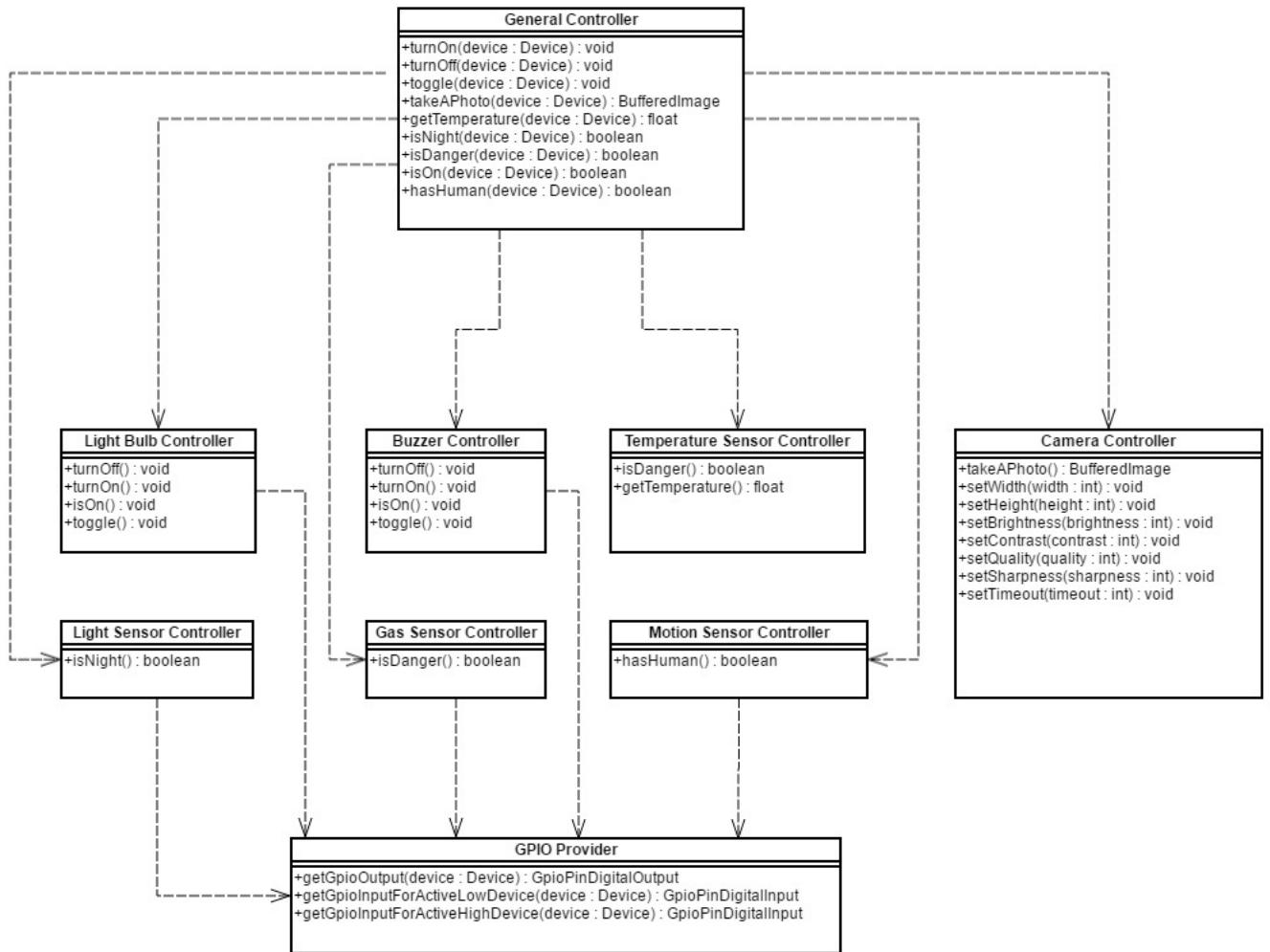
Do cảm biến DS18B20 sử dụng cơ chế truyền tín hiệu 1-Wire nên cho phép người sử dụng có thể kết nối nhiều cảm biến vào cùng một chân trên Raspberry Pi. Cách kết nối cũng vô cùng đơn giản, chỉ cần kết nối tất cả cổng OUT của tất cả cảm biến vào cùng chân GPIO số 4 của Raspberry Pi [9].

6.2.2 Hiện thực module

Module điều khiển thiết bị được chia làm nhiều bộ điều khiển nhỏ cho các loại thiết bị và được quản lý bởi bộ điều chung. Tất cả các yêu cầu từ server sẽ được bộ điều khiển chung xử lý và đẩy vào các bộ điều khiển thích hợp. Bên cạnh đó, module điều khiển thiết bị còn có thêm bộ cung cấp GPIO với nhiệm vụ khởi tạo đúng loại cổng mà một bộ điều khiển cần. Hình 6.17 thể hiện sơ đồ lớp (class diagram) của module.

Bộ điều khiển chung (GPIO Controller): Như đã trình bày, bộ điều khiển chung sẽ nhận tất cả yêu cầu từ server. Bộ điều khiển chung cung cấp tất cả các phương thức mà

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG



Hình 6.17: Sơ đồ lớp (class diagram) của module điều khiển thiết bị

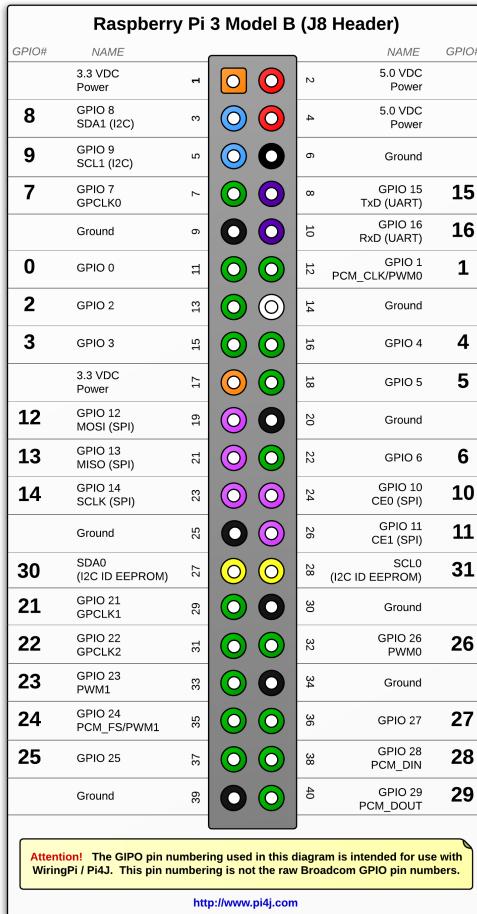
server cần để lấy dữ liệu từ cảm biến cũng như điều khiển các thiết bị điện. Để sử dụng một hàm, server cần truyền vào một thực thể device. Thực thể device sẽ cung cấp tất cả các thông tin mà bộ điều khiển chung cần để thực hiện đúng hành động vào đúng thiết bị mà server cần như loại thiết bị, số thứ tự cổng GPIO,... Bộ điều khiển chung sẽ dựa vào thông tin loại thiết bị và quyết định nên sử dụng bộ điều khiển thiết bị nào phù hợp. Ví dụ, server gửi yêu cầu tắt một thiết bị, trong yêu cầu của server sẽ chứa thông tin loại thiết bị mà server muốn tắt (đèn, còi,...) kèm theo số thứ tự của cổng GPIO đang điều khiển thiết bị đó. Nếu loại thiết bị là đèn, bộ điều khiển chung sử dụng bộ điều khiển đèn (Light bulb controller) để thực hiện phương thức tắt.

Bộ cung cấp GPIO (GPIO Provider): Trong thư viện Pi4J, để sử dụng một cổng GPIO, bước đầu tiên là khai báo cổng cần sử dụng. Bộ cung cấp GPIO có nhiệm vụ hỗ

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

trợ các bộ điều khiển thiết bị thực hiện thao tác khai báo một cổng GPIO. Tùy thuộc vào mục đích mà một cổng GPIO có thể được khai báo là cổng nhập (input pin) hay cổng xuất (output pin). Cổng nhập (input pin) được các bộ điều khiển cảm biến sử dụng để lấy dữ liệu từ các cổng GPIO. Cổng xuất (output pin) được bộ điều khiển đèn và còi sử dụng để điều khiển việc bật hoặc tắt. Cú pháp khai báo một cổng GPIO được trình bày ở đoạn code sau:

```
1 final GpioPinDigitalOutput outputPin = gpio.provisionDigitalOutputPin(  
2     RaspiPin.GPIO_01, "MyLED", PinState.HIGH);
```



Hình 6.18: Quy tắc đánh số của Pi4J cho Raspberry Pi 3 Model B

Để tránh những sự thay đổi về phần cứng sẽ ảnh hưởng đến phần mềm sử dụng các chân GPIO của Raspberry Pi, Pi4J sử dụng bộ quy tắc đánh số riêng cho những cổng GPIO. Vì thế, bộ cung cấp GPIO còn giữ vai trò chuyển đổi số thứ tự một cổng GPIO trên Raspberry Pi về số thứ tự hợp lệ trên Pi4J. Hình 6.18 mô tả cách đánh số của thư viện Pi4J trên Raspberry Pi 3 Model B. Ví dụ, cổng GPIO ở vị trí 11 trên Raspberry Pi khi chuyển đổi

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

sang cách đánh số của thư viện Pi4J sẽ có số thứ tự là GPIO 0.

Bộ điều khiển đèn và bộ điều khiển còi: Hai bộ điều khiển này được sử dụng bởi bộ điều khiển chung khi server có các yêu cầu liên quan đến đèn và còi. Sau khi nhận yêu cầu từ server, bộ điều khiển chung sẽ truyền thông tin về thiết bị đến đúng bộ điều khiển thích hợp. Khi nhận thông tin về thiết bị bao gồm số thứ tự cổng GPIO, bộ điều khiển đèn hoặc còi sẽ sử dụng bộ cung cấp GPIO để khởi tạo đúng loại cổng, input pin hay output pin.

Như đã trình bày, mỗi cổng GPIO sẽ có hai trạng thái là Low hoặc High. Bộ điều khiển đèn và còi sẽ điều khiển các thiết bị bằng cách thay đổi trạng thái của chân GPIO đang điều khiển thiết bị đó. Tùy loại thiết bị mà trạng thái High hoặc Low sẽ tương ứng với việc bật hay tắt và ngược lại. Ví dụ, nếu loại đèn được sử dụng là tích cực mức thấp, khi chân GPIO ở trạng thái Low thì đèn sẽ sáng và ngược lại. Loại đèn và còi mà nhóm đang sử dụng để thuộc loại tích cực mức thấp. Cú pháp thay đổi trạng thái một cổng GPIO khi sử dụng thư viện Pi4J thể hiện như sau:

```
1   outputPin.setState(PinState.LOW);  
2   outputPin.setState(PinState.HIGH);
```

Các bộ điều khiển cảm biến: Tương tự như bộ điều khiển đèn và còi, các bộ điều khiển cảm biến cũng được sử dụng bởi bộ điều khiển chung khi server gửi một yêu cầu lấy dữ liệu từ các cảm biến đến bộ điều khiển chung. Tùy loại cảm biến mà cách lấy dữ liệu sẽ khác nhau, tuy nhiên, giống như đèn và còi, mỗi cảm biến sẽ được điều khiển bởi một GPIO nhất định.

Đối với bộ điều khiển cảm biến khí gas, do loại cảm biến khí gas mà nhóm sử dụng thuộc loại tích cực mức thấp, tức bình thường, chân DOUT của cảm biến sẽ ở trạng thái High, khi phát hiện khí gas, lập tức chân DOUT sẽ chuyển sang trạng thái Low. Vì vậy, sau khi nhận thông tin về số thứ tự GPIO từ bộ điều khiển chung, bộ điều khiển khí gas cần nhờ sự hỗ trợ của bộ cung cấp GPIO để khai báo một cổng nhập phù hợp với loại cảm biến tích cực mức thấp. Sau khi đã khai báo xong, bộ điều khiển khí gas có thể lấy thông tin từ cảm biến bằng cách kiểm trang xem GPIO điều khiển cảm biến đang ở trạng thái High hay Low, vì GPIO điều khiển cảm biến sẽ được nối vào chân DOUT của cảm biến khí gas. Cú pháp kiểm tra khi sử dụng thư viện Pi4J cũng khá đơn giản, được thể hiện ở đoạn code sau:

```
1   activeLowPin.isLow();
```

Khác với bộ điều khiển cảm biến khí gas, loại cảm biến ánh sáng và cảm biến chuyển động mà nhóm đang sử dụng thuộc loại tích cực mức cao, tức khi trời tối hoặc khi phát hiện có người chuyển động, cổng Signal của cảm biến ánh sáng và cổng OUT của cảm biến chuyển động sẽ ở trạng thái High. Vì vậy, cách khai báo cổng nhập cho hai bộ điều khiển này khác với bộ điều khiển cảm biến khí gas, cần sử dụng phương thức khai báo cổng nhập tích cực mức cao do bộ cung cấp GPIO hỗ trợ. Sau khi đã hoàn thành việc khai báo, công việc lấy dữ liệu từ cảm biến sẽ vô cùng đơn giản, chỉ cần kiểm tra xem cổng GPIO đang điều khiển cảm biến ánh sáng hoặc cảm biến chuyển động có ở trạng thái High không, nếu có

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

tức trời đang tối hoặc có người đang đi ngang qua. Cú pháp kiểm tra được thực hiện bởi đoạn code sau với sự hỗ trợ của thư viện Pi4J:

```
1 activeHighPin . isHigh () ;
```

Trong tất cả các thiết bị cảm biến mà nhóm hiện có, cảm biến nhiệt độ là một trường hợp đặc biệt. Do cảm biến nhiệt độ mà nhóm sử dụng là DS18B20, sử dụng cơ chế truyền tín hiệu 1wire, có thể kết nối nhiều cảm biến vào chân GPIO số 4 của Raspberry Pi nên cách lấy dữ liệu sẽ khác với các cảm biến khác. Thông tin về nhiệt độ sẽ được để vào một tệp có tên là w1_slave. Mỗi cảm biến nhiệt độ DS18B20 sẽ có một thư mục riêng chứa tệp w1_slave, vì vậy, chúng ta cần vào đúng thư mục của cảm biến phù hợp. Cấu trúc thư mục của mỗi cảm biến chứa thông tin nhiệt độ là:

```
/sys/bus/w1/devices/<temperature-sensor-id>/w1_slave
```

Trong đó, <temperature-sensor-id> là thư mục riêng của mỗi cảm biến, và <temperature-sensor-id> sẽ khác nhau với từng cảm biến. Ví dụ, cảm biến nhiệt độ mà nhóm đang sử dụng có <temperature-sensor-id> là 28-0316054b32ff. Thông tin về nhiệt độ mà cảm biến nhận được sẽ được liên tục cập nhật vào tệp w1_slave. Nội dung toàn bộ của tệp w1_slave là:

```
1 d9 01 4b 46 7f ff 0c 10 13 : crc=13 YES
2 d9 01 4b 46 7f ff 0c 10 13 t=29562
```

Trong đó, t=29562 chính là giá trị mà cảm biến đo được. Theo ví dụ trên, nhiệt độ hiện tại là 29.562 độ C và đây cũng chính là giá trị mà bộ điều khiển cảm biến nhiệt độ trả về cho server.

Bộ điều khiển camera: Raspberry Pi cung cấp một chân cắm đặc biệt để kết nối camera chuyên dụng vào Raspberry Pi. Vì vậy, khác với các bộ điều khiển khác sử dụng GPIO để lấy dữ liệu từ cảm biến hoặc điều khiển hoạt động bật tắt, bộ điều khiển camera điều khiển hoạt động chụp hình thông qua chân cắm được dành riêng cho camera. Do thư viện Pi4J không hỗ trợ điều khiển camera, nên bộ điều khiển camera sử dụng một thư viện khác là JRPiCam. Để sử dụng JRPiCam, đầu tiên chúng ta cần khởi tạo đối tượng RPiCamera. Cú pháp khởi tạo được thể hiện trong đoạn code sau:

```
1 RPiCamera piCamera = new RPiCamera () ;
```

Bộ điều khiển camera cung cấp một chức năng chính là chụp hình khi cần thiết. Khi server muốn chụp một tấm ảnh từ camera, server sẽ gửi một yêu cầu đến một điều khiển chung và bộ điều khiển chung sẽ tiếp tục gửi yêu cầu đến bộ điều khiển thiết bị. Khi nhận được yêu cầu, bộ điều khiển camera sẽ tiến hành chụp ảnh với sự trợ giúp của thư viện JRPiCam. Cú pháp sử dụng thư viện để chụp ảnh được mô tả qua đoạn code sau:

```
1 //Take image and store in BufferedImage
2 BufferedImage bufferedImage = piCamera . takeBufferedStill () ;
```

Ngoài chụp ảnh, bộ điều khiển camera còn cung cấp các phương thức hỗ trợ trong việc cấu hình camera. Các thông số của camera đều có thể được tùy chỉnh như kích thước ảnh,

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

độ sáng, độ tương phản, độ nét, độ trẽ,... Cách tùy chỉnh thông số khi sử dụng JRPiCam được thể hiện qua đoạn code sau:

```
1 // Change Automatic White Balance setting to automatic
2 piCamera.setAWB(AWB.AUTO);
3 // Turn off Dynamic Range Compression
4 piCamera.setDRC(DRC.OFF);
5 // Set maximum contrast
6 piCamera.setContrast(contrast);
7 // Set maximum sharpness
8 piCamera.setSharpness(sharpness);
9 // Set brightness
10 piCamera.setBrightness(brightness);
11 // Set maximum quality
12 piCamera.setQuality(quality);
13 // Wait 1 second to take the image
14 piCamera.setTimeout(timeout);
15 // Set image width
16 piCamera.setWidth(width);
17 // Set image height
18 piCamera.setHeight(height);
19 // Turn on image preview
20 piCamera.turnOnPreview();
21 // Change encoding of images to PNG
22 piCamera.setEncoding(Encoding.PNG);
```

6.2.3 Đánh giá

Đối với cách thiết kế và hiện thực như trên, module điều khiển thiết bị đã cung cấp đầy đủ các phương thức cần thiết giúp server có thể điều khiển các thiết bị như đèn, còi, camera và lấy dữ liệu từ các cảm biến. Bên cạnh đó, module đã được hiện thực đúng với cách thiết kế ban đầu là điều khiển thiết bị thông qua các chân GPIO của Raspberry Pi, cung cấp cho server cách thức đơn giản để điều khiển thiết bị mà không cần quan tâm đến thiết bị thuộc loại gì.

Tuy nhiên, việc sử dụng các chân GPIO của Raspberry Pi để điều khiển và nhận dữ liệu từ các thiết bị vẫn còn nhiều hạn chế. Do GPIO chỉ có hai trạng thái là Low và High nên việc điều khiển đèn và còi chỉ dừng lại ở mức bật hoặc tắt, vẫn chưa có cách để có thể cấu hình độ sáng cho đèn hoặc âm lượng cho còi thông qua các chân GPIO. Cũng vì lý do trên mà độ nhạy của các cảm biến như cảm biến ánh sáng hoặc cảm biến khí gas phải được điều chỉnh bằng tay bằng bộ điều chỉnh có trên cảm biến. Số lượng giới hạn các chân GPIO cũng là một khó khăn nếu một căn nhà có nhiều thiết bị.

6.3 Phát triển ứng dụng di động

6.3.1 Hướng phát triển ứng dụng di động

Để quá trình phát triển ứng dụng được hiệu quả, trước tiên chúng tôi cần quyết định hướng phát triển ứng dụng di động. Hiện nay có 2 xu hướng phát triển ứng dụng di động chính là: phát triển ứng dụng di động thuần túy (Native Mobile Application Development) và phát triển ứng dụng di động lai (Hybrid Mobile Application Development).

Phát triển ứng dụng di động thuần túy (Native Mobile Application Development): phát triển ứng dụng di động thuần túy là phát triển một ứng dụng đặc biệt chỉ chạy trên một hệ điều hành nhất định thuộc một thiết bị nhất định, với ngôn ngữ phát triển cụ thể (như Objective-C hoặc Swift cho hệ điều hành iOS hoặc Java cho hệ điều hành Android), và thường phải thông qua điều chỉnh để có thể chạy được trên nhiều thiết bị khác nhau [6].

Vì ứng dụng được phát triển hoàn toàn trong môi trường dành riêng cho một hệ điều hành nhất định, với những đặc tính kỹ thuật và giao diện đặc trưng của hệ điều hành đó, nên không chỉ có lợi thế về hiệu xuất, ứng dụng di động thuần túy còn có lợi thế trong trải nghiệm người dùng. Lợi thế trong trải nghiệm người dùng ở đây là ứng dụng có được sự đồng nhất về mặt giao diện và cảm nhận với nhiều ứng dụng thuần túy khác trên thiết bị. Người dùng có thể dễ dàng nắm bắt cách thức sử dụng cũng như tương tác với ứng dụng một cách nhanh chóng hơn. Ngoài ra, những ứng dụng di động thuần túy có lợi thế không nhỏ trong việc có khả năng truy cập và sử dụng một cách dễ dàng những tính năng đặc thù của thiết bị (GPS, sổ địa chỉ, camera, bộ phận cảm ứng,...) [1].

Phát triển ứng dụng di động lai (Hybrid Mobile Application Development): phát triển ứng dụng di động lai là phát triển ứng dụng dựa trên nền tảng Web, sử dụng công nghệ phổ biến là HTML5 và JavaScript, được đóng gói lại trong một thành phần thuần túy (native container). Thành phần này thực hiện việc tải phần lớn thông tin lên giao diện khi người dùng truy cập qua từng chức năng của ứng dụng [1].

Dựa trên thông tin so sánh giữa 2 hướng phát triển ứng dụng di động chính ở Bảng 6.1, để có thể hỗ trợ được nhiều nền tảng di động một cách dễ dàng, tiết kiệm thời gian trong việc mở rộng sang các nền tảng khác nhau cũng như nâng cấp cải thiện ứng dụng, đồng thời vì kiến trúc ứng dụng yêu cầu không quá phức tạp và không đòi hỏi nhiều tới những tính năng phần cứng đặc thù của thiết bị, chúng tôi quyết định sẽ phát triển ứng dụng di động theo hướng phát triển ứng dụng di động lai.

6.3.2 Các công nghệ sử dụng

Theo hướng phát triển ứng dụng di động lai, ứng dụng cần được xây dựng trên nền tảng Web. Giao diện Web của ứng dụng sẽ được thiết kế tập trung vào việc hỗ trợ cho thiết

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

Bảng 6.1: So sánh điểm mạnh, điểm yếu của Phát triển ứng dụng di động thuần túy và Phát triển ứng dụng di động lai [4]

So sánh	Phát triển ứng dụng di động thuần túy	Phát triển ứng dụng di động lai
Điểm mạnh	<ul style="list-style-type: none"> - Về mặt hiệu năng, trong hầu hết trường hợp, ứng dụng di động thuần túy chạy nhanh hơn ứng dụng lai nhưng sự chênh lệch không quá lớn và thường khó nhận biết bởi người dùng. - Về tính năng, ứng dụng di động thuần túy có thể dễ dàng truy cập tới phần cứng của thiết bị (camera, thiết bị thu âm...) cũng như truy cập tới các chức năng đặc quyền như sao chép, tạo, ghi dữ liệu trên bộ nhớ, thông tin danh bạ, cuộc gọi, tin nhắn... so với ứng dụng di động lai còn nhiều hạn chế về việc tận dụng toàn bộ sức mạnh của thiết bị. - Khi không có kết nối Internet, ứng dụng di động thuần túy có thể sử dụng những dữ liệu đã lưu trữ tạm (cache) trước đó trong khi ứng dụng lai đa phần đều cần phải có kết nối Internet để truy cập dữ liệu. 	<ul style="list-style-type: none"> - Người phát triển không bị hạn chế vào một hệ điều hành nhất định, có thể phát triển chỉ một ứng dụng nhưng chạy được trên nhiều hệ điều hành khác nhau. - Chi phí cho việc phát triển, bảo trì và nâng cấp ứng dụng sẽ được giảm thiểu đáng kể vì chỉ có một phiên bản duy nhất. - Ngôn ngữ lập trình cho ứng dụng di động lai là HTML và JavaScript rất phổ dụng, đa số người phát triển đều biết.
Điểm yếu	<ul style="list-style-type: none"> - Ứng dụng di động thuần túy không thể chạy trên nhiều hệ điều hành khác nhau, nói cách khác, một ứng dụng di động thuần túy chỉ chạy được trên một hệ điều hành nhất định. - Khi muốn phát triển đa nền tảng, phát triển ứng dụng di động thuần túy sẽ có chi phí phát triển cao vì đòi hỏi khả năng thành thạo nhiều ngôn ngữ của người phát triển. - Sự đa dạng các phiên bản hệ điều hành gây cản trở tính tương thích của các ứng dụng di động thuần túy. 	<ul style="list-style-type: none"> - Về mặt hiệu năng, đa phần các ứng dụng di động lai không chạy nhanh bằng các ứng dụng di động thuần túy. - Ứng dụng di động lai thường không tận dụng được tối đa các tính năng phần cứng, sức mạnh của thiết bị. - Một số chợ ứng dụng sẽ không chấp nhận ứng dụng lai được đăng lên nếu như không hoạt động đủ trơn tru.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

bị di động. Sau đó, ứng dụng sẽ được chuyển tiếp từ nền tảng Web sang các nền tảng di động thông qua các công nghệ hỗ trợ.

Các công nghệ chính được sử dụng để phát triển ứng dụng:

- AngularJS – Công nghệ được sử dụng để phát triển ứng dụng trên nền tảng Web
- Apache Cordova – Công nghệ được sử dụng để chuyển tiếp ứng dụng từ nền tảng Web sang các nền tảng di động khác nhau

AngularJS: là một framework mạnh mẽ mã nguồn mở có cấu trúc hỗ trợ phát triển ứng dụng Web động. Framework này cho phép sử dụng HTML như là một ngôn ngữ mẫu, đồng thời cho phép mở rộng các cú pháp HTML để diễn đạt các thành phần của ứng dụng một cách rõ ràng và ngắn gọn. Hai tính năng chính của AngularJS là Liên kết dữ liệu (Data Binding) và Tiêm nhiễm phụ thuộc (Dependency Injection) giúp loại bỏ phần lớn mã code mà người phát triển ứng dụng thường phải viết.

Các đặc tính của AngularJS:

- AngularJS hỗ trợ người phát triển viết ứng dụng theo mô hình MVC (Model View Controller).
- Các ứng dụng AngularJS có khả năng tương thích với hầu hết các trình duyệt web với nhiều phiên bản trên các nền tảng khác nhau.
- AngularJS là framework mã nguồn mở, hoàn toàn miễn phí và được sử dụng rộng rãi bởi hàng ngàn lập trình viên trên thế giới.

Các tính năng và thành phần cốt lõi của AngularJS:

- **Khung nhìn (View):** là những gì mà người dùng nhìn thấy được.
- **Mô hình (Model):** dữ liệu nằm trên View mà có thể tương tác.
- **Liên kết dữ liệu (Data Binding):** đồng bộ dữ liệu giữa 2 thành phần model và view.
- **Chỉ thị (Directive):** mở rộng các thẻ HTML với các đặc tính và yếu tố tự tạo.
- **Bộ quản lý (Controller):** xử lý các thao tác nghiệp vụ bên dưới các Khung nhìn.
- **Biểu thức (Expression):** truy cập các biến và hàm từ Phạm vi.
- **Phạm vi (Scope):** phạm vi nơi các Mô hình được lưu trữ để các Bộ quản lý, Chỉ thị và Biểu thức có thể truy cập.
- **Dịch vụ (Service):** những thao tác nghiệp vụ có thể sử dụng lại độc lập với các Khung nhìn.
- **Tiêm nhiễm phụ thuộc (Dependency Injection):** tạo và liên kết các đối tượng và hàm.

Với các tính năng hỗ trợ phù hợp với kiến trúc của ứng dụng, cộng với sự phổ biến và được sử dụng rộng rãi của AngularJS. Nhóm chúng tôi quyết định sử dụng công nghệ này để

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

phát triển ứng dụng trên nền tảng Web.

Apache Cordova: là một framework mã nguồn mở hỗ trợ phát triển ứng dụng di động trên nền tảng Web. Framework này cho phép sử dụng những kĩ thuật Web chuẩn như HTML5, CSS3 và JavaScript để phát triển. Ứng dụng Web được xây dựng sẽ được chuyển tiếp thành ứng dụng trên các nền tảng di động gốc và dựa trên các API ràng buộc chuẩn để truy cập tới các chức năng của từng thiết bị như cảm biến, dữ liệu, tình trạng mạng,...

Apache Cordova thường được sử dụng khi:

- Người phát triển ứng dụng di động muốn mở rộng ứng dụng từ một nền tảng sang nhiều nền tảng khác, mà không cần phải phát triển lại toàn bộ ứng dụng theo từng ngôn ngữ lập trình và công cụ riêng của từng nền tảng.
- Người phát triển ứng dụng Web muốn đưa ứng dụng lên nhiều nền tảng cũng như nhiều cửa hàng ứng dụng khác nhau.
- Người phát triển ứng dụng di động muốn pha trộn nhiều thành phần thuộc nền tảng di động gốc và các thành phần này có thể truy cập tới các API cấp thiết bị.

	android	blackberry10	ios	Ubuntu	wp8 (Windows Phone 8)	windows (8.1, 10, Phone 8.1)	OS X
cordova CLI	✓ Mac, Windows, Linux	✓ Mac, Windows, Linux	✓ Mac	✓ Ubuntu	✓ Windows	✓	✓ Mac
Embedded WebView	✓	X	✓	✓	X	X	✓
Plugin Interface	✓	✓	✓	✓	✓	✓	✓

Hình 6.19: Bảng các nền tảng Apache Cordova hỗ trợ

Với tính đơn giản dễ sử dụng, đồng thời có khả năng hỗ trợ nhiều nền tảng di động khác nhau được thể hiện ở 6.19, đặc biệt là 3 nền tảng di động chính được sử dụng rộng rãi hiện nay là Android, iOS và Windows Phone, chúng tôi quyết định sử dụng công nghệ Apache Cordova để chuyển tiếp ứng dụng từ nền tảng Web sang các nền tảng di động khác nhau.

6.3.3 Hiện thực ứng dụng di động

Theo kiến trúc ứng dụng đã được trình bày ở Chương 5, ứng dụng di động sẽ bao gồm 6 trang giao diện chính (main pages) và 8 thành phần chia sẻ (shared components) là các thành phần chung được sử dụng lại nhiều lần giữa các trang giao diện khác nhau hoặc trong cùng một trang giao diện.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

Các thành phần chia sẻ (reusable components):

Các thành phần chia sẻ trong ứng dụng bao gồm 8 thành phần:

- Thanh công cụ (Navbar).
- Thẻ nhà (Home-panel).
- Thẻ kiểu thiết bị (Device-type-panel).
- Thẻ thiết bị (Device-panel).
- Thẻ kịch bản (Device-script-panel).
- Thẻ kịch bản khi/thì (Device-script-when-then).
- Thẻ kịch bản từ/đến (Device-script-from-to).
- Thẻ kịch bản tự tạo (Device-script-custom).

Các thành phần này thực chất là các Chỉ thị (Directive) của AngularJS, với mỗi thành phần có một giao diện với một bộ quản lý (controller) đảm nhận các xử lý nghiệp vụ riêng. Vì bản chất là Chỉ thị (Directive), mỗi thành phần đều có thể dễ dàng gắn vào nhau hoặc vào các thành phần chính như là một dạng thẻ HTML bình thường.

Thanh công cụ (Navbar)

Thanh công cụ (Navbar) ở hình 6.20 là thành phần chia sẻ cho phép người dùng thực hiện một số các thao tác cơ bản của ứng dụng một cách nhanh chóng, được sử dụng ở các trang giao diện chính là: danh sách các ngôi nhà, danh sách các kiểu thiết bị và danh sách các thiết bị. Thanh công cụ có nhiều chức năng khác nhau tùy thuộc vào trang giao diện chính mà nó được sử dụng. Các chức năng của thanh công cụ:

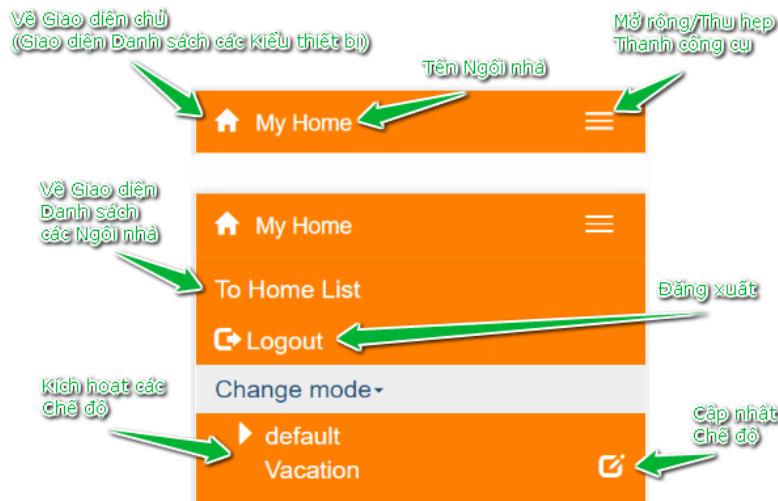
- Thẻ hiện tên của ngôi nhà hiện tại đang truy cập.
- Cho phép người dùng quay lại trang giao diện danh sách các Kiểu thiết bị.
- Cho phép người dùng đăng xuất.
- Cho phép người dùng kích hoạt các chế độ.
- Cho phép người dùng xem, cập nhật chỉnh sửa thông tin hoặc xóa các chế độ như ở hình 6.21.

Thẻ nhà (Home-panel)

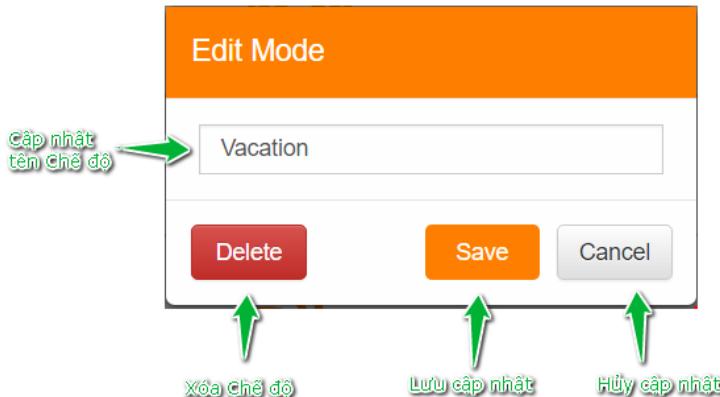
Thẻ nhà (Home-panel) ở hình 6.22 là thành phần chia sẻ thể hiện thông tin của một ngôi nhà được lắp đặt hệ thống, thành phần này được sử dụng nhiều lần như một danh sách trong trang giao diện chính danh sách các ngôi nhà. Các chức năng của thẻ nhà:

- Thẻ hiện tên của ngôi nhà.
- Cho phép người dùng ngưng hoạt động tạm thời (Disable) hoặc tái hoạt động (Enable) ngôi nhà.
- Cho phép người dùng xem, cập nhật chỉnh sửa thông tin hoặc xóa ngôi nhà như ở hình 6.23.
- Chuyển tiếp người dùng tới thành phần chính Danh sách các Kiểu thiết bị.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG



Hình 6.20: Thanh công cụ (Navbar)



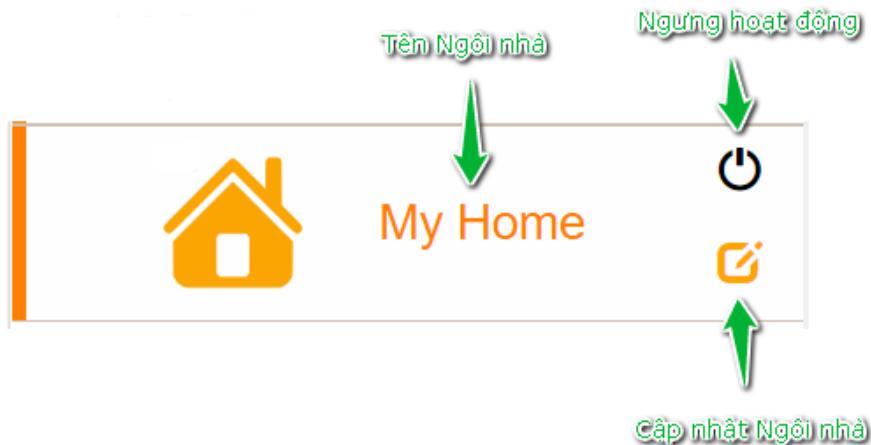
Hình 6.21: Bảng hộp thoại cập nhật chế độ

Thẻ Kiểu thiết bị (Device-type-panel)

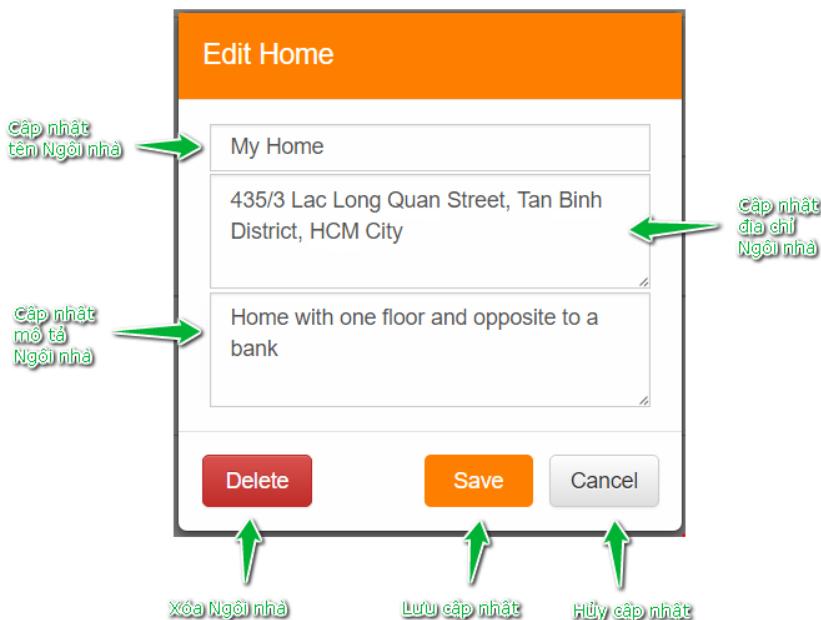
Thẻ kiểu thiết bị (Device-type-panel) ở hình 6.24 là thành phần đóng góp thê hiện thông tin của một kiểu thiết bị được hệ thống hỗ trợ, thành phần này được sử dụng nhiều lần như một danh sách trong thành phần chính Danh sách Các kiểu thiết bị. Ứng với mỗi kiểu thiết bị thì thẻ kiểu thiết bị có kí hiệu thê hiện và tên khác nhau, có 6 kiểu thiết bị hỗ trợ: Cảm biến nhiệt độ (Temperature Sensor), cảm biến chuyển động (Motion Sensor), cảm biến ánh sánh (Light Sensor), cảm biến khí gas (Gas Sensor), bóng đèn (Light) và còi hú (Buzzer). Các chức năng của thẻ kiểu thiết bị:

- Thê hiện thông tin của kiểu thiết bị.
- Chuyển tiếp người dùng tới trang giao diện chính danh sách các thiết bị.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG



Hình 6.22: Thẻ nhà

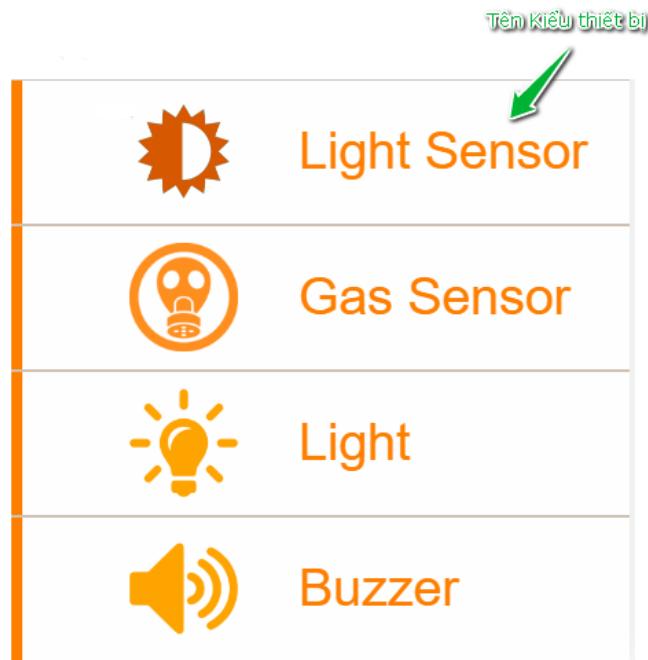


Hình 6.23: Bảng hộp thoại cập nhật Ngôi nhà

Thẻ Thiết bị (Device-panel)

Thẻ thiết bị (Device-panel) ở hình 6.25 là thành phần chia sẻ thẻ hiển thị thông tin của một thiết bị cùng với các kịch bản định sẵn của thiết bị đó, được sử dụng nhiều lần như một danh sách trong trang giao diện danh sách các thiết bị. Thành phần này chứa bên trong nó một tập hợp các thành phần chia sẻ thẻ kịch bản để hỗ trợ việc hiển thị thông tin các Kịch bản thuộc thiết bị. Các chức năng của thẻ thiết bị:

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG



Hình 6.24: Một số Thẻ kiểu thiết bị

- Thẻ hiện kiểu, tên và chân cắm của thiết bị.
- Cho phép người dùng ngưng hoạt động tạm thời (Disable) hoặc tái hoạt động (Enable) thiết bị.
- Cho phép người dùng xem, cập nhật chỉnh sửa thông tin hoặc xóa thiết bị như ở hình 6.26
- Cho phép người dùng xem thông tin các kịch bản định sẵn của thiết bị.
- Cho phép người dùng tạo thêm kịch bản định sẵn cho thiết bị.

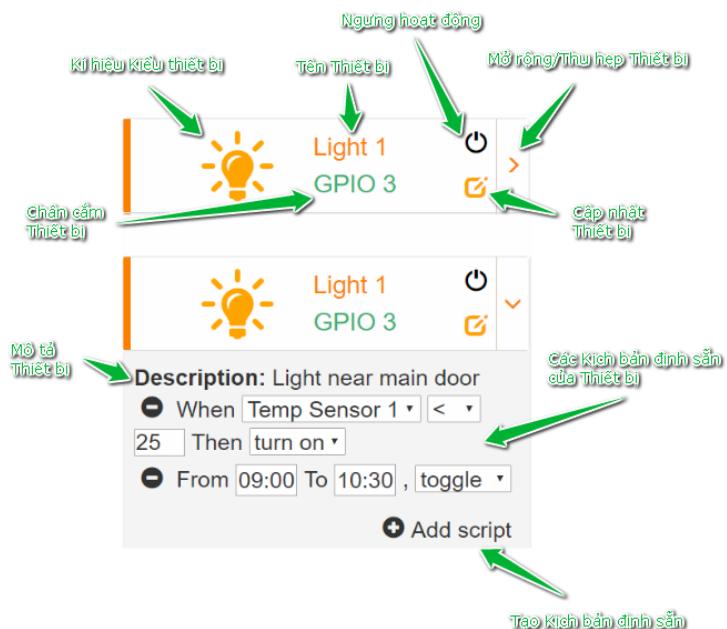
Thẻ kịch bản (Device-script-panel)

Thẻ kịch bản (Device-script-panel) chứa bên trong nó 3 thành phần đóng góp khác là thẻ Kịch bản khi/thì (Device-script-when-then), thẻ Kịch bản từ/đến (Device-script-from-to) và thẻ Kịch bản tự tạo (Device-script-custom). Tùy vào loại kịch bản được truyền vào mà thành phần này sẽ quyết định lựa chọn một trong 3 thành phần đóng góp bên trong nó để thể hiện.

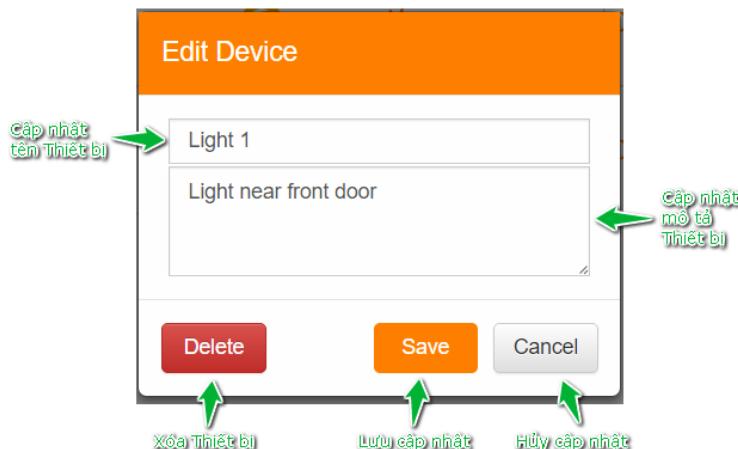
Thẻ Kịch bản khi/thì (Device-script-when-then)

Thẻ kịch bản khi/thì (Device-script-when-then) ở hình 6.27 là thành phần chia sẻ thể hiện thông tin của một kịch bản thuộc loại khi/thì (When/Then), với 2 bộ phận là điều kiện và hành động. Bộ phận điều kiện gồm 3 bộ phận nhỏ hơn là thiết bị điều kiện, trạng thái của thiết bị điều kiện và tham số của thiết bị điều kiện. Tham số của thiết bị điều kiện có được thể hiện hay không tùy thuộc vào kiểu của thiết bị điều kiện. Các chức năng của thẻ kịch bản khi/thì:

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG



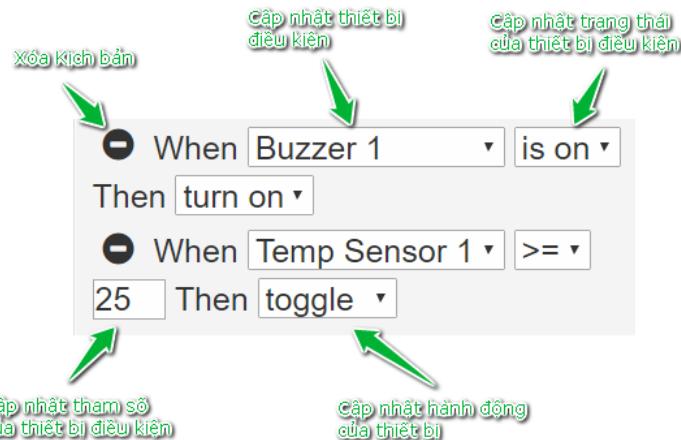
Hình 6.25: Thẻ Thiết bị



Hình 6.26: Bảng hộp thoại cập nhật Thiết bị

- Thể hiện nội dung bao gồm điều kiện và hành động của kịch bản thuộc loại khi/thì.
- Cho phép người dùng cập nhật chỉnh sửa thiết bị điều kiện.
- Cho phép người dùng cập nhật chỉnh sửa trạng thái của thiết bị điều kiện.
- Cho phép người dùng cập nhật chỉnh sửa tham số của thiết bị điều kiện.
- Cho phép người dùng cập nhật chỉnh sửa hành động của thiết bị.
- Cho phép người dùng xóa kịch bản.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG



Hình 6.27: Một số Thẻ Kịch bản khi/thì

Thẻ kịch bản từ/đến (Device-script-from-to)



Hình 6.28: Thẻ Kịch bản từ/đến

Thẻ kịch bản từ/đến (Device-script-from-to) ở hình 6.28 là thành phần chia sẻ thẻ hiện thông tin của một kịch bản thuộc loại từ/đến (From/To), với 2 bộ phận là khoảng thời gian hành động được thực hiện và hành động. Các chức năng của thẻ kịch bản từ/đến:

- Thẻ hiện nội dung bao gồm khoảng thời gian hành động được thực hiện và hành động của kịch bản thuộc loại từ/đến.
- Cho phép người dùng cập nhật chỉnh sửa khoảng thời gian thực hiện hành động của thiết bị.
- Cho phép người dùng cập nhật chỉnh sửa hành động của thiết bị.
- Cho phép người dùng xóa kịch bản.

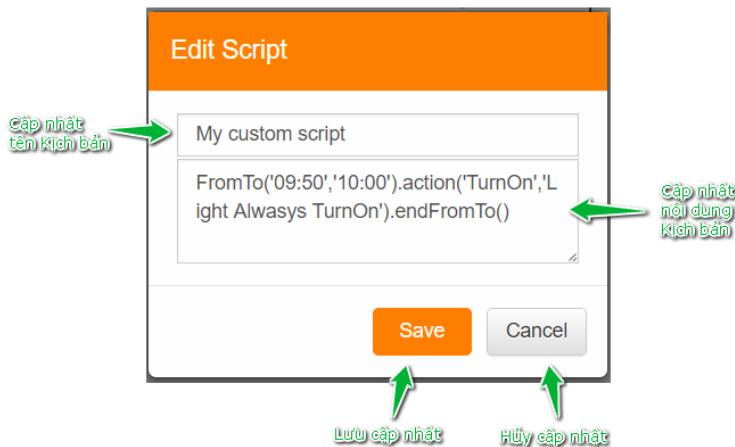
Thẻ kịch bản tự tạo (Device-script-custom)

Thẻ kịch bản tự tạo (Device-script-custom) ở hình 6.29 là thành phần chia sẻ thẻ hiện thông tin của một kịch bản tự tạo, với khả năng tùy biến nội dung kịch bản phức tạp cần nhiều thiết bị, điều kiện và hành động. Thành phần này gồm 2 bộ phận là tên kịch bản và nội dung kịch bản. Các chức năng của thẻ kịch bản tự tạo:

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG



Hình 6.29: Thẻ Kịch bản tự tạo



Hình 6.30: Bảng hộp thoại cập nhật Kịch bản tự tạo

- Thẻ hiện tên của kịch bản tự tạo.
- Cho phép người dùng cập nhật chỉnh sửa tên và nội dung của kịch bản như ở hình 6.30.
- Cho phép người dùng xóa kịch bản.

Các trang giao diện chính (main pages)

Các trang giao diện chính trong ứng dụng bao gồm 6 trang:

- Trang đăng nhập (Login Page).
- Trang đăng ký (Registration Page).
- Trang danh sách các ngôi nhà (Home List Page).
- Trang danh sách các kiểu thiết bị (Device Category List Page).
- Trang danh sách các thiết bị (Device List Page).
- Trang danh sách các kịch bản tự tạo (Custom Script List Page).

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

Các trang giao diện chính này là các trang HTML, với mỗi trang có một bộ quản lý (controller) của AngularJS đảm nhận các xử lý nghiệp vụ riêng.

Trang đăng nhập (Login Page)

The screenshot shows a 'Login Form' with the following components:

- A 'Username' input field with a green arrow pointing to it labeled 'Tên đăng nhập'.
- A 'Password' input field with a green arrow pointing to it labeled 'Mật khẩu'.
- An orange 'Login' button.
- The word 'OR' in the center.
- An orange 'Register' button.
- Green arrows pointing to the 'Login' and 'Register' buttons are labeled 'Đăng nhập' and 'Đăng ký' respectively.

Hình 6.31: Trang đăng nhập

Trang đăng nhập (Login Page) ở hình 6.13 là trang giao diện đầu tiên người dùng truy cập khi sử dụng ứng dụng, cung cấp giao diện đơn giản để người dùng thực hiện thao tác đăng nhập vào ứng dụng. Các chức năng chính của trang đăng nhập:

- Cho phép người dùng đăng nhập vào ứng dụng với tên đăng nhập và mật khẩu.
- Chuyển tiếp người dùng đến trang đăng ký.
- Chuyển tiếp người dùng đến trang danh sách các ngôi nhà.

Trang đăng ký (Registration Page)

The screenshot shows a 'Register' form with the following components:

- A 'Fullname' input field with a green arrow pointing to it labeled 'Tên đầy đủ'.
- A 'Username' input field with a green arrow pointing to it labeled 'Tên đăng nhập'.
- A 'Password' input field with a green arrow pointing to it labeled 'Mật khẩu'.
- A 'Confirm password' input field with a green arrow pointing to it labeled 'Xác nhận mật khẩu'.
- An 'Email' input field with a green arrow pointing to it labeled 'Địa chỉ Email'.
- An orange 'Submit' button.
- Green arrows pointing to the 'Submit' button are labeled 'Đăng ký'.

Hình 6.32: Trang đăng ký

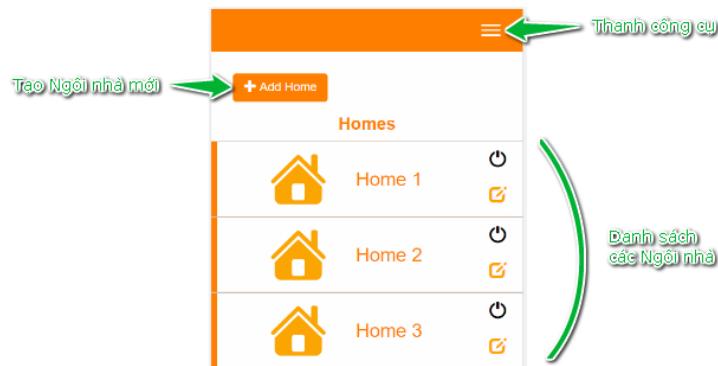
Trang đăng ký (Registration Page) ở hình 6.32 cung cấp giao diện đơn giản để người dùng thực hiện thao tác đăng ký vào ứng dụng. Khi đăng ký với các thông tin hợp lệ, ứng dụng

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

sẽ gửi một Email tới địa chỉ Email đã nhập của người dùng để xác nhận thông tin đăng ký. Các chức năng chính của trang đăng ký:

- Cho phép người dùng đăng ký sử dụng ứng dụng với tên đầy đủ, tên đăng nhập, mật khẩu và địa chỉ Email.
- Chuyển tiếp người dùng về trang đăng nhập.

Trang danh sách các ngôi nhà (Home List Page)



Hình 6.33: Trang danh sách các ngôi nhà

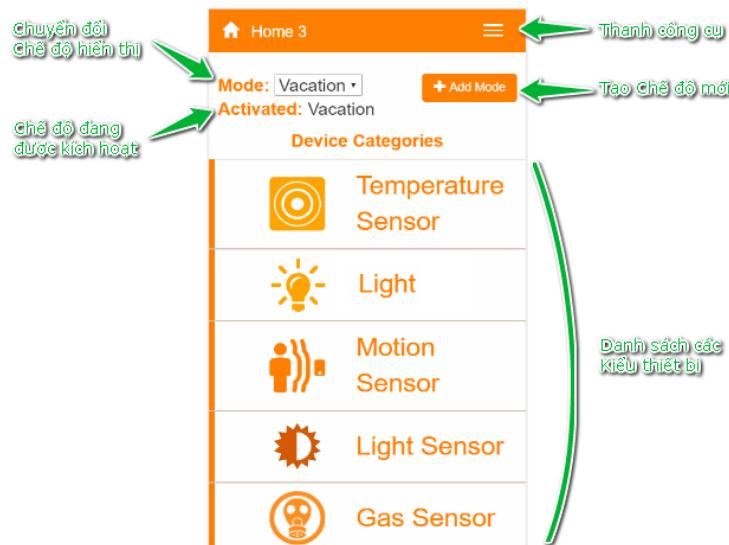
Hình 6.34: Bảng hộp thoại tạo ngôi nhà mới

Trang danh sách các ngôi nhà (Home List Page) ở hình 6.33 cung cấp giao diện để người dùng xem thông tin, quản lý và tạo mới các ngôi nhà được lắp đặt hệ thống. Trang giao diện này sử dụng các thành phần chia sẻ là thanh công cụ và thẻ nhà để thể hiện thông tin. Các chức năng chính của trang danh sách các ngôi nhà:

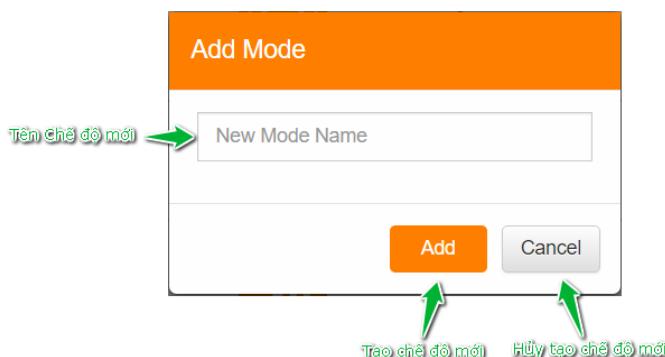
CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

- Thể hiện thông tin danh sách các ngôi nhà của người dùng.
- Cho phép người dùng tạo mới một ngôi nhà như ở hình 6.34.
- Chuyển tiếp người dùng đến trang danh sách các kiểu thiết bị.

Trang danh sách các kiểu thiết bị (Device Category List Page)



Hình 6.35: Trang danh sách các kiểu thiết bị



Hình 6.36: Bảng hộp thoại tạo chế độ mới

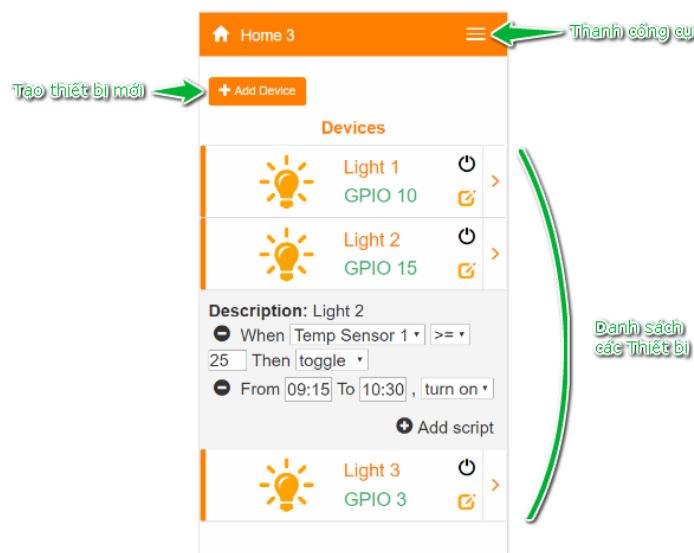
Trang danh sách các kiểu thiết bị (Device Category List Page) ở hình 6.35 cung cấp giao diện để người dùng xem thông tin các kiểu thiết bị, đồng thời kích hoạt, tạo mới và chuyển đổi các chế độ hiển thị. Trang giao diện này sử dụng các thành phần đóng góp là thanh công cụ và thẻ Kiểu thiết bị để thể hiện thông tin. Các chức năng chính của trang danh sách các kiểu thiết bị:

- Thể hiện thông tin danh sách các kiểu thiết bị mà ứng dụng hỗ trợ.

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

- Cho phép người dùng chuyển đổi chế độ hiển thị.
- Cho phép người dùng kích hoạt một chế độ.
- Cho phép người dùng tạo mới một chế độ như ở hình 6.36.
- Chuyển tiếp người dùng đến trang danh sách các thiết bị.
- Chuyển tiếp người dùng đến trang danh sách các kịch bản tự tạo.

Trang danh sách các thiết bị (Device List Page)



Hình 6.37: Trang danh sách các thiết bị



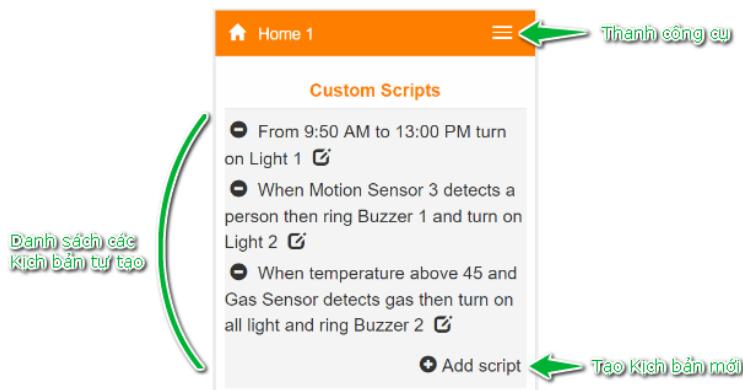
Hình 6.38: Bảng hộp thoại tạo thiết bị mới

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

Trang danh sách các thiết bị (Device List Page) ở hình 6.37 cung cấp giao diện để người dùng xem thông tin, quản lý và tạo mới các thiết bị. Trang giao diện này sử dụng các thành phần đóng góp là thanh công cụ và thẻ thiết bị để thể hiện thông tin. Các chức năng chính của trang danh sách các thiết bị:

- Thể hiện thông tin danh sách các thiết bị thuộc một kiểu thiết bị của người dùng.
- Cho phép người dùng tạo mới một thiết bị như ở hình 6.38.

Trang danh sách các kịch bản tự tạo (Custom Script List Page)



Hình 6.39: Trang danh sách các kịch bản tự tạo



Hình 6.40: Bảng hộp thoại tạo kịch bản tự tạo mới

Trang danh sách các kịch bản tự tạo (Custom Script List Page) ở hình 6.39 cung cấp giao diện để người dùng xem thông tin, quản lý và tạo mới các kịch bản tự tạo. Trang giao diện này sử dụng các thành phần đóng góp là thanh công cụ và thẻ Kịch bản tự tạo để thể hiện thông tin. Các chức năng chính của trang danh sách các kịch bản tự tạo:

CHƯƠNG 6. HIỆN THỰC HỆ THỐNG

- Thể hiện thông tin danh sách các kịch bản tự tạo của người dùng.
- Cho phép người dùng tạo mới một kịch bản tự tạo như ở hình 6.40.

Với việc hiện thực 6 trang giao diện chính cùng các thành phần chia sẻ như trên, ứng dụng di động cung cấp đầy đủ các chức năng cần thiết để người dùng có thể dễ dàng quản lý các ngôi nhà, thiết bị và kịch bản của hệ thống cũng như phản ánh được kiến trúc tổng quát đã được trình bày ở Chương 5.

Chương 7

Thí nghiệm và đánh giá hệ thống

7.1 Phương pháp thực nghiệm

Nhóm tiến hành lắp đặt lần lượt các thiết bị gồm: cảm biến nhiệt, cảm biến ánh sáng, cảm biến khí gas, cảm biến chuyển động, đèn và còi hú vào Raspberry Pi. Hình XX mô tả thiết bị demo của nhóm được gắn trên Raspberry Pi. Sau đó, nhóm cho thiết bị mobile cùng Raspberry Pi kết nối đến cùng 1 mạng local qua sử dụng máy tính làm Wifi hotspot. Nhóm tiến hành đăng ký thông qua giao diện ứng dụng và tạo mới nhà, tạo mới chế độ, thiết bị, thêm kịch bản. Một vài kịch bản mà nhóm chuẩn bị cho thí nghiệm.

Chế độ 1 (kiểm tra cảm biến ánh sáng):

- Nếu cảm biến ánh sáng phát hiện xung quanh sáng thì đèn tắt.
- Nếu cảm biến ánh sáng phát hiện xung quanh tối thì đèn bật.

Chế độ 2 (kiểm tra cảm biến khí gas):

- Nếu cảm biến khí gas phát hiện nồng độ gas vượt quá một ngưỡng cho phép (có thể điều chỉnh thông qua cảm biến khí gas) thì còi hú.
- Nếu cảm biến khí gas phát hiện nồng độ gas ở mức bình thường thì còi tắt.
- Chuẩn bị kịch bản có khả năng mâu thuẫn: Trong khoảng 19h00 đến 20h00 thì còi tắt.

Chế độ 3 (kịch bản thời gian):

- Trong khoảng thời gian từ 18h00 đến 22h00 thì đèn bật.
- Trong khoảng thời gian từ 22h01 đến 17h59 thì đèn tắt.
- Chuẩn bị kịch bản mâu thuẫn: Trong khoảng 19h00 đến 20h00 thì đèn tắt.

Chế độ 4 (nhiều kịch bản phối hợp):

- Nếu nhiệt độ thu được từ cảm biến nhiệt lớn hơn 31 độ thì đèn bật.
- Nếu nhiệt độ thu được từ cảm biến nhiệt bé hơn 31 độ thì đèn tắt.
- Nếu đèn bật thì còi hú.
- Nếu đèn tắt thì còi tắt.

Chế độ 5 (kiểm tra cảm biến chuyển động cùng phối hợp kịch bản thời gian):

- Nếu phát hiện vật thể chuyển động thì đèn bật và nếu trong khoảng thời gian từ 22h00 đến 06h00 thì còi hú.

CHƯƠNG 7. THÍ NGHIỆM VÀ ĐÁNH GIÁ HỆ THỐNG

- Nếu không còn phát hiện vật thể chuyển động nữa thì đèn và còi tắt.

Chế độ 6 (kịch bản phức tạp):

Nếu (cảm biến ánh sáng phát hiện xung quanh tối) **thì**
(bật đèn và nêu (phát hiện khí gas) **thì**
(đèn tắt đèn, mở còi hú)
hoặc không mà trong thời gian **từ** 09h00 **đến** 14h00 **thì**
(tắt đèn và tắt còi)
)

Ngôi nhà sẽ có lần lượt 5 chế độ với các kịch bản dành cho các thiết bị như trên. Việc đánh giá hệ thống được thực hiện thủ công bằng cách so sánh kết quả mong đợi từ kịch bản với kết quả thực tế quan sát thông qua thiết bị trong cùng điều kiện. Ngoài ra, nhóm còn thực hiện unit test cho việc kiểm tra kịch bản hợp lệ (trùng tên, kịch bản mâu thuẫn, kịch bản có khả năng mâu thuẫn, kịch bản trùng nội dung nhưng đảo thứ tự điều kiện, hành động,...).

7.2 Kết quả và đánh giá

Sau khi tiến hành các thực nghiệm trên, nhóm rút ra một số đánh giá sơ bộ về hệ thống:

- Các thiết bị hoạt động tốt với Raspberry Pi.
- Với hầu hết các kịch bản mô tả từ đơn giản đến phức tạp thì hệ thống vẫn có thể xử lý tốt, chạy đúng kết quả mong muốn (ví dụ như các kịch bản ở phần thực nghiệm trên) miễn rằng nó là kịch bản hợp lệ.
- Việc kiểm tra các kịch bản mâu thuẫn, có khả năng mâu thuẫn để loại trừ ra khỏi hệ thống hoạt động ổn định và tốt.
- Hệ thống hỗ trợ bảo mật tài khoản người dùng, bảo mật tài nguyên hệ thống (chỉ có người dùng sở hữu ngôi nhà mới được truy xuất đến các tài nguyên thuộc nhà ấy).
- Khi thực hiện việc vô hiệu hóa (disable) ngôi nhà, hay thiết bị thì các kịch bản lần lượt thuộc nhà và thiết bị ấy sẽ vào trạng thái dừng như mong đợi. Cũng như thực hiện thao tác kích hoạt (enable) nhà, thiết bị trở lại thì kịch bản sẽ chạy lại như bình thường.
- Giao diện ứng dụng đơn giản, cung cấp nhiều chức năng cho người dùng thao tác, chỉnh sửa và xem thông tin về ngôi nhà và các thiết bị, kịch bản trong nó. Ngoài ra, ứng dụng cho phép tạo nhanh kịch bản với vài mẫu đơn giản.

Tuy nhiên, còn một số khó khăn mà hệ thống đang gặp phải:

- Giao diện chưa hỗ trợ tối đa người dùng trong việc xây dựng các kịch bản tùy ý (custom).

CHƯƠNG 7. THÍ NGHIỆM VÀ ĐÁNH GIÁ HỆ THỐNG

- Trải nghiệm người dùng chưa thực sự tốt với thời gian phản hồi hệ thống còn khá cao.
- Các mẫu kịch bản hỗ trợ săn còn ít, chưa thỏa mãn được nhiều nhu cầu thực tế người dùng.
- Khi số lượng kịch bản tăng lên (tầm 30-35) thì hiệu năng hệ thống giảm đi khá nhiều, do máy tính Raspberry Pi có bộ nhớ khá “khiêm tốn” và việc quản lý kịch bản chưa thực sự tốt lắm.

Chương 8

Thảo luận

8.1 Giải pháp xử lý các kịch bản có khả năng mâu thuẫn

Tại thời điểm viết luận văn này, các kịch bản có khả năng mâu thuẫn sẽ không được phép tồn tại trong hệ thống. Nhóm muốn đề xuất một phương pháp có thể xử lý tình huống này khi mở rộng hệ thống trong tương lai, đó là cho phép người dùng thiết lập thêm độ ưu tiên cho kịch bản trong trường hợp có khả năng mâu thuẫn. Giả sử trong một điều kiện nào đó, 2 kịch bản có khả năng mâu thuẫn thực sự mâu thuẫn nhau, ta hoàn toàn có thể dựa trên độ ưu tiên để xem xét kịch bản nào được chạy trong trường hợp trên, bằng cách chọn ra kịch bản có độ ưu tiên cao hơn. Tuy nhiên trong điều kiện bình thường, việc xét độ ưu tiên khi tạo ra một kịch bản mới có thể không mang lại nhiều ý nghĩa và khiến người dùng bị rối. Đứng từ khía cạnh người dùng, hệ thống có thể được xem là thông minh hơn nếu như có khả năng tự thiết lập một độ ưu tiên mặc định cho một số dạng kịch bản, dựa trên ý nghĩa, nhu cầu thực tế của kịch bản đó. Người dùng có thể tự do thay đổi giá trị mặc định đó dựa trên mục đích riêng của họ.

8.1.1 Độ ưu tiên mặc định cho một số loại kịch bản

Sau khi tham khảo từ một số nhu cầu thực tế, nhóm đề xuất ra một số dạng kịch bản có thể được xét giá trị ưu tiên mặc định. Nếu như kịch bản chứa một trong những điều kiện thuộc dạng khẩn cấp sau thì chúng lần lượt có thứ tự ưu tiên như sau (số thứ tự càng thấp thì độ ưu tiên càng cao):

1. Temperature > N (N là một giá trị nào đó).
2. Phát hiện có gas.
3. Phát hiện người chuyển động trong thời gian đêm khuya.
4. Các dạng kịch bản khác.

Có thể giải thích cho việc đưa ra quyết định trên như sau:

- Nếu nhiệt độ vượt ngưỡng => Khả năng có cháy xảy ra cao, dẫn đến độ nguy hiểm tính mạng cao => Ưu tiên số 1.
- Trường hợp có gas, tức có khả năng xảy ra cháy, nhưng thật sự vẫn chưa cháy => Ưu tiên số 2.

CHƯƠNG 8. THẢO LUẬN

- Trường hợp này có khả năng là trộm => dẫn đến có thể bị mất đồ nhưng khả năng gây chết người thấp => Ưu tiên số 3.

Nếu kịch bản có chứa nhiều điều kiện dạng trên thì độ ưu tiên mặc định của kịch bản ấy chỉnh là độ ưu tiên cao nhất của một trong các điều kiện liệt kê ở trên.

8.1.2 Cách xử lý một số trường hợp trùng độ ưu tiên

Như đã thảo luận ở mục trên, việc xử lý kịch bản có khả năng mâu thuẫn có thể giải quyết bằng độ ưu tiên. Nhưng một số trường hợp, chúng có cùng độ ưu tiên (người dùng không có ý muốn thay đổi độ ưu tiên mặc định) thì ta phải xử lý ra sao? Nhóm xin liệt kê và giải thích qua một vài ví dụ.

Ví dụ 1:

- Kịch bản 1: Nếu có gas thì tắt đèn phòng.
- Kịch bản 2: Nếu có gas thì bật còi hú gần cửa ra vào và đồng thời kiểm tra nếu cảm biến ánh sáng cho kết quả bây giờ là ban ngày thì bật đèn phòng.

Nhìn sơ qua, 2 kịch bản trên có khả năng mâu thuẫn nhau và đang có cùng độ ưu tiên mặc định là 2. Trường hợp này, nhóm đề xuất nếu như kịch bản nào có độ chi tiết hơn thì nó sẽ được xem là có độ “ưu tiên” cao hơn. Lý do là vì đứng từ góc nhìn người đặt ra kịch bản, việc thực thi 1 kịch bản có mức chi tiết hơn sẽ đáp ứng mong đợi người dùng nhiều hơn.

Ví dụ 2:

- Kịch bản 1: Từ 22h đến 6h thì tắt đèn phòng.
- Kịch bản 2: Từ 5h đến 7h thì bật đèn phòng.

Trường hợp này, nhóm đề xuất kịch bản 2 có độ ưu tiên cao hơn. Có thể giải thích rằng là do kịch bản bật đèn từ 5h đến 7h có khoảng giá trị nhỏ hơn từ 22h -> 6h, mang nghĩa kịch bản 2 có mức độ xuất hiện không thường xuyên cao hơn kịch bản 1 cho nên có ưu tiên cao hơn.

Nếu xảy ra một trường hợp khác không nằm trong các trường hợp liệt kê mục này, hệ thống có thể thông báo cho người dùng, yêu cầu họ nhập giá trị ưu tiên cho các kịch bản có khả năng mâu thuẫn trên.

8.2 Giải pháp tối ưu việc quản lý, vận hành các kịch bản dựa trên kỹ thuật multi-threading

Kịch bản là một khái niệm quan trọng trong hệ thống này. Như ta đã biết, kịch bản được tạo ra, được hệ thống chuyển đổi, kiểm tra hợp lệ,... và cuối cùng là chạy nó. Với thiết kế hệ thống hiện tại, mỗi kịch bản được quản lý bởi 1 thread riêng biệt. Khi kịch bản ở trạng thái chạy, cứ cách một khoảng thời gian ngắn, kịch bản ấy sẽ được kiểm tra lại điều kiện

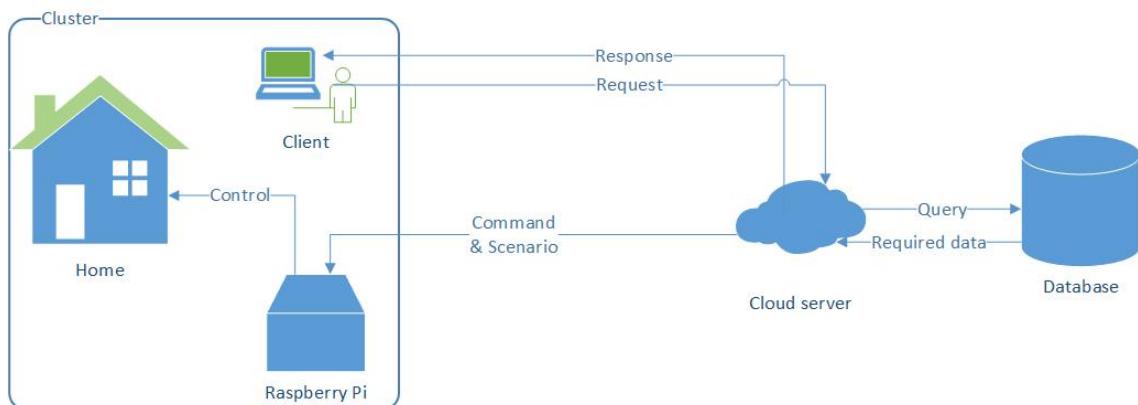
CHƯƠNG 8. THẢO LUẬN

và thực thi hành động tương ứng, điều đó diễn ra thành vòng lặp vô tận. Có thể dễ ý thấy, mỗi thread không luôn luôn phải ở trạng thái sẵn sàng (active) mà nó chỉ cần hoạt động sau mỗi khoảng thời gian ngắn. Từ đó dẫn đến việc lãng phí một lượng lớn tài nguyên hệ thống (ở đây là lãng phí xử lý CPU và cả bộ nhớ, memory) chỉ phục vụ cho mục đích chờ đợi là chủ yếu. Thử tưởng tượng nếu hệ thống có tầm 20 kịch bản, đồng nghĩa có 20 threads tại thời điểm này, trong khi bộ nhớ của Raspberry Pi khá thấp, CPU cũng không đủ mạnh, việc xử lý đa luồng và mỗi luồng là một kịch bản có thể sẽ làm giảm hiệu năng toàn hệ thống.

Sau thời gian tìm hiểu và thảo luận, nhóm xin đề xuất một thuật toán cải tiến nhỏ giúp cho việc quản lý trên hiệu quả hơn. Ta sẽ vẫn giữ mỗi thread quản lý một kịch bản, tuy nhiên điểm khác biệt đó là việc chạy kịch bản chỉ được thực hiện 1 lần khi tạo, hay 1 lần sau khi cập nhật mới nội dung kịch bản. Trong module “Scenario Runner”, ta hiện thực thêm 1 hashmap “scenario mapping” chả hạn, để ánh xạ kịch bản với thread quản lý nó. Nếu như có bất kì thay đổi nào tới kịch bản, dựa trên hashmap “scenario mapping” trên để thông báo đến thread tương ứng và chạy lại kịch bản mới ấy. Với cách hiện thực này, việc lãng phí tài nguyên CPU sẽ giảm đi đáng kể và chỉ còn chi phí khi lưu thread trên bộ nhớ thôi.

8.3 Giải pháp xây dựng hệ thống ứng dụng điện toán đám mây phục vụ nhiều người dùng

Trong luận văn này, ứng dụng hệ thống chỉ dừng lại ở mức demo với các thiết bị đơn giản và mọi dịch vụ được cung cấp bởi Raspberry Pi. Tuy nhiên, khi hệ thống này được áp dụng vào thực tế, nhóm đề xuất sẽ đưa các dịch vụ lên trên đám mây (cloud). Cụ thể, ta sẽ có 1 cloud server, cung cấp các dịch vụ liên quan việc quản lý thông tin các ngôi nhà, chế độ, thiết bị, kịch bản người dùng và cả Raspberry Pi trong nhà.



Hình 8.1: Cách thức giao tiếp giữa client, server và Raspberry Pi

CHƯƠNG 8. THẢO LUÂN

Hình 8.1 mô tả cách giao tiếp giữa người dùng, Pi và cloud server. Khi người dùng đăng ký sử dụng dịch vụ của hệ thống, họ sẽ được cấp 1 con Pi với mã code do ta quy định và được lưu trên cloud. Sau đó, người dùng được trợ giúp lắp đặt thiết bị vào con Pi đó và tiến hành đăng ký với hệ thống (thông qua giao diện ứng dụng) để xác nhận việc muốn gắn Pi vào ngôi nhà của mình. Hệ thống nhận được yêu cầu đăng ký và dựa trên mã code gửi kèm sẽ kích hoạt con Pi có mã tương ứng. Kể từ lúc này, thông qua giao diện ứng dụng, người dùng có thể thêm nhà, thiết bị, kịch bản,... Mọi thông tin này được lưu trữ và quản lý trên cloud. Raspberry Pi sau khi đã kích hoạt cũng sẽ kết nối với cloud và nhận các thông tin về kịch bản gửi từ cloud server. Nhiệm vụ của Pi giờ chỉ là thực thi những kịch bản đó cùng với việc điều khiển các thiết bị theo như kịch bản đã định.

Chương 9

Tổng kết

Với mục tiêu của đề tài “Xây dựng hệ thống nhà thông minh sử dụng Raspberry Pi” là hướng tới phát triển một hệ thống điều khiển các thiết bị thông dụng trong nhà thông qua các kịch bản người dùng mong muốn một cách dễ dàng và tiện lợi, chúng tôi đã tìm hiểu được cách thức hoạt động và điều khiển các thiết bị phần cứng được gắn trên máy tính kích thước nhỏ Raspberry Pi, hiện thực một máy chủ với vai trò xử lý các kịch bản của người dùng và một ứng dụng di động cho phép người dùng dễ dàng truy cập và quản lý các thiết bị.

Về mặt các thiết bị hỗ trợ, chúng tôi đã triển khai việc lắp đặt thành công các thiết bị: Bóng đèn, còi hú, cảm biến nhiệt độ, cảm biến chuyển động, cảm biến ánh sáng và cảm biến khí gas vào Raspberry Pi. Trong quá trình lắp đặt, có một số khó khăn xảy ra như các thiết bị có cách thức hoạt động khác nhau, <[điền thêm vô nha Tùng]>. Các khó khăn này đã được nhóm giải quyết thông qua <[điền thêm vô nha Tùng]>. Vì mục tiêu đề tài hướng trọng tâm vào việc xử lý các tác vụ trên phương diện phần mềm, nên việc lắp đặt các thiết bị phần cứng còn tồn tại một số hạn chế như là các thiết bị vẫn còn dùng ở mức thí nghiệm, chưa phải là các thiết bị thật được sử dụng trong thực tế cũng như số lượng chủng loại thiết bị chưa thực sự nhiều.

Về các kịch bản điều khiển thiết bị, nhóm chúng tôi cung cấp cho người dùng 2 dạng kịch bản với cấu trúc được định sẵn là Khi/Thì (If/Then) và Từ/Đến (From/To), thể hiện mặt luận lý đơn giản và người dùng có thể dễ dàng chỉnh sửa. Ngoài ra, để có thể thiết lập các kịch bản với cấu trúc phức tạp hơn, nhóm chúng tôi còn cung cấp dạng kịch bản tự tạo (custom) cho phép người dùng có thể kết hợp nhiều thiết bị với nhiều điều kiện khác nhau thông qua hệ thống cú pháp định sẵn. Trong quá trình hiện thực, nhóm đã gặp phải khó khăn trong việc xử lý các kịch bản mâu thuẫn, kịch bản có khả năng mâu thuẫn. Hơn nữa, về mặt hiệu năng hệ thống, các kịch bản vẫn chưa được quản lý tối ưu và cần cải thiện để tăng sức tải của hệ thống để đáp ứng số lượng lớn kịch bản người dùng.

Về ứng dụng di động, chúng tôi đã xây dựng được ứng dụng với thiết kế giao diện đơn giản cung cấp các tính năng cơ bản để người dùng có thể quản lý các ngôi nhà, các thiết bị cũng như các kịch bản điều khiển trong hệ thống. Một số khó khăn nhóm gặp phải trong quá trình xây dựng ứng dụng như việc thiếu tính đồng nhất giữa ứng dụng và máy chủ, việc lọc các thiết bị cùng chân cắm khi có kịch bản liên hệ chưa thực sự chính xác đã được nhóm giải quyết tương đối triệt để. Ứng dụng còn tồn tại một số hạn chế như thời gian đáp ứng người dùng đôi khi còn chậm, chưa thực sự tối ưu hóa trải nghiệm vì hiện tại máy chủ được cài đặt trực tiếp trên máy tính Raspberry Pi nên tốc độ xử lý các yêu cầu từ

CHƯƠNG 9. TỔNG KẾT

phía ứng dụng còn hạn hẹp.

Nhìn chung, hệ thống chúng tôi phát triển đã đáp ứng được mục tiêu của đề tài về việc xây dựng hệ thống nhà thông minh sử dụng Raspberry Pi điều khiển các thiết bị thông qua các kịch bản tuy vẫn còn một số hạn chế. Để phát triển hệ thống trong tương lai cũng như khắc phục các mặt còn hạn chế, chúng tôi dự định sẽ mở rộng và cải thiện hệ thống theo các hướng:

- Hỗ trợ thêm nhiều chủng loại thiết bị (như camera, khóa cửa,...) và triển khai lắp đặt các thiết bị thực tế.
- Xử lý và quản lý tốt hơn các kịch bản mâu thuẫn, kịch bản có khả năng mâu thuẫn.
- Hỗ trợ thêm các dạng kịch bản mới sát với mục đích thực tế, mở rộng cấu trúc cú pháp kịch bản.
- Tăng cường hiệu năng đáp ứng của máy chủ với các thay đổi của kịch bản.
- Tối ưu hóa trải nghiệm và thời gian đáp ứng người dùng của ứng dụng di động.
- Đưa máy chủ lên nền tảng đám mây (Cloud Server) độc lập với Raspberry Pi.
- Áp dụng cơ chế bảo mật quét và diệt virus cho máy chủ cũng như Raspberry Pi để tăng cường bảo mật cho hệ thống và các ngôi nhà được lắp đặt hệ thống.

Nhà thông minh hiện đang dần trở thành một xu hướng tất yếu của nền văn minh hiện đại ngày nay với sự phát triển không ngừng nghỉ trên toàn cầu cũng như trong nước. Với mong muốn mang đến một giải pháp nhà thông minh tiết kiệm, đầy đủ và thuận tiện cho người dùng, chúng tôi hy vọng hệ thống mà nhóm đã làm được sẽ đóng góp một phần vào trong sự phát triển đó.

Tài liệu tham khảo

- [1] R. Abed. Hybrid vs native mobile apps – the answer is clear. [Online] Available at: <<https://www.ymediolabs.com/hybrid-vs-native-mobile-apps-the-answer-is-clear/>>, 2016. [Accessed 01 December 2016].
- [2] R. P. Foundation. Gpio: Models a+, b+, raspberry pi 2 b and raspberry pi 3 b. [Online] Available at: <<https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/>>. [Accessed 26 November 2016].
- [3] Gartner. Gartner says worldwide iot security spending to reach \$348 million in 2016. [Online] Available at: <<http://www.gartner.com/newsroom/id/3291817>>, 2016. [Accessed 12 December 2016].
- [4] V. Group. Xu hướng nhà thông minh năm 2016 - vinteli home. [Online] Available at: <<http://vinteligroup.com/bai-viet/xu-huong-nha-thong-minh-nam-2016-vinteli-home-41>>, 2016. [Accessed 13 December 2016].
- [5] K. Hà. Giải pháp nhà thông minh. [Online] Available at: <<http://www.pcworld.com.vn/articles/cong-nghe/song-va-cong-nghe/2014/10/1236584/giai-phap-nha-thong-minh/>>, 2014. [Accessed 11 December 2016].
- [6] D. Innovation. Phân biệt ứng dụng gốc và ứng dụng web. [Online] Available at: <<http://dvms.vn/tin-tuc/tin-nganh/114-phan-biet-ung-dung-goc-native-application-va-ung-dung-web-web-application.html>>, 2011. [Accessed 30 November 2016].
- [7] S. McManus and M. Cook. *Raspberry Pi For Dummies*. John Wiley & Sons, Inc., 2013.
- [8] H. Nguyen. Những câu hỏi phổ biến về spring framework. [Online] Available at: <<http://www.javadevchannel.com/2015/07/nhung-cau-hoi-pho-bien-ve-spring.html>>, 2015. [Accessed 29 November 2016].
- [9] NuBryte. Top 4 smart home trends for 2016. [Online] Available at: <<http://www.nubryte.com/posts/87-top-4-smart-home-trends-for-2016>>, 2016. [Accessed 14 December 2016].
- [10] T. T. Reviews. The best home automation systems of 2016. [Online] Available at: <<http://www.toptenreviews.com/home/smart-home/best-home-automation-systems/>>, 2016. [Accessed 13 December 2016].

TÀI LIỆU THAM KHẢO

- [11] A. Rodriguez. Restful web services: The basics. [Online] Available at: <<http://www.ibm.com/developerworks/library/ws-restful/>>, 2015. [Accessed 30 November 2016].
- [12] P. Software. Spring framework. [Online] Available at: <<https://projects.spring.io/spring-framework/>>, 2016. [Accessed 29 November 2016].
- [13] S. Sundararajan. Spring framework: @RestController vs @Controller. [Online] Available at: <<https://www.genuitec.com/spring-frameworkrestcontroller-vs-controller/>>, 2015. [Accessed 30 November 2016].
- [14] R. team. Frontpage - raspbian. [Online] Available at: <<https://www.raspbian.org/FrontPage>>. [Accessed 28 November 2016].
- [15] Wikipedia. Raspberry pi. [Online] Available at: <https://en.wikipedia.org/wiki/Raspberry_Pi>. [Accessed 27 November 2016].

Phụ lục

Một số kịch bản thông dụng

Các điều kiện (condition) đang có:

- Đèn bật, đèn tắt.
- Còi bật, còi tắt.
- Nhiệt độ $>$, $<$, \geq , \leq .
- Có gas.
- Có người.
- Trời sáng, trời tối.

Các hành động (action) đang có:

- Bật đèn, tắt đèn.
- Bật còi, tắt còi.

Một vài kịch bản dạng If - Then thông dụng:

- **If** (đèn ngủ bật) **Then** (tắt đèn phòng khách và nhà bếp).
- **If** (đèn cầu thang 1 bật) **Then** (tắt đèn cầu thang 2).
- **If** (đèn cầu thang 2 bật) **Then** (tắt đèn cầu thang 1).
- **If** (còi bật) **Then** (bật tắt cả các đèn).
- **If** (nhiệt độ > 50) **Then** (bật còi và bật tắt cả đèn).
- **If** (có gas) **Then** (bật còi và bật tắt cả các đèn).
- **If** (có người trước cửa và trời tối) **Then** (Bật đèn và hú còi).
- **If** (có người trước cửa và trời sáng) **Then** (Hú còi).
- **If** (có người vào nhà và trời tối) **Then** (Hú còi và bật tắt cả đèn).
- **If** (có người vào nhà và trời sáng) **Then** (Hú còi).
- **If** (có người lên cầu thang) **Then** (bật đèn cầu thang).
- **If** (trời sáng) **Then** (tắt đèn).
- **If** (trời tối) **Then** (bật đèn).

Một vài kịch bản dạng From - To thông dụng:

- **From** (20h) **To** (6h) **Do** (Bật đèn ngủ).
- **From** (6h) **To** (20h) **Do** (Tắt đèn ngủ).
- **From** (20h) **To** (6h) **Do** (**When** (có người) **Then** (Hú còi + Bật đèn)).

TÀI LIỆU THAM KHẢO

- **From (20h) To (6h) Do (When (có gas) Then (Hú còi + Bật đèn)).**
- **From (20h) To (6h) Do (If (nhiệt độ > 50) Then (Hú còi + Bật đèn)).**
- **From (20h) To (6h) Do (If (còi hú) Then (Bật tắt cả các đèn)).**

Danh sách API cung cấp cho client

Danh sách API được liệt kê theo bảng sau:

Page/Function	Method Type	URL	Input	Output
Get list categories given home	GET	homes/homeId/device-types	homeId: Long	HTTP Status OK 200 if successful categories: List<Category>
Get all devices given category	GET	homes/homeId/device-types/deviceTypeId/devices	homeId: Long deviceTypeId: Long	HTTP Status OK 200 if successful Return: List<Device>
Enabled/ Disabled device	PATCH	homes/homeId/device-types/deviceTypeId/devices/deviceId	deviceId: Long homeId: Long deviceTypeId: Long Pass json: "enabled": true to update enabled = true	HTTP Status OK 204 if successful
Delete device in home	DELETE	/homes/homeId/device-types/deviceTypeId/devices/deviceId	deviceId: Long homeId: Long deviceTypeId: Long	HTTP Status OK 204 if successful
Get list scripts given mode	GET	devices/deviceId/modes/modeId/scripts	deviceId: Long modeId: Long	HTTP Status OK 204 if successful Return: List<Script>
Update script of one specific mode	PATCH	devices/deviceId/modes/modeId/scripts/scriptId	deviceId: Long modeId: Long script: Script	HTTP Status OK 204 if successful
Delete script in one specific mode	DELETE	devices/deviceId/modes/modeId/scripts/scriptId	scriptId: Long	HTTP Status OK 204 if successful
Add script to device given mode	POST	devices/deviceId/modes/modeId/scripts	deviceId: Long modeId: Long script: Script	HTTP Status OK 201 if successful Also return the URI of new object. E.g devices/5/modes/1/scripts/3
Add device of specific category	POST	homes/homeId/device-types/deviceTypeId/devices	homeId: Long categoryId: Long device: Device	HTTP Status OK 201 if successful Also return the URI of new object. E.g homes/5/device-types/3/devices/4
Get all valid GPIO pins	GET	homes/all-gpios		HTTP Status OK 200 if successful. Return: List<Integer>
Get home	GET	homes/homeId	homeId: Long	HTTP Status OK 200 if successful. Return: Home
Delete home	DELETE	homes/homeId	homeId: Long	HTTP Status OK 204 if successful

TÀI LIỆU THAM KHẢO

Page/Function	Method Type	URL	Input	Output
Add new home (also add default mode)	POST	homes	home: Home	HTTP Status OK 201 if successful Also return the URI of new object. E.g homes/1
Update home (Enabled/ Disable / Current mode,...)	PATCH	homes/homeId	homeId: Long home: Home Pass json: "enabled": true to update enabled = true	HTTP Status OK 204 if successful
Get list home of one user	GET	homes		HTTP Status OK 200 if successful Return: List<Home>
Get list mode in home	GET	homes/homeId/modes	homeId: Long	HTTP Status OK 200 if successful Return: List<Mode>
Delete mode	DELETE	homes/homeId/modes/ modeId	homeId: Long modeId: Long	HTTP Status OK 204 if successful
Add new mode	POST	homes/homeId/modes	mode:Mode homeId: Long	HTTP Status OK 201 if successful Also return the URI of new object. E.g homes/3/modes/3
Update existing mode	PATCH	homes/homeId/modes/ modeId	homeId: Long modeId: Long mode: Mode	HTTP Status OK 204 if successful
Login	POST	/login	Add headers: X-Username X-Password	HTTP Status: 200 OK 401 Unauthorized
Logout	POST	/logout	Add header: X-Auth-Token	Http Status: 200 OK 401 Unauthorized
Sign up	POST	/users/signup	User information: Username Password (encrypted) Fullname Email	Return: User's activation link Return code: -1: ER-ROR_WHEN_ADD_USER -2: USER-NAME_ALREADY_EXISTED -3: EMAIL_ALREADY_EXISTED > 0: Successful Http status: 200 OK 400 Bad request
Activate user	GET	/users/activation/userId	User id	Http status: 200 OK 400 Bad request
Get user information	GET	/users	X-Auth-Token	Http status: 200 Ok 400 Bad request Return: User model
Get list device id in script	POST	/homes/homeId/scripts	Script object with content and script type id	Http status: 200 OK 400 BAD REQUEST Return list device id in that script.