TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN ĐIỆN TỬ - TRUYỀN THÔNG

ĐỒ ÁN

TỐT NGHIỆP ĐẠI HỌC

Đề tài:

THIẾT KẾ HỆ THỐNG ĐIỀU KHIỂN TIVI BẰNG GIỌNG NÓI ỨNG DỤNG TRONG NHÀ THÔNG MINH

Sinh viên thực hiện: Trần Ngọc Tiến

Lớp KT ĐTTT04 - K58

Giảng viên hướng dẫn: TS. Nguyễn Tiến Hòa

Hà Nội, 6 - 2018

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN ĐIỆN TỬ - TRUYỀN THÔNG

ĐỒ ÁN

TỐT NGHIỆP ĐẠI HỌC

Đề tài:

THIẾT KẾ HỆ THỐNG ĐIỀU KHIỂN TIVI BẰNG GIỌNG NÓI ỨNG DỤNG TRONG NHÀ THÔNG MINH

Sinh viên thực hiện: Trần Ngọc Tiến

Lớp KT ĐTTT04 - K58

Giảng viên hướng dẫn: TS. Nguyễn Tiến Hòa

Cán bộ phản biện:

Hà Nội, 6 - 2018

Đánh giá quyển đồ án tốt nghiệp
(Dùng cho giảng viên hướng dẫn)

Giảng viên đánh giá:.....

Họ và tên Sinh viên:..... MSSV:……...

Tên đồ án: .....

……

Chọn các mức điểm phù hợp cho sinh viên trình bày theo các tiêu chí dưới đây:

Rất kém (1); Kém (2); Đạt (3); Giỏi (4); Xuất sắc (5)

Có sự kết hợp giữa lý thuyết và thực hành (20)						
1	Nêu rõ tính cấp thiết và quan trọng của đề tài, các vấn đề và các giả thuyết (bao gồm mục đích và tính phù hợp) cũng như phạm vi ứng dụng của đồ án	1	2	3	4	5
2	Cập nhật kết quả nghiên cứu gần đây nhất (trong nước/quốc tế)	1	2	3	4	5
3	Nêu rõ và chi tiết phương pháp nghiên cứu/giải quyết vấn đề	1	2	3	4	5
4	Có kết quả mô phỏng/thực nghiệm và trình bày rõ ràng kết quả đạt được	1	2	3	4	5
Có khả năng phân tích và đánh giá kết quả (15)						
5	Kế hoạch làm việc rõ ràng bao gồm mục tiêu và phương pháp thực hiện dựa trên kết quả nghiên cứu lý thuyết một cách có hệ thống	1	2	3	4	5
6	Kết quả được trình bày một cách logic và dễ hiểu, tất cả kết quả đều được phân tích và đánh giá thỏa đáng.	1	2	3	4	5
7	Trong phần kết luận, tác giả chỉ rõ sự khác biệt (nếu có) giữa kết quả đạt được và mục tiêu ban đầu đề ra đồng thời cung cấp lập luận để đề xuất hướng giải quyết có thể thực hiện trong tương lai.	1	2	3	4	5
Kỹ năng viết (10)						
8	Đồ án trình bày đúng mẫu quy định với cấu trúc các chương logic và đẹp mắt (bảng biểu, hình ảnh rõ ràng, có tiêu đề, được đánh số thứ tự và được giải thích hay đề cập đến trong đồ án, có căn lề, dấu cách sau dấu chấm, dấu phẩy v.v), có mở đầu chương và kết luận chương, có liệt kê tài liệu tham khảo và có trích dẫn đúng quy định	1	2	3	4	5

9	Kỹ năng viết xuất sắc (cấu trúc câu chuẩn, văn phong khoa học, lập luận logic và có cơ sở, từ vựng sử dụng phù hợp v.v.)	1	2	3	4	5
Thành tựu nghiên cứu khoa học (5) (chọn 1 trong 3 trường hợp)						
10a	Có bài báo khoa học được đăng hoặc chấp nhận đăng/đạt giải SVNC khoa học giải 3 cấp Viện trở lên/các giải thưởng khoa học (quốc tế/trong nước) từ giải 3 trở lên/ Có đăng ký bằng phát minh sáng chế	5				
10b	Được báo cáo tại hội đồng cấp Viện trong hội nghị sinh viên nghiên cứu khoa học nhưng không đạt giải từ giải 3 trở lên/Đạt giải khuyến khích trong các kỳ thi quốc gia và quốc tế khác về chuyên ngành như TI contest.	2				
10c	Không có thành tích về nghiên cứu khoa học	0				
Điểm tổng		**/50**				
Điểm tổng quy đổi về thang 10						

3. Nhận xét thêm của Thầy/Cô (*giảng viên hướng dẫn nhận xét về thái độ và tinh thần làm việc của sinh viên****)***

.....

.....

.....

.....

.....

.....

Ngày: / /2018

Người nhận xét

(Ký và ghi rõ họ tên)

Đánh giá quyển đồ án tốt nghiệp

(Dùng cho cán bộ phản biện)

Giảng viên đánh giá:.....

Họ và tên Sinh viên:..... MSSV:……...

Tên đồ án: ....

…….

Chọn các mức điểm phù hợp cho sinh viên trình bày theo các tiêu chí dưới đây:

Rất kém (1); Kém (2); Đạt (3); Giỏi (4); Xuất sắc (5)

Có sự kết hợp giữa lý thuyết và thực hành (20)						
1	Nêu rõ tính cấp thiết và quan trọng của đề tài, các vấn đề và các giả thuyết (bao gồm mục đích và tính phù hợp) cũng như phạm vi ứng dụng của đồ án	1	2	3	4	5
2	Cập nhật kết quả nghiên cứu gần đây nhất (trong nước/quốc tế)	1	2	3	4	5
3	Nêu rõ và chi tiết phương pháp nghiên cứu/giải quyết vấn đề	1	2	3	4	5
4	Có kết quả mô phỏng/thực nghiệm và trình bày rõ ràng kết quả đạt được	1	2	3	4	5
Có khả năng phân tích và đánh giá kết quả (15)						
5	Kế hoạch làm việc rõ ràng bao gồm mục tiêu và phương pháp thực hiện dựa trên kết quả nghiên cứu lý thuyết một cách có hệ thống	1	2	3	4	5
6	Kết quả được trình bày một cách logic và dễ hiểu, tất cả kết quả đều được phân tích và đánh giá thỏa đáng.	1	2	3	4	5
7	Trong phần kết luận, tác giả chỉ rõ sự khác biệt (nếu có) giữa kết quả đạt được và mục tiêu ban đầu đề ra đồng thời cung cấp lập luận để đề xuất hướng giải quyết có thể thực hiện trong tương lai.	1	2	3	4	5
Kỹ năng viết (10)						
8	Đồ án trình bày đúng mẫu quy định với cấu trúc các chương logic và đẹp mắt (bảng biểu, hình ảnh rõ ràng, có tiêu đề, được đánh số thứ tự và được giải thích hay đề cập đến trong đồ án, có căn lề, dấu cách sau dấu chấm, dấu phẩy v.v), có mở đầu chương và kết luận chương, có liệt kê tài liệu tham khảo và có trích dẫn đúng quy định	1	2	3	4	5

9	Kỹ năng viết xuất sắc (cấu trúc câu chuẩn, văn phong khoa học, lập luận logic và có cơ sở, từ vựng sử dụng phù hợp v.v.)	1	2	3	4	5
Thành tựu nghiên cứu khoa học (5) (chọn 1 trong 3 trường hợp)						
10a	Có bài báo khoa học được đăng hoặc chấp nhận đăng/đạt giải SVNC khoa học giải 3 cấp Viện trở lên/các giải thưởng khoa học (quốc tế/trong nước) từ giải 3 trở lên/ Có đăng ký bằng phát minh sáng chế	5				
10b	Được báo cáo tại hội đồng cấp Viện trong hội nghị sinh viên nghiên cứu khoa học nhưng không đạt giải từ giải 3 trở lên/Đạt giải khuyến khích trong các kỳ thi quốc gia và quốc tế khác về chuyên ngành như TI contest.	2				
10c	Không có thành tích về nghiên cứu khoa học	0				
Điểm tổng		**/50**				
Điểm tổng quy đổi về thang 10						

3. Nhận xét thêm của Thầy/Cô

.....

.....

.....

.....

.....

.....

Ngày: / /2018

Người nhận xét

(Ký và ghi rõ họ tên)

LỜI NÓI ĐẦU

Khoa học công nghệ ngày càng phát triển, Smarthome (nhà thông minh) cũng đang vận động theo chiều hướng tích cực. Smarthome dường như đã trở thành xu hướng trong những năm trở lại đây. Tuy nhiên, sự phát triển mạnh mẽ cũng đòi hỏi chất lượng của mỗi hệ thống Smarthome cần phải được nâng cao, các tính năng tiện ích cho người dùng cần được bổ sung, giúp người dùng có thể kiểm soát các thiết bị trong nhà một cách hiệu quả nhất. Một số hệ thống nhà thông minh ở Việt Nam hiện tại chưa cho phép người dùng điều khiển thiết bị sâu vào chức năng của nó, mới chỉ dừng lại ở mức cơ bản là bật hoặc tắt thiết bị. Đề tài ***Hệ thống điều khiển tivi bằng giọng nói trong nhà thông minh*** sẽ một phần đưa ra được giải pháp cho vấn đề này.

Hệ thống điều khiển tivi bằng giọng nói trong nhà thông minh là hệ thống được thiết kế nhằm mục đích giúp người dùng có thể ra lệnh bằng giọng nói để điều khiển chiếc tivi của mình ở bất cứ đâu có kết nối internet. Hệ thống có giao diện cho người dùng trên điện thoại đồng thời hệ thống cũng được tích hợp trợ lý thông minh Google Assistant, giúp người dùng không những ra lệnh qua ứng dụng điện thoại mà còn ra lệnh qua trợ lý ảo Google Assistant được tích hợp trên các thiết bị loa thông minh Google Home, Google Mini.

Bằng sự cố gắng nỗ lực của bản thân và đặc biệt là sự giúp đỡ tận tình, chu đáo của thầy TS. Nguyễn Tiến Hòa, em đã hoàn thành đồ án đúng thời hạn. Do thời gian làm đồ án có hạn và trình độ còn nhiều hạn chế nên không thể tránh khỏi những thiếu sót. Em rất mong nhận được sự đóng góp ý kiến của các thầy cô để bài đồ án này hoàn thiện hơn nữa. Em xin chân thành cảm ơn thầy TS. Nguyễn Tiến Hòa, các thầy cô giáo trong Viện Điện tử truyền thông đã tạo điều kiện giúp đỡ em trong thời gian qua.

Hà Nội, ngày 30 tháng 5 năm 2018

Sinh viên thực hiện

Trần Ngọc Tiến

TÓM TẮT ĐỒ ÁN

Đồ án ***Hệ thống điều khiển tivi bằng giọng nói trong nhà thông minh*** được thực hiện với mục đích là đưa ra một mô hình nhà thông minh mà trong đó người dùng có thể ra lệnh điều khiển thiết thiết bị điện trong nhà, cụ thể ở đây là chiếc tivi thông qua ứng dụng trên điện thoại hoặc qua trợ lý ảo Google Assistant.

Hệ thống điều khiển tivi bằng giọng nói trong nhà thông minh được thiết kế theo mô hình Client – Server, giao thức giữa các Client và Server sử dụng là WebSocket, giúp cho các thiết bị luôn được kết nối với nhau theo thời gian thực. Sơ đồ khối của hệ thống bao gồm ba khối chính: khối nhận lệnh từ người dùng, khối Server và khối thực thi lệnh. Yêu cầu của người dùng được gửi đến Server xử lý và được Server truyền đến khối thực thi để thực hiện lệnh tương ứng.

Đồ án sẽ trình bày các bước để thiết kế nên từng khối của hệ thống, bao gồm thiết kế Server, thiết kế ứng dụng trên điện thoại di động, thiết kế ứng dụng trên nền tảng Google Assistant, thiết kế khối thực thi trên nền tảng Raspberry Pi.

Phần thiết kế Server trình bày các chức năng của Server, phương pháp thiết kế một Server sử dụng Nodejs. Server sẽ quản lý các kết nối đến từ các Client theo hướng sự kiện với khả năng xử lý không đồng bộ, thời gian đáp ứng nhanh.

Thiết kế ứng dụng trên điện thoại và ứng dụng trên nền tảng Google Assistant sẽ trình bày chi tiết các bước để xây dựng nên ứng dụng, phương pháp kết nối với Server, đồng thời trình bày kỹ thuật xử lý yêu cầu của người dùng thông qua giọng nói.

Thiết kế khối thực thi trên nền tảng Raspberry sẽ tập trung vào cách Raspberry xử lý dữ liệu nhận được từ Server, phương pháp điều khiển các module ngoại vi đặc biệt là module hồng ngoại để thực hiện lệnh điều khiển tivi như đúng như yêu cầu của người dùng.

ABSTRACT

Home Automated TV Voice Control System is designed to provide a smart home model in which users can command home electrical appliances, specifically in This is the TV through the phone app or through the Google Assistant virtual assistant.

Home Automated TV Voice Control System designed as a client-server model, and the protocol between the client and the server is WebSocket, which allows devices to stay connected in real time. The system block diagram consists of three main blocks: the command block from the user, the server block, and the execution block. User requests are sent to the processing server and sent to the execution unit by the server to execute the corresponding command.

The project will describe the steps to design each block of the system, including Server Design, Mobile Application Design, Application Design on the Google Assistant Platform, Raspberry Pi.

The server design shows the functions of the server, the method of designing a server using Nodejs. Server will manage incoming connections from event-oriented clients with asynchronous processing capability and fast response time.

The application design on phones and applications on the Google Assistant platform will detail the steps for building the application, the method of connecting to the server, and demonstrating the processing techniques required by users by voice.

Raspberry-based execution block design will focus on how Raspberry handles data received from the server, the method of controlling peripheral modules, especially the infrared module, to perform TV control commands as required. of the user.

MỤC LỤC

DANH MỤC HÌNH VẼ

MỞ ĐẦU

Đề tài ***Hệ thống điều khiển tivi bằng giọng nói trong nhà thông minh*** giải quyết một số hạn chế trong các hệ thống nhà thông minh hiện nay đó là: điều khiển sâu vào chức năng của thiết bị, hỗ trợ ra lệnh bằng giọng nói tiếng Việt, tích hợp một trợ lý ảo thông minh. Nội dung của đồ án bao gồm các chương sau:

Chương 1 - Cơ sở lý thuyết: Chương này giới thiệu cơ sở lý thuyết của các công nghệ mà đề tài sử dụng như nền tảng lập trình Nodejs, máy tính nhúng Raspberry Pi,…

Chương 2 - Sơ đồ khối hệ thống: Giới thiệu thiết kế sơ đồ khối cho toàn bộ hệ thống, bao gồm các phương thức kết nối, cách hoạt động của hệ thống.

Chương 3 - Thiết kế Server: Trình bày chi tiết cách thiết kế Server của hệ thống, phương pháp xử lý sự kiện từ các Client gửi tới.

Chương 4 - Thiết kế ứng dụng trên điện thoại: Trình bày cách thiết kế ứng dụng người dùng trên điện thoại Android bao gồm giao diện, phương thức kết nối đến Server,…

Chương 5 - Thiết kế ứng dụng trên Google Assistant: Trình bày phương pháp tạo một trợ lý ảo riêng chạy trên nền Google Assistant với một số mẫu câu hội thoại cơ bản.

Chương 6 - Thiết kế khối thực thi trên Raspberry: Xử lý dữ liệu từ Server và kết nối các module ngoại vi để điều khiển tivi.

Chương 7 - Kết luận chung: Đưa ra kết luận, những việc chưa làm được và phương hướng phát triển đề tài.

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

1.1 Nodejs

Node.js là một hệ thống được thiết kế để viết các ứng dụng internet có khả năng mở rộng, đặc biệt là máy chủ web. Chương trình được viết bằng JavaScript, sử dụng kỹ thật điều khiển theo sự kiện, nhập/xuất không đồng bộ để tối thiểu tổng chi phí và tối đại khả năng mở rộng. Node.js bao gồm có V8 JavaScript engine của Google, libUV, và vài thư viện khác [1].

Node.js được tạo bởi Ryan Dahl từ năm 2009, và phát triển dưới sự bảo trợ của Joyent. Mục tiêu ban đầu của Dahl là làm cho trang web có khả năng push như trong một số ứng dụng web như Gmail. Sau khi thử với vài ngôn ngữ Dahl chọn Javascript vì một API Nhập/Xuất không đầy đủ. Điều này cho phép anh có thể định nghĩa một quy ước Nhập/Xuất điều khiển theo sự kiện, non-blocking [1].

Vài môi trường tương tự được viết trong các ngôn ngữ khác bao gồm Twisted cho Python, Perl Object Environment cho Perl, libevent cho C và EventMachine cho Ruby. Khác với hầu hết các chương trình Javascript, Nodejs không chạy trên một trình duyệt mà chạy trên Server. Node.js sử dụng nhiều chi tiết kỹ thuật của CommonJs. Nó cung cấp một môi trường REPL cho kiểm thử tương tác.

Node.js là một ngôn ngữ mới, xây dựng thuần túy bằng javascript. Đây là một điểm lợi thế của Node.js để lập trình web-socket:

Thứ nhất: javascript là ngôn ngữ lập trình hướng sự kiện, mà trong lập trình thời gian thực, cách tiếp cận bằng lập trình sự kiện là cách tiếp cận khôn ngoan nhất.

Thứ hai: Node.js chạy non-blocking việc hệ thống không phải tạm ngừng để xử lý xong một request sẽ giúp cho server trả lời client gần như ngay tức thì.

Thứ ba: lập trình socket yêu cầu bạn phải xây dựng được mô hình lắng nghe – trả lời từ cả 2 bên. Nói khác đi, vai trò của client và server phải tương đương nhau, mà

client thì chạy bằng javascript, nên nếu server cũng chạy bằng javascript nữa, thì việc lập trình sẽ dễ dàng và thân thiện hơn.

1.2 Giao thức WebSocket

WebSoket là công nghệ hỗ trợ giao tiếp hai chiều giữa client và server bằng cách sử dụng một TCP socket để tạo một kết nối hiệu quả và ít tốn kém. Mặc dù được thiết kế để chuyên sử dụng cho các ứng dụng web, lập trình viên vẫn có thể đưa chúng vào bất kì loại ứng dụng nào [2].

WebSockets mới xuất hiện trong HTML5, là một kỹ thuật Reverse Ajax. WebSockets cho phép các kênh giao tiếp song song hai chiều và hiện đã được hỗ trợ trong nhiều trình duyệt (Firefox, Google Chrome và Safari). Kết nối được mở thông qua một HTTP request (yêu cầu HTTP), được gọi là liên kết WebSockets với những header đặc biệt. Kết nối được duy trì để bạn có thể viết và nhận dữ liệu bằng JavaScript như khi bạn đang sử dụng một TCP socket đơn thuần [2].

Dữ liệu truyền tải thông qua giao thức HTTP (thường dùng với kĩ thuật Ajax) chứa nhiều dữ liệu không cần thiết trong phần header. Một header request/response của HTTP có kích thước khoảng 871 byte, trong khi với WebSocket, kích thước này chỉ là 2 byte (sau khi đã kết nối). Vậy giả sử bạn làm một ứng dụng game có thể tới 10,000 người chơi đăng nhập cùng lúc, và mỗi giây họ sẽ gửi/nhận dữ liệu từ server. Hãy so sánh lượng dữ liệu header mà giao thức HTTP và WebSocket trong mỗi giây:

HTTP: 871 x 10,000 = 8,710,000 bytes = 69,680,000 bits per second (66 Mbps)

WebSocket: 2 x 10,000 = 20,000 bytes = 160,000 bits per second (0.153 Kbps)
Như bạn thấy chỉ riêng phần header thôi cũng đã chiếm một phần lưu lượng đáng kể với giao thức HTTP truyền thống.

❖ Giao thức bắt tay của WebSocket

Để thực hiện kết nối, client phải gửi một WebSocket handshake request đến server. Server sẽ gửi trả lại WebSocket handshake response như bên dưới:

Client request:

```
GET /mychat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Protocol: chat

Sec-WebSocket-Version: 13

Origin: http://example.com
```

Server response:

```
HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
```

Để xác nhận việc kết nối, client sẽ gửi một giá trị Sec-WebSocket-Key được mã hóa bằng Based64 đến server. Sau đó bên server sẽ thực hiện: – Nối thêm chuỗi cố định là “258EAFA5-E914-47DA-95CA-C5AB0DC85B11″ vào Sec-WebSocket-Key để được chuỗi mới là “x3JJHMbDL1EzLkh9GBhXDw==258EAFA5-E914-47DA-95CA-C5AB0DC85B11″. – Thực hiện mã hóa SHA-1 chuỗi trên để được “1d29ab734b0c9585240069a6e4e3e91b61da1969″. – Mã hóa kết quả vừa nhận được bằng Base64 để được “HSmrc0sMlYUkAGmm5OPpG2HaGWk=”. – Gửi response lại client kèm với giá trị Sec-WebSocket-Accept chính là chuỗi kết quả vừa tạo ra.

Client sẽ kiểm tra status code (phải bằng 101) và Sec-WebSocket-Accept xem có đúng với kết quả mong đợi không và thực hiện kết nối.

❖ **Ưu điểm:**

WebSockets cung cấp khả năng giao tiếp hai chiều mạnh mẽ, có độ trễ thấp và dễ xử lý lỗi. Không cần phải có nhiều kết nối như phương pháp Comet long-polling và cũng không có những nhược điểm như Comet streaming.

API cũng rất dễ sử dụng trực tiếp mà không cần bất kỳ các tầng bổ sung nào, so với Comet, thường đòi hỏi một thư viện tốt để xử lý kết nối lại, thời gian chờ timeout, các Ajax request (yêu cầu Ajax), các tin báo nhận và các dạng truyền tải tùy chọn khác nhau (Ajax long-polling và jsonp polling).

❖ **Nhược điểm:**

Nó là một đặc tả mới của HTML5, nên nó vẫn chưa được tất cả các trình duyệt hỗ trợ. Không có phạm vi yêu cầu nào. Do WebSocket là một TCP socket chứ không phải là HTTP request, nên không dễ sử dụng các dịch vụ có phạm vi-yêu cầu, như SessionInViewFilter của Hibernate. Hibernate là một framework kinh điển cung cấp một bộ lọc xung quanh một HTTP request. Khi bắt đầu một request, nó sẽ thiết lập một contest (chứa các transaction và liên kết JDBC) được ràng buộc với luồng request. Khi request đó kết thúc, bộ lọc hủy bỏ contest này.

1.3 Thư viện Socket.IO trong Nodejs

Socket.IO là một thư viện javascript có mục đích tạo ra các ứng dụng realtime trên trình duyệt cũng như thiết bị di động. Việc sử dụng thư viện này cũng rất đơn giản và giống nhau ở cả server lẫn client.

Server tạo một đối tượng socket bằng phương thức ***listen(port).*** Phương thức này chờ đợi một yêu cầu kết nối từ client.

Client kết nối đến server bằng phương thức ***connect(url,{port: server_port}).***

Socket.IO cung cấp 3 event chính là ***connect, message và disconnect***. Chúng được kích hoạt khi client/server:

- ***connect:*** tạo kết nối
- ***message:*** nhận được thông điệp
- ***disconnect:*** ngắt kết nối

Ví dụ: Khai báo cho socket nhận một sự kiện "message"

```
socket.on("message", function(msg){
// console.log("Received: "+ msg);
});
```

Socket.IO có thể gửi và nhận các event tự tạo với phương thức ***emit().*** Hai phía gửi và nhận phải biết được tên của event đó để thực hiện giao tiếp:

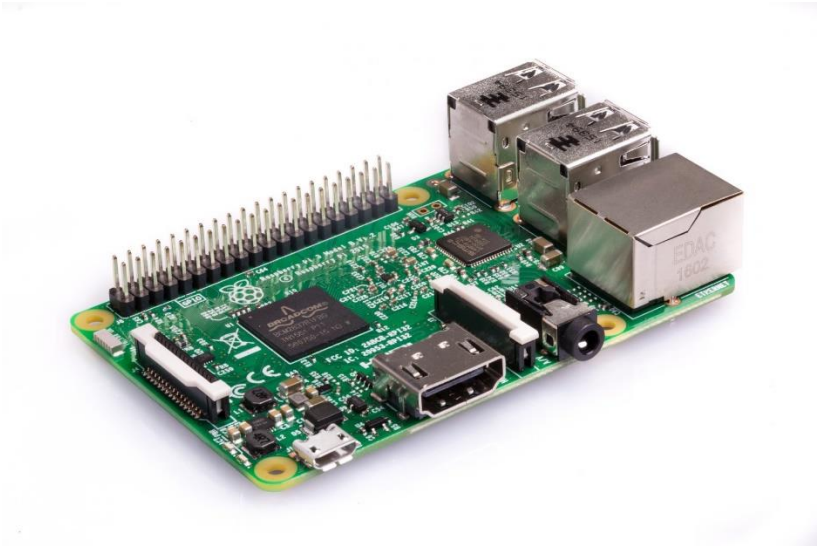
Ví dụ:

```
// client gửi một dòng message "welcome" lên event "hello"
socket.emit("hello", {msg: "welcome"});
// Server nhận sự kiện event đưa lên
socket.on("hello", function (data) {
     console.log(data);});
```

1.4 Bo nhúng Raspberry Pi

Raspberry Pi là chiếc máy tính kích thước nhỏ được tích hợp nhiều phần cứng mạnh mẽ đủ khả năng chạy hệ điều hành và cài đặt được nhiều ứng dụng trên nó. Với giá chỉ vài chục USD, Raspberry hiện đang là mini computer nổi bật nhất hiện nay. Ban đầu, tổ chức Raspberry Pi Foundation phát triển dự án Raspberry với mục tiêu chính là giảng dạy máy tính cho trẻ em và tạo ra một công cụ giá rẻ (chỉ vài chục USD) để sinh viên nghiên cứu học tập. Tuy nhiên, sau khi xuất hiện, Raspberry Pi được cộng đồng đánh giá cao về tính ứng dụng với phần cứng được hỗ trợ tốt, Pi đã nhanh chóng

phát triển một cách rộng rãi. Pi phù hợp cho những ứng dụng cần khả năng xử lý mạnh mẽ, đa nhiệm hoặc giải trí và đặc biệt cần chi phí thấp. Hiện nay đã có hàng ngàn ứng dụng đa dạng được cài đặt trên Rasberry Pi [3].



Hình 1.1 Raspberry Pi 3 Model B

Raspberry Pi có hai phiên bản, Model A có giá 25$ và Model B có giá 35$. Model B như hình trên thông dụng hơn cả. Model B bao gồm những phần cứng và những cổng giao diện:

- SoC 700MHz với 512MB RAM .
- 1 cổng HDMI cho đầu ra âm thanh / video số .
- 1 cổng video RCA cho đầu ra video Analog .
- Jack Headphone Stereo 3.5mm cho đầu ra âm thanh Analog .
- 02 cổng USB .
- 01 đầu đọc thẻ nhớ SD để tải hệ điều hành .
- 01 cổng Ethernet LAN.
- 01 giao diện GPIO (General Purpose Input/Output) .

Model A cũng gần tương tự như Model B nhưng có sự thay đổi như sau:

- 1 cổng USB

- Không có cổng Ethernet vì thế người dùng phải thêm Adapter USB Wi-Fi hoặc Ethernet nếu cần kết nối mạng .
- 256MB RAM .

Raspberry Pi chạy hệ điều hành dựa trên nhân Linux. Raspbian - một phiên bản dựa trên Debian đã được tối ưu cho phần cứng của Pi là hệ điều hành được tổ chức Raspberry Pi Foundation đề nghị sử dụng.

1.5 Thư viện LIRC

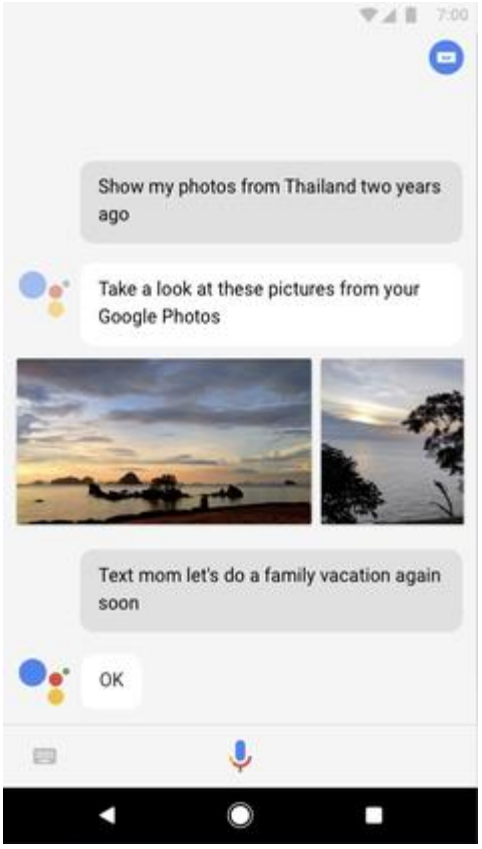
Lirc là viết tắt của "Linux Infrared Remote Control". Đây là một gói phần mềm sẽ cho phép giải mã và lưu tín hiệu hồng ngoại để sử dụng sau này. Thư viện hỗ trợ cho các thiết bị chạy hệ điều hành Linux. Lirc cho phép thu các tín hiệu hồng ngoại và lưu chúng lại vào một file, đồng thời cho phép chúng ta phát lại các tín hiệu ấy qua module phát. Trong Lirc có 3 chức năng chính để phân tích các tín hiệu hồng ngoại:

- **mode2 :** đầu ra pulse/space của tín hiệu hồng ngoại.
- **irrecord :** dùng để ghi tín hiệu hồng ngoại vào file .conf dùng cho việc phát lại sau này.
- **irsend**: gửi tín hiệu hồng ngoại từ dòng lệnh. Tín hiệu có thể được gửi một lần hoặc nhiều lần.

1.6 Trợ lý Google Assistant

1.6.1 Giới thiệu

Google Assistant là một trợ lý cá nhân ảo được phát triển bởi Google và được giới thiệu tại hội nghị nhà phát triển của hãng vào tháng 5 năm 2016. Không giống như Google Now, Google Assistant có thể tham gia các cuộc trò chuyện hai chiều.



Hình 1.2 Trợ lý cá nhân ảo Google Assistant

Assistant ban đầu được đưa vào ứng dụng nhắn tin Google Allo, và loa thông minh Google Home. Sau một thời gian chỉ có mặt trên hai chiếc điện thoại thông minh Pixel và Pixel XL của hãng, Google bắt đầu triển khai Assistant trên các thiết bị Android khác vào tháng 2 năm 2017, bao gồm cả các điện thoại thông minh bên thứ ba và các thiết bị Android Wear, và được phát hành dưới dạng ứng dụng riêng biệt trên iOS vào tháng 5. Cùng với sự ra mắt một bộ phát triển phần mềm vào tháng 4 năm 2017, Assistant đã và đang được tiếp tục mở rộng hỗ trợ cho một lượng lớn thiết bị, bao gồm cả xe hơi và các thiết bị nhà thông minh. Các chức năng của Assistant cũng có thể được bổ sung bởi các nhà phát triển bên thứ ba [4].

Người dùng chủ yếu có thể tương tác với Google Assistant qua giọng nói tự nhiên, hoặc có thể nhập qua bàn phím. Các chức năng cơ bản của nó cũng tương tự như Google Now, như tìm kiếm trên Internet, đặt sự kiện trên lịch và báo thức, điều chỉnh

cài đặt phần cứng trên thiết bị người dùng và hiển thị thông tin từ tài khoản Google của người dùng. Google cũng bổ sung các tính năng khác cho Assistant bao gồm khả năng nhận diện vật thể và thu thập thông tin về vật thể thông qua máy ảnh của thiết bị, cùng với việc hỗ trợ mua sản phẩm và chuyển tiền.

Vào tháng 12 năm 2016, Google khởi động "Actions on Google", một nền tảng nhà phát triển cho Google Assistant. Actions on Google cải thiện trải nghiệm người dùng với Assistant bằng cách cho phép các nhà phát triển đưa dịch vụ của mình vào Assistant. Vào tháng 3 năm 2017, Google bổ sung thêm các công cụ phát triển mới cho Actions on Google để hỗ trợ việc tạo các trò chơi trên Google Assistant. Ban đầu chỉ giới hạn trong chiếc loa thông minh Google Home, Actions on Google được ra mắt cho các thiết bị Android và iOS vào tháng 5 năm 2017, và cùng lúc này hãng cũng giới thiệu một khu vực giới thiệu các sản phẩm và dịch vụ tương thích với Assistant [4].

Vào tháng 4 năm 2017, Google phát hành một bộ phát triển phần mềm (SDK), cho phép các nhà phát triển bên thứ ba có thể tự xây dựng phần cứng tương thích với Google Assistant. Nó cũng đã được tích hợp vào Raspberry Pi, những chiếc xe hơi từ Audi và Volvo, và các thiết bị nhà thông minh, bao gồm tủ lạnh, máy giặt và lò nướng, từ các công ty như iRobot, LG, General Electric, và D-Link.

1.6.2 Google Home

Google Home là một thiết bị loa thông minh được phát triển bởi Google . Thiết bị đầu tiên được công bố vào tháng 5 năm 2016 và được phát hành tại Mỹ vào tháng 11 năm 2016, với các bản phát hành tiếp theo trên toàn cầu trong suốt năm 2017 [5].

Hình 1.3 Loa thông minh Google home

Loa Google Home cho phép người dùng nói lệnh thoại để tương tác với các dịch vụ thông qua trợ lý cá nhân Google Assistant. Một số lượng lớn các dịch vụ, cả trong nhà và bên thứ ba, được tích hợp, cho phép người dùng nghe nhạc, kiểm soát phát lại video hoặc ảnh hoặc nhận cập nhật tin tức hoàn toàn bằng giọng nói. Các thiết bị Google Home cũng có hỗ trợ tích hợp cho tự động hóa tại nhà, cho phép người dùng điều khiển thiết bị gia dụng thông minh bằng giọng nói của họ. Nhiều thiết bị Google Home có thể được đặt trong các phòng khác nhau trong nhà để phát nhạc đồng bộ. Bản cập nhật vào tháng 4 năm 2017 đã hỗ trợ đa người dùng, cho phép thiết bị phân biệt giữa tối đa sáu người bằng giọng nói. Vào tháng 5 năm 2017, Google đã công bố nhiều bản cập nhật cho chức năng của Google Home, bao gồm: gọi điện thoại rảnh tay miễn phí tại Hoa Kỳ và Canada; chủ động cập nhật trước các sự kiện đã lên lịch; câu trả lời trực quan trên thiết bị di động hoặc TV có hỗ trợ Chromecast ; Phát trực tuyến âm thanh Bluetooth ; và khả năng thêm lời nhắc và cuộc hẹn trên lịch [5].

1.6.3 Hỗ trợ từ nhà phát triển

1.6.3.1 Actions on Google

Actions on Google là một chương trình dành cho nhà phát triển chạy thông qua trợ lý ảo Google Assistant trên các thiết bị như loa thông minh Google Home và điện thoại thông minh, hay nói cách khác Actions on Google cho phép lập trình viên xây dựng các ứng dụng của mình trên trợ lý ảo Google Assistant.

Lập trình viên có thể xây dựng hai loại Actions. Actions thứ nhất là các Actions trực tiếp như: yêu cầu cung cấp các thông tin và nhận được câu trả lời, yêu cầu tắt đèn thì đèn tắt, yêu cầu phát một bài hát và nhạc đó phát. Actions thứ hai là các Actions đối thoại lại nhiều hơn như các cuộc trò chuyện. Và để xây dựng các đoạn hội thoại thì Google cung cấp một công cụ cho lập trình viên là DialogFlow [6].

Hình 1.4 Các ứng dụng của Action on Google

1.6.3.2 DialogFlow

Dialogflow (tiền thân là API.AI) là một dịch vụ do Google cung cấp nhằm giúp các lập trình viên dễ dàng hơn khi lập trình các sản phẩm có giao tiếp giữa người dùng với sản phẩm thông qua các đoạn hội thoại bằng văn bản (text) hoặc giọng nói (voice).

Dialogflow sử dụng trí tuệ nhân tạo (AI) giúp phân tích ngôn ngữ tự nhiên để hiểu được những gì người dùng đưa vào. Hiện Dialogflow có 2 phiên bản:

- Standard: Hoàn toàn miễn phí để sử dụng.
- Enterprise: Trả phí để sử dụng.

DialogFlow sử dụng các khái niệm sau để xây dựng nên các cuộc hội thoại:

❖ **Intent**

Một intent là đại diện cho một ánh xạ giữa những gì người dùng đưa vào, và hành động sẽ được thực hiện bởi phần mềm.

Ví dụ bạn đang cần làm một Chatbot dự báo thời tiết, giả sử khi người dùng hỏi với Chatbot:

- Ngày mai ở Hà Nội trời có mưa không?
- Thứ 6 này đến Đà Nẵng thì có cần mang áo ấm không?

Với cả hai câu hỏi này, người dùng đều đang mong muốn Chatbot trả lời thông tin dự báo về thời tiết. Như vậy, khi lập trình Chatbot, bạn chỉ cần một hành động cho cả hai câu hỏi, hay bạn cần tạo một intent cho chúng.

Nói một cách khác, một intent là một tập những gì người dùng nói mà chúng có cùng một ý nghĩa.

❖ **Entities**

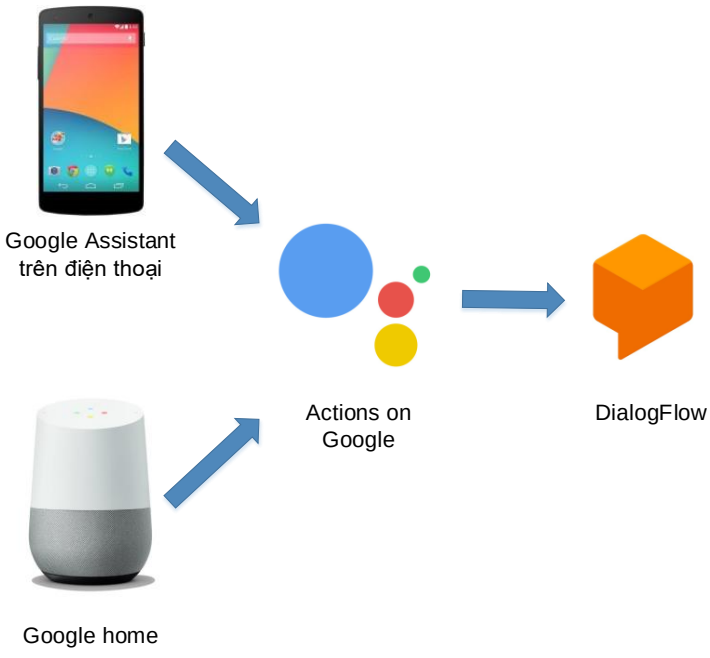
Entities là những công cụ được sử dụng để trích xuất các giá trị của tham số từ ngôn ngữ tự nhiên. Bất kỳ những gì mà bạn muốn biết từ nội dung của người dùng đều sẽ có một Entities tương ứng.

Quay lại với ví dụ ở trên, "Ngày mai" hay "Thứ 6 này" sẽ có một Entities tương ứng là **Thời gian**. Hà Nội, Đà Nẵng, New York, … sẽ là Entities **Vị trí.**

❖ **Agent**

Agent là khái niệm được dùng để đại diện cho một module NLU (Natural Language Understanding – Phân tích để hiểu ngôn ngữ tự nhiên).

Agent giúp bạn phân tích những những gì người dùng đưa vào (text hoặc voice) để chuyển thành những dữ liệu mà bạn có thể xử lý được bằng lập trình. Bạn sẽ sử dụng một Agent để quản lý các đoạn hội thoại thông qua Intent, Entities.



Hình 1.5 Mô hình phát triển cho Google Assistant

CHƯƠNG 2. SƠ ĐỒ KHỐI HỆ THỐNG

2.1 Tổng quan

Hiện nay, IoT đang ngày càng phát triển trong cuộc sống, một trong những ứng dụng to lớn của IoT đấy là lĩnh vực nhà thông minh. Nhà thông minh cho phép người dùng có thể điều khiển được thiết bị điện trong nhà như bật quạt, bật nóng lạnh, bật tivi,...

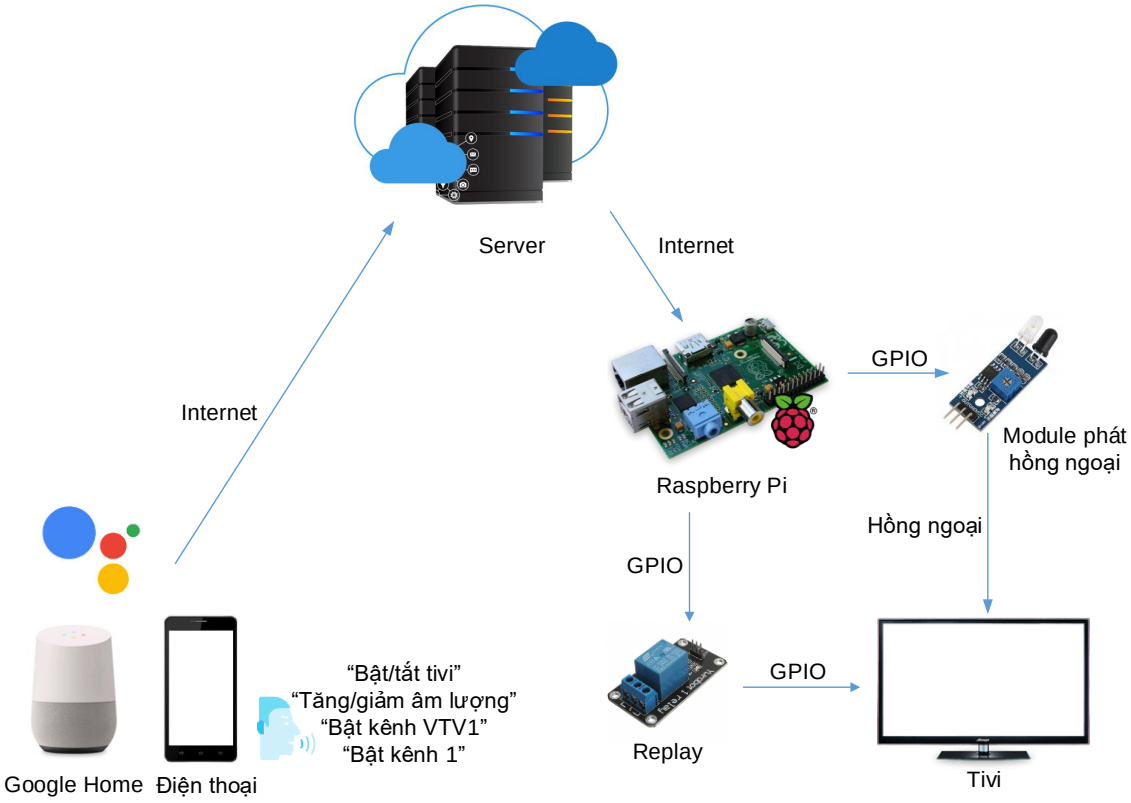
Ý tưởng xây dựng hệ thống điều khiển tivi bằng giọng nói tiếng Việt cũng xuất phát từ sự phát triển của IoT và cụ thể là lĩnh vực nhà thông minh. Hệ thống cho phép người dùng điều khiển các chức năng của tivi bằng giọng nói tiếng Việt thông qua ứng dụng điện thoại hoặc các thiết bị chạy nền tảng trợ lý ảo Google Assistant.

Xuất phát từ ý tưởng đó, để xây dựng nên hệ thống cần có một số yêu cầu:

- Có ứng dụng điều khiển bằng giọng nói tiếng Việt trên điện thoại
- Hỗ trợ nền tảng trợ lý ảo Google Assistant
- Điều khiển chức năng bật/tắt tivi, chuyển kênh và các chức năng khác của tivi.

2.2 Sơ đồ khối

Hệ thống hỗ trợ người dùng qua nền tảng Internet, chính vì vậy việc thiết kế hệ thống theo mô hình Server – Client là tối ưu. Server là trung tâm kết nối các Client lại với nhau, đồng thời cũng là nơi xử lý dữ liệu. Các Client ở đây là ứng dụng trên điện thoại, các thiết bị chạy Google Assistant, và Client thực thi lệnh từ Server. Dưới đây là sơ đồ khối của của hệ thống:



Hình 2.1 Sơ đồ khối của hệ thống

Nhìn vào sơ đồ ta thấy, hệ thống bao gồm một Server, các client kết nối tới Server thông qua môi trường Internet. Điện thoại, Google Assistant là các client nhận lệnh từ người dùng. Raspberry Pi là client có chức năng thực thi các lệnh từ Server, cũng là thiết bị điều khiển trực tiếp đến tivi.

Giao thức kết nối giữa Server và Client sử dụng chủ yếu là WebSocket, đảm bảo tính ổn định và thời gian đáp ứng nhanh. Giao thức kết nối giữa Google Assistant với Server là Http, một giao thức truyền thống. Do đó, Server được thiết kế cần hỗ trợ cả hai giao thức này. Công nghệ được sử dụng để thiết kế Server là Nodejs với khả năng xử lý nhiều sự kiện một lúc, hỗ trợ nhiều giao thức kết nối,…

Phần thiết bị thực thi ở đây sử dụng là Raspberry Pi. Với khả năng xử lý mạnh mẽ, kết nối được Internet và đặc biệt có thể giao tiếp với các module ngoại vi khác thông qua các GPIO. Module thu hồng ngoại có chức năng thu tín hiệu từ remote tivi

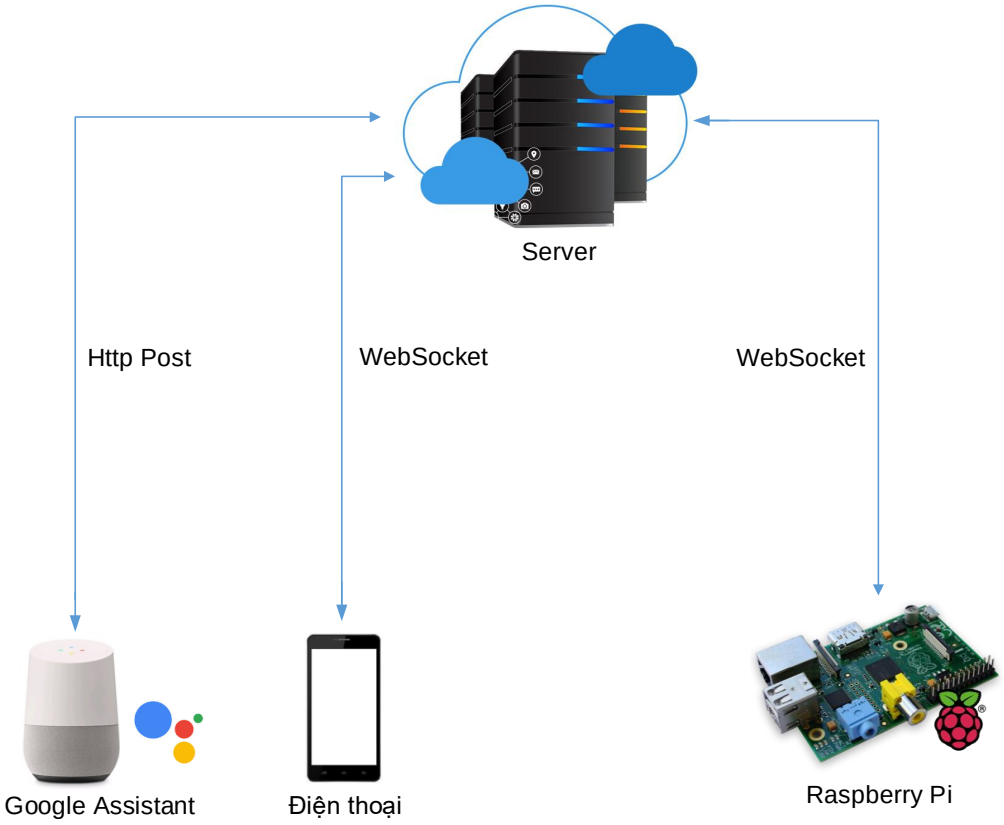
để thư viện LIRC trên Raspberry lưu lại các tín hiệu đó phục vụ cho việc phát sau này. Module phát được điều khiển bởi thư viện LIRC sẽ phát ra đúng tín hiệu như ở remote tivi đã lưu trước đó. Module relay có nhiệm vụ điều khiển nguồn điện cấp cho tivi.

CHƯƠNG 3. THIẾT KẾ SERVER

3.1 Sơ đồ khối

Server được thiết kế có chức năng kết nối các thiết bị ra lệnh của người dùng là ứng dụng trên điện thoại và Google Assistant với khối thực thi trên Raspberry Pi. Phương thức kết nối giữa Server và Client là WebSocket và Http. Cụ thể, Http dùng cho kết nối giữa Google Assistant với Server, WebSocket dùng cho kết nối giữa điện thoại và Raspberry với Server.

Server được viết trên ngôn ngữ Nodejs, sử dụng kỹ thật điều khiển theo sự kiện, nhập/xuất không đồng bộ. Vì Node.js chạy non-blocking nên việc hệ thống không phải tạm ngừng để xử lý xong một request sẽ giúp cho server trả lời client gần như ngay tức thì, đây là một điểm mạnh so với kết nối Http truyền thống.



Hình 3.1 Sơ đồ khối Server

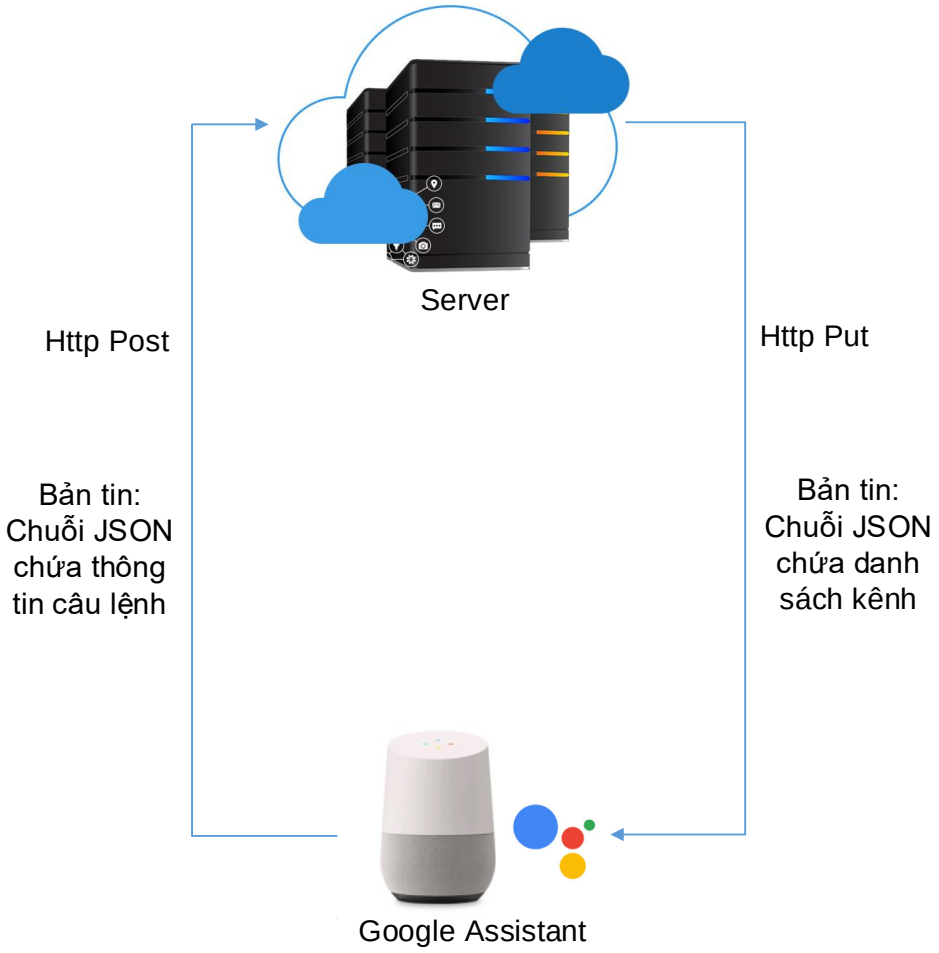
Kết nối giữa Google Assistant với Server là kết nối Http hai chiều, hiện tại Google chỉ hỗ trợ phương thức kết nối duy nhất là Http. Google Assistant khi nhận được lệnh của người dùng sẽ gửi một chuỗi JSON chứa thông tin của câu lệnh tới Server theo phương thức POST trong Http. Khi người dùng cập nhật hoặc thêm mới kênh thì dữ liệu sẽ được gửi lên Server và Server sẽ gửi dữ liệu kênh đó cho Google Assistant học.

Kết nối giữa ứng dụng trên điện thoại với Server là kết nối WebSocket hai chiều, bản tin giao tiếp là chuỗi JSON. Với chiều từ ứng dụng tới Server, chuỗi JSON chứa thông tin là các lệnh điều khiển từ người dùng như thông tin kênh, tăng giảm âm lượng,..Với chiều từ Server tới ứng dụng, chuỗi JSON chứa thông tin trạng thái nguồn điện của tivi, thông tin này được Raspberry gửi lên cho Server.

Tương tự, kết nối giữa Raspberry với Server cũng là kết nối WebSocket hai chiều, bản tin giao tiếp là chuỗi JSON. Với chiều từ Server tới Raspberry, chuỗi JSON chứa thông tin kênh hoặc thông tin điều khiển tivi như tăng giảm âm lượng,…Với chiều ngược lại, chuỗi JSON chứa thông tin trạng thái nguồn điện của tivi.

3.2 Thiết kế

3.2.1 Google Assistant với Server



Hình 3.2 Kết nối giữa Google Assistant với Server

❖ **Với chiều kết nối từ Google Assistant đến Server:**

Khi người dùng giao tiếp với Google Assistant, với mỗi câu lệnh của người dùng thì Google Assistant sẽ gửi đến Server một bản tin là chuỗi JSON thông qua phương thức POST của giao thức Http. Chuỗi JSON này chứa thông tin của cuộc hội thoại và thông tin các Entities trong câu nói của người dùng. Ví dụ:

- **Người dùng nói:** *Turn on 25 channel*

Với câu nói trên thì Entities ở đây là: 25, là số kênh người dùng muốn bật. Chuỗi JSON mà Google Assistant gửi đến Server sẽ là:


```
{
  "id": "911edc5f-892b-474e-a5d6-f68c07493a8e",
  "timestamp": "2018-05-18T10:41:12.433Z",
  "lang": "en",
  "result": {
    "source": "agent",
    "resolvedQuery": "turn on 25 channel",
    "action": "",
    "actionIncomplete": false,
    "parameters": {
      "controltv": "",
      "channel": "25"
    },
    "contexts": [],
    "metadata": {
      "intentId": "d80c7ca1-934d-4ab3-b1d5-78988af11469",
      "webhookUsed": "true",
      "webhookForSlotFillingUsed": "false",
      "webhookResponseTime": 10059,
      "intentName": "Control"
    },
    "fulfillment": {
      "speech": "OK, 25 channel is on now",
      "messages": [
```

```
        {
          "type": "simple_response",
          "platform": "google",
          "textToSpeech": "OK, 25 channel is on now"
        },
        {
          "type": 0,
          "speech": "OK, 25 channel is on now"
        }
      ]
    },
    "score": 1
  },
  "status": {
    "code": 206,
    "errorType": "partial_content",
    "errorDetails": "Webhook call failed. Error: Request timeout."
  },
  "sessionId": "2f611008-450d-4bc8-84e9-7540d49a770d"
}
```


Trong chuỗi JSON trên có chứa nhiều thông tin liên quan đến cuộc hội thoại giữa người dùng và Google Assistant, tuy nhiên chúng ta chỉ cần quan tâm đến thông số ***result:***


```
"result": {
    "source": "agent",
    "resolvedQuery": "turn on 25 channel",
```

```
    "action": "",
    "actionIncomplete": false,
    "parameters": {
      "controltv": "",
      "channel": "25"
    },
```

Trong ***result*** chứa thông tin quan trọng đó là các Entities trong câu lệnh của người dùng, cụ thể đó là thông số ***parameters:***

```
"parameters": {
      "controltv": "",
      "channel": "25"
    },
```

Dựa vào thông số này, chúng ta có thể phân tích được yêu cầu của người dùng là gì. Đối với câu lệnh: ***"Turn on 25 channel"*** thì Entities ở đây là Channel Entities với giá trị là 25.

Google Assistant gửi chuỗi JSON đến Server thông qua phương thức POST của giao thức Http, chính vì vậy Server cần phải lắng nghe sự kiện do Google Assistant gửi đến để có thể bắt được chuỗi JSON và trong Nodejs, thư viện **express** cho phép chúng ta làm điều đó.

Khai báo thư viện express trong Nodejs:

```
var express = require("express");
var exp = express();
```

Để bắt các sự kiện do phương thức POST của Http gửi tới và đọc thông số trong chuỗi JSON ta dùng:

```
exp.use(bodyParser.json({extended: true}));

exp.post("/", function(request, response){

    var channel = request.body.result.parameters.channel;

    var controltv = request.body.result.parameters.controltv;

}
```

Sau khi lấy được giá trị của các Entities của chuỗi JSON, chúng ta cần gửi chúng đến cho Raspberry Pi. Thông tin gửi đến Raspberry là chuỗi JSON có dạng:

```
{"code": data}
```

Trong đó code chính là các Entities đã đọc được từ JSON của Google Assistant. Để gửi đến cho Raspberry, ta sử dụng lệnh **emit** của thư viện **Socket.io:**

```
raspi.emit("APP_RASPI", {"code": data});
```

❖ **Với chiều từ Server tới Google Assistant**

Khi người dùng chỉnh sửa hoặc thêm mới một kênh tivi thì Google Assistant cần phải biết được sự thay đổi đó, để có thể hiểu được lệnh của người dùng. Khi người dùng thay đổi danh sách kênh, thông tin danh sách kênh mới sẽ được gửi đến Server, và Server sẽ gửi lên cho Google Assistant, cụ thể là gửi lên cho công cụ DialogFlow. Chuỗi JSON chứa thông tin kênh có dạng:

```
  [{"value":"50"},{"value":"26"},{"value":"27"},{"value":"4"},{"value":"80"},{"
value":"24"},{"value":"25"},{"value":"86"},{"value":"690"}]
```

trong đó ***value*** là thông tin chứa số kênh của mỗi kênh.

Server gửi thông tin danh sách kênh cho DialogFlow theo phương thức PUT của giao thức Http. Để gửi dữ liệu theo phương thức PUT, ta sử dụng thư viện **request** trong Nodejs:

```
var request = require('request');

request(options, function (error, response, body) {

        if (!error && response.statusCode == 200) {

            // Print out the response body

            console.log("Dialogflow response: " + JSON.stringify(body));

        }

    })
```

Trong đó options là bản tin mà Server gửi lên cho DialogFlow. Bản tin options có dạng:

```
var options = {

        url: 'https://api.dialogflow.com/v1/entities?v=20150910&lang=en',

        method: 'PUT',

        headers: headers,

        body: body

    }
```

Url ở đây là địa chỉ do DialogFlow cung cấp cho chức năng cập nhật Entities theo phương thức PUT và địa chỉ này là cố định. **Headers** của bản tin chứa **Developer access token** của project trên DialogFlow:

```
var headers = {

        'Authorization':    'Bearer b90129cb0a5a4b568bc2e960e1397168',

        'Content-Type':     'application/json'

    }
```

Body của bản tin là chuỗi JSON gồm 2 thông số là ***entries*** và ***name***. Entries là thông tin chứa chuỗi JSON danh sách kênh mà ứng dụng điện thoại gửi lên, name có giá trị là ENTITIES:

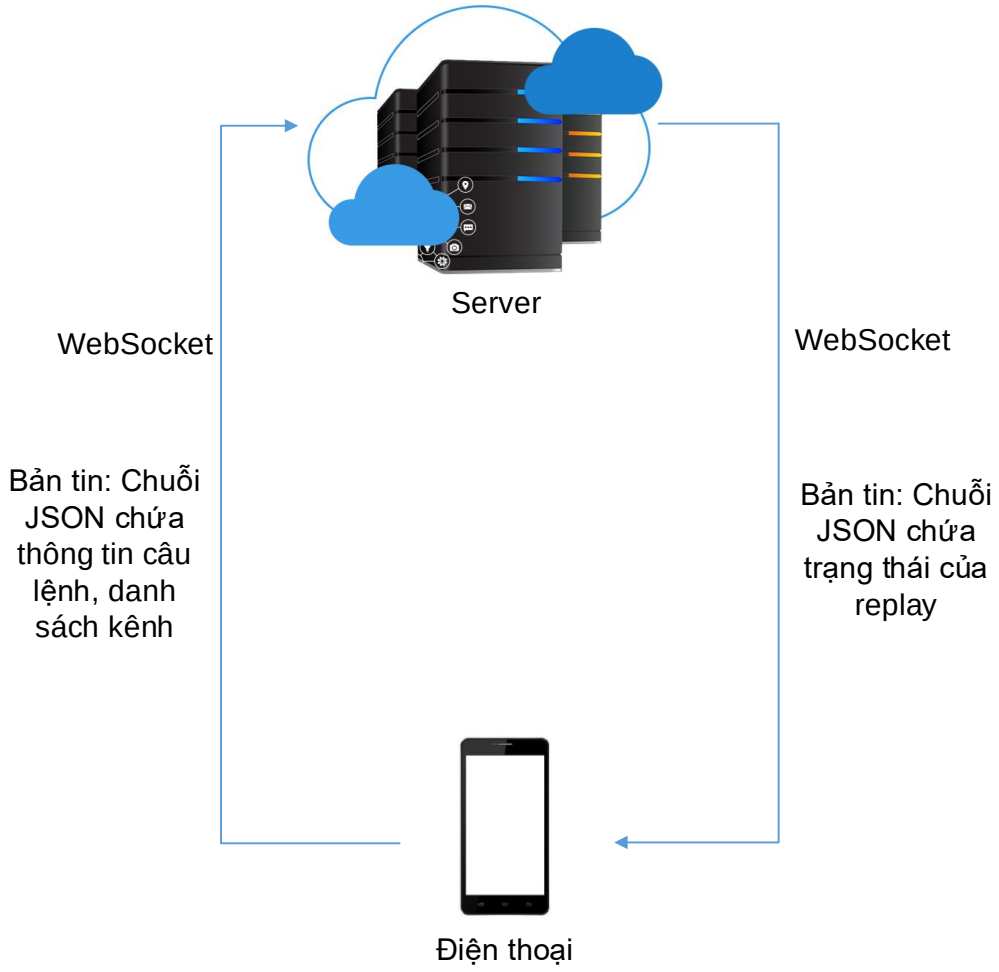
```
var body =
    [
        {
            "entries": data,
            "name": ENTITIES
        }
    ]
```

Dưới đây là hàm gửi chuỗi JSON danh sách kênh từ Server cho DialogFlow, tham số vào là **data** - chuỗi JSON danh sách kênh:

```
function UpdateEntities(data){
    // Set the headers
    var headers = {
        'Authorization':    'Bearer b90129cb0a5a4b568bc2e960e1397168',
        'Content-Type':     'application/json'
    }
    //Json to update entities, if not create -> create new entities
    var bodyJson =
    [
        {
            "entries": data,
            "name": ENTITIES
        }
    ]
    //Sap xep lai bodyJson
    var body = JSON.stringify(bodyJson);
    // Configure the request
```

```
    var options = {
        url: 'https://api.dialogflow.com/v1/entities?v=20150910&lang=en',
        method: 'PUT',
        headers: headers,
        body: body
    }
    // Start the request
    request(options, function (error, response, body) {
         if (!error && response.statusCode == 200) {
            // Print out the response body
            console.log("Dialogflow response: " + JSON.stringify(body));
        }
    })
}
```

3.2.2 Điện thoại với Server



Hình 3.3 Sơ đồ kết nối giữa điện thoại và server

3.2.2.1 Điện thoại gửi tới Server

Ứng dụng điện thoại kết nối tới Server theo phương thức WebSocket, bản tin giao tiếp dưới dạng chuỗi JSON. Phía Server sẽ lắng nghe các sự kiện được gửi từ điện thoại, có 4 loại sự kiện được gửi từ điện thoại bao gồm:

- Sự kiện điều khiển Relay nguồn điện
- Sự kiện điều khiển tivi
- Sự kiện cập nhật danh sách kênh
- Sự kiện cập nhật trạng thái Relay nguồn điện

❖ **Sự kiện điều khiển relay**

Sự kiện điều khiển Relay được gửi đi khi người dùng có yêu cầu bât/tắt nguồn điện tivi từ ứng dụng.

Hình 3.4 Bật/tắt relay

Chuỗi JSON của sự kiện bật tắt relay có dạng:

```
{"Status":1}
```

trong đó Status là trạng thái của relay, 1 tương ứng với relay bật, 0 tương ứng với relay tắt. Sự kiện điều khiển relay có tên là **APP_ESP**, phía Server sẽ dựa vào tên sự kiện này để bắt được dữ liệu và gửi cho Raspberry Pi:

```
socket.on("APP_ESP", function(data){
        console.log("App send to Esp8266: " + JSON.stringify(data));
        raspi.emit("APP_ESP", data);
    })
```

❖ **Sự kiện điều khiển tivi**

Sự kiện này được gửi đi khi người dùng yêu cầu chức năng ra lệnh bằng giọng nói trên ứng dụng để điều khiển tivi như tăng/giảm âm lượng, chuyển kênh,…Sự kiện điều khiển tivi có tên là **APP_RASPI**, dữ liệu gửi lên là chuỗi JSON có dạng:

```
{"remote":"TCL_TV","code":"25"}
{"remote":"TCL_TV","code":"VOLUMEUP"}
```

trong đó ***remote*** là tên của loại tivi, ***code*** là thông tin điều khiển. Khi người dùng ra lệnh điều khiển kênh tivi thì ***code*** có giá trị là số kênh tivi, khi lệnh điều khiển chức năng tivi thì code có giá trị là VOLUMEUP, VOLUMEDOWN,…Phía Server lắng nghe sự kiện **APP_RASPI** và gửi dữ liệu bắt được cho Raspberry:

```
socket.on("APP_RASPI", function(data){
        console.log("App send to Raspberry: " + JSON.stringify(data));
        raspi.emit("APP_RASPI", data);
    })
```

❖ **Sự kiện cập nhật danh sách kênh**

Sự kiện cập nhật danh sách kênh được gửi đi khi người dùng chỉnh sửa, thêm mới hoặc xóa một kênh trong danh sách kênh. Mọi sự thay đổi trong danh sách kênh đều được gửi lên cho Server để Server gửi cho Google Assistant nhận biết được sự thay đổi đó. Sự kiện cập nhật danh sách kênh có tên CHANNEL, chuỗi JSON gửi lên có dạng:

```
[{"value":"50"},{"value":"26"},{"value":"27"},{"value":"4"},{"value":"80"},{"va
lue":"24"},{"value":"25"},{"value":"86"},{"value":"690"}]
```

Trong đó ***value*** có giá trị là số kênh trong danh sách kênh. Phía Server lắng nghe sự kiện CHANNEL và gửi dữ liệu thu được cho Google Assistan:

```
socket.on("CHANNEL", function(data){
        console.log("App send Dialogflow list channel: " + JSON.stringify(data)
);
        UpdateEntities(data);
    })
```

❖ **Sự kiện cập nhật trạng thái relay**

Sự kiện cập nhật trạng thái relay được gửi đi mỗi khi người dùng mở ứng dụng trên điện thoại. Mục đích của sự kiện này là giúp cho người dùng biết được relay đang bật hay đang tắt. Sự kiện này có tên UPDATE và dữ liệu gửi lên rỗng. Phía Server lắng nghe sự kiện UPDATE và gửi yêu cầu đến cho Raspberry:

```
socket.on("UPDATE", function(){
        console.log("App request update to Esp");
        raspi.emit("UPDATE", "");
    })
```

3.2.2.2 Server gửi tới điện thoại

Khi người dùng mở ứng dụng lần đầu, ứng dụng sẽ yêu cầu cập nhật trạng thái của relay. Nhận được yêu cầu từ ứng dụng, Server sẽ gửi sự kiện **ESP_APP** đến ứng dụng, chuỗi JSON được gửi đi chứa thông tin trạng thái hiện tại của relay:

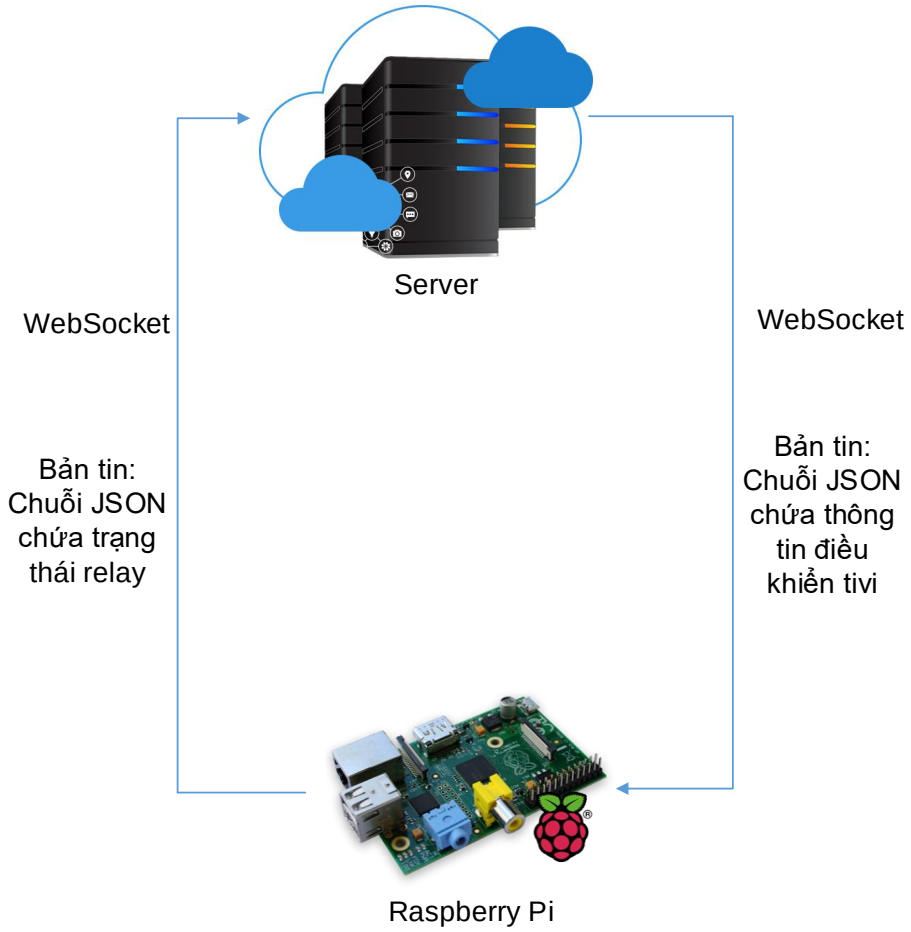
```
{"Status":1}
```

với 1 tương ứng với trạng thái bật, 0 tương ứng với trạng thái tắt.

Để gửi thông tin trạng thái của relay cho điện thoại, Server cần lắng nghe sự kiện cập nhật trạng thái relay từ Raspberry và gửi dữ liệu thu được cho điện thoại:

```
raspi.on('connection', function(socket){
    //Khi Esp gửi cho app trạng thái của replay
    socket.on("ESP_APP", function(data){
        console.log("Esp send to App: " + JSON.stringify(data));
        mobileApp.emit("ESP_APP", data);
    })
})
```

3.2.3 Raspberry Pi với Server



Hình 3.5 Sơ đồ kết nối giữa Raspberry và Server

3.2.3.1 Raspberry gửi tới Server

Raspberry Pi giao tiếp với Server thông qua giao thức WebSocket, sự kiện mà Raspberry gửi lên cho Server là sự kiện cập nhật trạng thái của relay, tên sự kiện ESP_APP, chuỗi JSON có dạng:

```
{"Status":1}
```

Với 1 tương ứng với trạng thái relay bật, 0 tương ứng với trạng thái relay tắt. Phía Server lắng nghe sự kiện ESP_APP và gửi dữ liệu thu được cho điện thoại:

```
socket.on("ESP_APP", function(data){
```

```
        console.log("Esp send to App: " + JSON.stringify(data));
        mobileApp.emit("ESP_APP", data);
    })
```

3.2.3.2 Server gửi tới Raspberry

Khi người dùng ra lệnh điều khiển tivi từ ứng dụng điện thoại hay trợ lý ảo Google Assistant thì các lệnh sẽ được chuyển tới Server, Server sẽ phân tích dữ liệu và chuyển đến cho Raspberry. Sự kiện Server gửi cho Raspberry có tên APP_RASPI, chuỗi JSON có dạng:

```
{"remote": NAME_TV, "code": code}
```

Trong đó, ***remote*** là tên của loại tivi, ***code*** là lệnh mà người dùng muốn điều khiển. Ví dụ, người dùng ra lệnh tăng âm lượng thì chuỗi JSON là:

```
{"remote": NAME_TV, "code": VOLUMEUP}
```

Các giá trị của ***code*** sẽ được Server xử lý khi nhận được lệnh của người dùng. ***Remote*** là thông tin tên của chủng loại tivi, tính năng này chưa được phát triển và được để mặc định trong Project này.

3.3 Kết luận

Server được thiết kế để kết nối các thiết bị ra lệnh của người dùng với thiết bị thực thi là Raspberry Pi, phương thức kết nối sử dụng chủ yếu là WebSocket, đảm bảo các Client luôn được kết nối với Server theo thời gian thực. Server quản lý các kết nối đến theo hướng sự kiện, kết hợp với khả năng chạy non-blocking của Nodejs đã giúp cho Server có thể xử lý nhiều sự kiện cùng một lúc.

Server đã nhận được các lệnh của người dùng gửi lên thông qua ứng dụng trên điện thoại và trợ lý ảo Google Assistant, dữ liệu gửi lên cho server là chuỗi JSON chứa thông tin về lệnh của người dùng. Dữ liệu gửi lên được Server xử lý và gửi cho thiết bị thực thi là Raspberry để thực hiện các công việc tiếp theo.

Điểm hạn chế lớn nhất của Server là chưa quản lý được người dùng và số lượng thiết bị kết nối tới, do đó việc triển khai hệ thống cho nhiều người dùng là chưa khả thi. Hiện tại hệ thống chỉ dừng lại dưới dạng mô hình cho một người dùng và một bộ thiết bị đi kèm, để triển khai cho nhiều người dùng thì Server cần có cơ sở dữ liệu cũng như chức năng quản lý người dùng và quản lý thiết bị kết nối tới.

CHƯƠNG 4. THIẾT KẾ ỨNG DỤNG TRÊN ĐIỆN THOẠI

4.1 Tổng quan

Ứng dụng được viết trên nền tảng hệ điều hành Android, có 3 chức năng chính:

- Điều khiển bật/tắt nguồn điện của tivi.
- Cho phép người dùng cài đặt danh sách kênh.
- Điều khiển các chức năng của tivi bằng giọng nói tiếng việt.

Hình 4.1 Giao diện chính của ứng dụng

4.2 Thiết kế các chức năng

4.2.1 Điều khiển bật/tắt nguồn tivi

Chức năng bật/tắt nguồn điện của tivi được thiết kế ngay trên giao diện chính của ứng dụng. Chức năng này cho phép người dùng kiểm soát được nguồn điện vào của tivi đồng thời cho người dùng biết được trạng thái của nguồn khi có điện hoặc mất điện.

Hình 4.2 Chức năng bật/tắt nguồn điện

Chức năng bật/tắt nguồn điện của ứng dụng được thiết kế là một Switch trong ứng dụng Android. Switch là một View trong android, nó có 2 trạng thái chọn lựa. Nó được sử dụng để hiển thị trạng thái **checked** và **unchecked** của một nút thông qua một thanh trượt cho người dùng. Switch có 2 button cơ bản là **on/off** cho biết trạng thái của Switch.

Khi người dùng ra lệnh điều khiển lên Switch, ứng dụng sẽ gửi sự kiện lên cho Server, bản tin gửi lên là một chuỗi JSON có dạng:

```
{"Status":1}
```

trong đó 1 tương ứng với trạng thái bật, 0 tương ứng với trạng thái tắt.

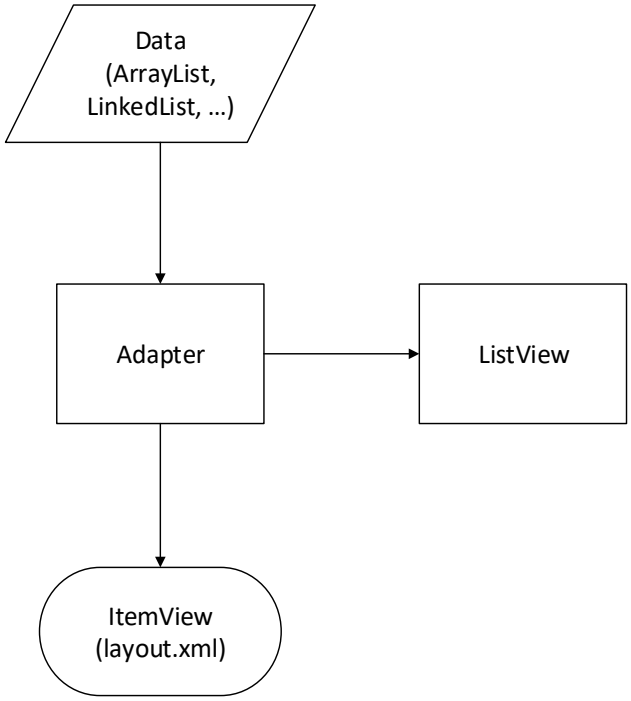
4.2.2 Danh sách kênh

Danh sách kênh hiển thị các kênh do người dùng thêm vào. Một kênh có ba thông số chính đó là:

- **Tên kênh:** là tên kênh truyền hình mà người dùng muốn gọi khi ra lệnh bằng giọng nói, ví dụ: VTV1, HÀ NỘI , HÀ NỘI 2,…
- **Số kênh:** là số kênh tương ứng với tên kênh trên điều khiển tivi, ví dụ kênh Hà Nội tương ứng kênh số 2, kênh VTV1 tương ứng kênh 1,…
- **Logo kênh:** là logo tương ứng với kênh người dùng đặt. Người dùng có thể chọn logo cho kênh của mình từ các logo có sẵn trong ứng dụng.

Hình 4.3 Danh sách cài đặt kênh

Để hiển thị được danh sách kênh, trong lập trình Android, ta sử dụng một **ListView** để hiển thị một danh sách do người dùng tự định nghĩa. **ListView** là một **GroupView** sẽ giúp chúng ta hiển thị dữ liệu lên màn hình dưới dạng một danh sách (có thể theo chiều đứng, hoặc nằm ngang).



Hình 4.4 Sơ đồ xây dựng một ListView

Để xây dựng 1 ListView ta cần có:

Thứ nhất: Về dữ liệu ta cần phải tổ chức dưới dạng 1 mảng dữ liệu: ta có thể sử dụng ArrayList<Object>, hoặc LinkedList, Enum, Cursor. Trong dự án này thì dữ liệu được sử dụng là ArrayList<Object>. Đối tượng Object ở đây chính là Kênh. Một kênh là một class riêng biệt với các thuộc tính là tên kênh, số kênh, và trạng thái đã chọn hay chưa chọn.

Thứ hai: Về ListView ta sẽ khai báo 1 thẻ ListView trong MainActivity cần hiển thị và mapping nó tới Class xử lý.

Thứ ba: ItemViews là 1 file layout.xml trong resource layout là giao diện các item mà ta muốn hiển thị. Dữ liệu của từng đối tượng trong mảng dữ liệu sẽ được hiển thị trên item này tùy vào yêu cầu người xây dựng. Đây là dạng layout do người dùng tự thiết kế, gọi là custom listview.

Thứ tư: Adapter là 1 lớp khai báo các itemView và mapping dữ liệu từ các đối tượng trong mảng dữ liệu và hiển thị lên ListView trên activity. Đây cũng là thành

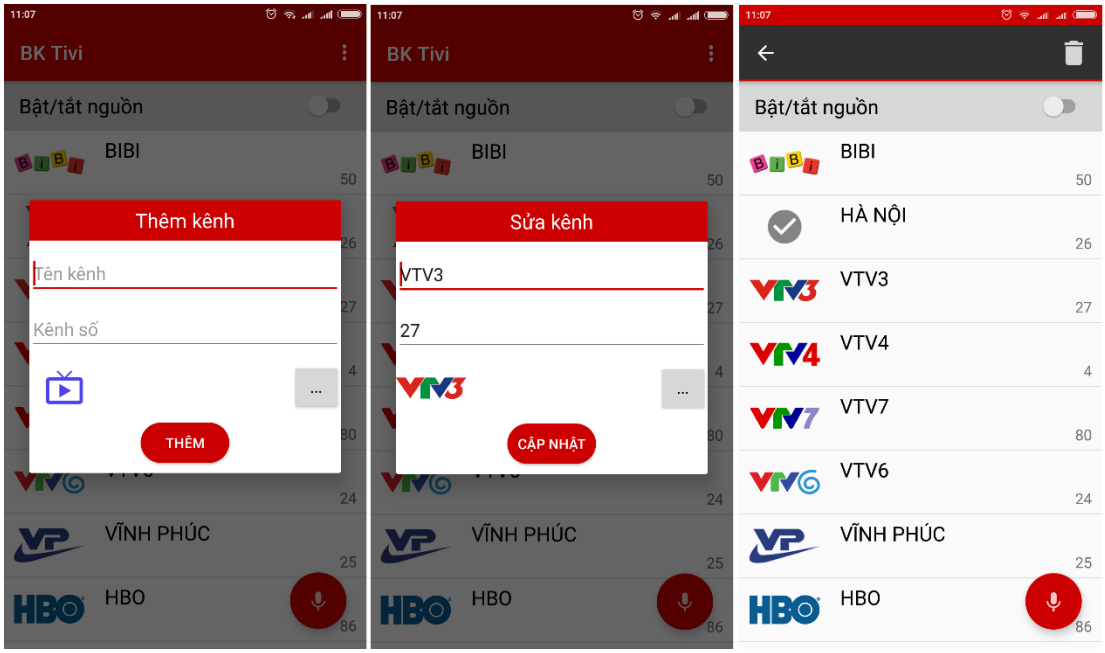
phần quan trọng nhất khi xây dựng 1 Listview để đảm bảo được hiệu năng và trải nghiệm người dùng tốt nhất thì nó hoàn toàn phụ thuộc vào cách xây dựng Adapater.

Trong **Adapter** các phương thức bắt buộc phải có:

- **getView():** hỗ trợ tạo ra các viewItem và mapping trực tiếp data lên các viewItem.
- **getCount():** trả về số lượng các viewItem dựa vào số size() của mảng data.
- **getItem(int position):** trả về Object dựa vào vị trí của đối tượng đó trong mảng dữ liệu.
- **getItemId(int position):** trả về id trên View của từng item mà adapter tạo ra.

4.2.3 Chỉnh sửa danh sách kênh

Chức năng chỉnh sửa danh sách kênh cho phép người dùng thêm, sửa hoặc xóa một kênh trong danh sách kênh của mình.



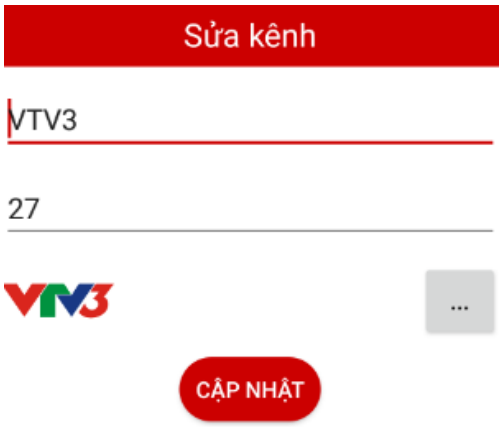
Hình 4.5 Thêm, sửa và xóa kênh

Chức năng thêm kênh mới được đặt trong menu trên thanh Actionbar, chức năng chỉnh sửa kênh sẽ hiện lên khi người dùng ấn vào một kênh bất kỳ. Hai chức năng

thêm kênh và sửa kênh là một hộp thoại ***Custom Dialog*** được thiết kế để người dùng có thể dễ dàng nhập dữ liệu.

Dialog có thể coi là một thông báo mà người dùng có thể tương tác trực tiếp được. Ví dụ khi muốn xóa một tập tin quan trọng hay muốn thoát một chương trình nào đấy thì việc hiển thì một thông báo để người dùng chắc chắn về hành vi của mình là rất quan trọng. Android hỗ trợ sẵn dialog cho lập trình viên nhưng đôi khi nó không phù hợp hoặc không đẹp, chúng ta có thể tự thiết kế dialog riêng theo ý mình mà không dùng đến layout có sẵn của Android.

Bố cục của hộp thoại **dialog** bao gồm một TextView là tên hộp thoại, các EditText cho phép người dùng nhập tên kênh và số kênh, một button chọn logo kênh và một button để cập nhật hoặc thêm kênh mới.



Hình 4.6 Hộp thoại Dialog thêm, sửa kênh

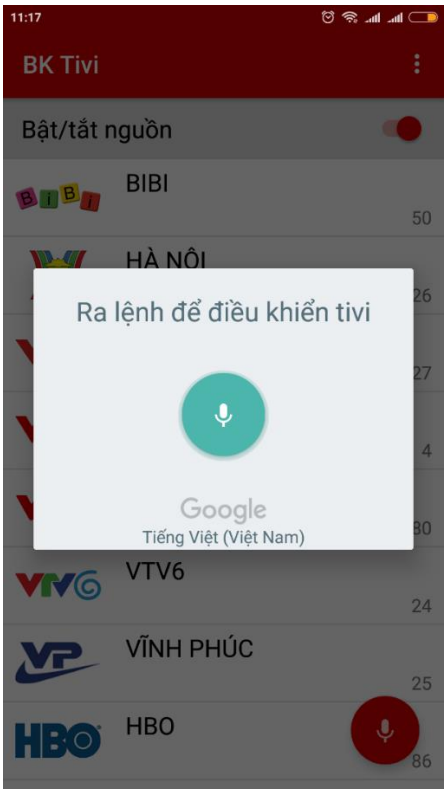
Khi người dùng thay đổi danh sách kênh, ứng dụng sẽ gửi danh sách kênh mới lên cho Server, mục đích của việc này nhằm giúp cho trợ lý Google Assistant hiểu được sự thay đổi của danh sách kênh. Chuỗi JSON ứng dụng gửi lên có dạng:

```
[{"value":"50"},{"value":"26"},{"value":"27"},{"value":"4"},{"value":"80"},{"va
lue":"24"},{"value":"25"},{"value":"86"},{"value":"690"}]
```

trong đó value chứa giá trị của số kênh. Để gửi lên server ta dùng câu lệnh ***emit*** trong thư viện ***Socket.io***:

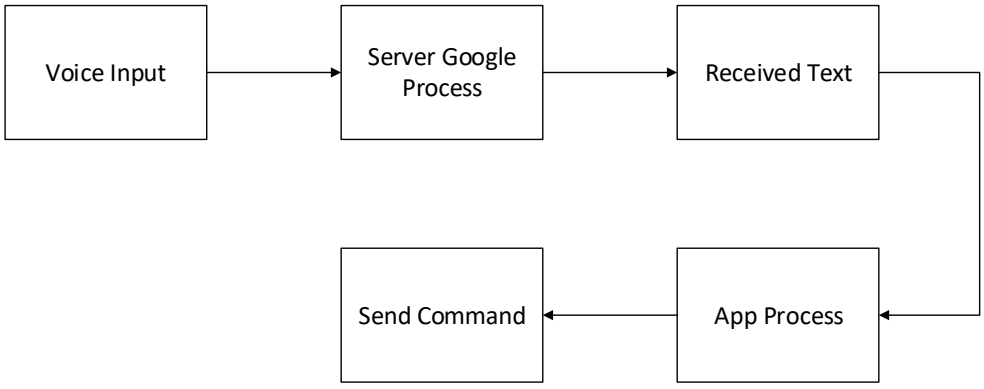
```
private void SendNumberChannelToServer(){
        JSONArray jsonArray = new JSONArray();
        for(int i = 0; i < arrayChannel.size(); i++){
            JSONObject jsonObject = new JSONObject();
            try {
                jsonObject.put("value", arrayChannel.get(i).numberChannel);
            } catch (JSONException e) {
                e.printStackTrace();
            }
            jsonArray.put(jsonObject);
        }
        //Emit len server
        mSocket.emit("CHANNEL", jsonArray);
    }
```

4.2.4 Điều khiển bằng giọng nói



Hình 4.7 Chức năng ra lệnh bằng giọng nói

Chức năng ra lệnh điều khiển bằng giọng nói được bật khi người dùng ấn vào **Button voice** góc dưới màn hình.



Hình 4.8 Sơ đồ hoạt động của Button Voice

Cơ chế hoạt động của **Button voice** là sẽ thu giọng nói đầu vào của người dùng (Voice Input), dữ liệu giọng nói sẽ được gửi lên Server của Google xử lý. Kết quả trả về là một chuỗi text dạng String của giọng nói đầu vào (Received Text). Chuỗi text nà được ứng dụng xử lý và gửi lệnh tương ứng đến Server.

Trong Android, Google cung cấp cho lập trình viên một API nhận dạng giọng nói là **ACTION_RECOGNIZE_SPEECH** , API này cho phép thu âm giọng nói của người dùng và trả về kết quả là một chuỗi String của giọng nói đó:

```
 public void VoiceClick(View v) {
        Intent voiceIntent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEEC
H);
        voiceIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerI
ntent.LANGUAGE_MODEL_FREE_FORM);
        voiceIntent.putExtra(RecognizerIntent.EXTRA_PROMPT,"Ra lệnh để điều khi
ển tivi");
        voiceIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, "vi");
        startActivityForResult(voiceIntent, REQ_CODE_SPEECH_INPUT);
    }
```

Kết quả String được trả về trong phương thức ***onActivityResult:***

```
@Override
    protected void onActivityResult(int requestCode, int resultCode, Intent dat
a) {
        super.onActivityResult(requestCode, resultCode, data);
        switch (requestCode) {
            case REQ_CODE_SPEECH_INPUT: {
                if (resultCode == RESULT_OK && null != data) {
                    ArrayList<String> result = data.getStringArrayListExtra(Rec
ognizerIntent.EXTRA_RESULTS);
```

```
                    try {
                        processResult(result.get(0));
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
                break;
            }
        }
    }
```

Phần xử lý String được thực hiện trong hàm ***processResult(String textResult)***. Hàm xử lý sẽ phân tích các từ khóa trong chuỗi thu được, sau đó tìm kiếm trong danh sách kênh của người dùng và gửi dữ liệu lên cho Server. Vì sử dụng phương pháp tìm kiếm từ khóa nên người dùng chỉ được nói các câu lệnh theo mẫu, ví dụ:

- Mẫu câu chuyển kênh:

  Bật kênh + tên kênh hoặc ***Bật kênh + số kênh***

- Mẫu câu điều khiển chức năng:

  Tăng âm lượng, giảm âm lượng, tắt tivi, bật tivi,…

Đây là các mẫu câu mà người dùng ra lệnh để bật kênh mong muốn. Phương pháp nhận dạng được kênh mà người dùng mong muốn bật ở đây là ta sẽ xét chuỗi string nhận được, tìm trong chuỗi từ khóa ***"kênh"***, khi tìm được vị trí của từ khóa ***"kênh"*** ta sẽ lấy phần chuỗi còn lại phía sau, phần chuỗi đó chính là kênh người dùng muốn bật.

Sau khi xử lý được yêu cầu của người dùng, ứng dụng sẽ gửi dữ liệu lên Server theo phương thức WebSocket, chuỗi JSON gửi lên có dạng:

```
{"remote": NAME_TV, "code": VOLUMEUP}
```

Trong đó ***remote*** là tên loại tivi, tính năng này hiện tại chưa được phát triển nên để mặc định, ***code*** chính là kết quả xử lý yêu cầu của người dùng. Để gửi lên Server ta sử dụng câu lệnh emit trong thư viện ***Socket.io***:

```
mSocket.emit(“APP_RASPI”, data);
```

trong đó data là chuỗi JSON gửi lên.

4.3 Kết luận

Ứng dụng trên điện thoại đã giúp người dùng tương tác với hệ thống một cách trực quan hơn. Ứng dụng giúp cho người dùng có thể cài đặt danh sách kênh theo ý thích của mình với một giao diện dễ sử dụng. Ngoài ra, với ứng dụng người dùng có thể điều khiển được nguồn điện cấp cho tivi, biết được tình trạng tivi đang có nguồn điện hay không.

Ứng dụng trên điện thoại khắc phục được nhược điểm của trợ lý ảo Google Assistant đó là hỗ trợ điều khiển bằng giọng nói tiếng biệt cho hệ thống. Người dùng có thể ra lệnh chuyển đến kênh mà mình đã cài đặt trước theo tên kênh hoặc số kênh. Ví dụ kênh **Vĩnh Phúc** có số kênh là **25**, người dùng chỉ cần nói:

“bật kênh Vĩnh Phúc” hoặc “***bật kênh 25***”,

ứng dụng sẽ nhận biết được lệnh của người dùng và gửi yêu cầu tới cho Server.

Tuy nhiên, ứng dụng vẫn còn một số hạn chế nhất định. Do việc xử lý ngôn ngữ người dùng dựa vào việc so sánh các key word nên ứng dụng chỉ nhận dạng được các câu lệnh theo mẫu quy định sẵn. Ứng dụng hiện tại chưa có hướng dẫn các mẫu câu cho người dùng ra lệnh, điều này khiến cho người dùng có thể nói những câu lệnh sai quy định, dẫn đến ứng dụng không nhận dạng được.

CHƯƠNG 5. THIẾT KẾ ỨNG DỤNG TRONG GOOGLE ASSISTANT

5.1 Tổng quan

Trợ lý ảo Google Assistant được Google tích hợp vào trong thiết bị loa thông minh Google home và điện thoại có hệ điều hành Android 6.0 trở lên. Vì vậy để sử dụng trợ lý ảo Google Assistant, người dùng có thể sử dụng trên điện thoại sử dụng hệ điều hành Android 6.0 trở lên hoặc sử dụng thiết bị loa thông minh Google home. Hiện tại, Google Assistant chưa hỗ trợ ngôn ngữ tiếng Việt vì vậy ứng dụng sẽ sử dụng ngôn ngữ tiếng Anh để giao tiếp với người dùng.

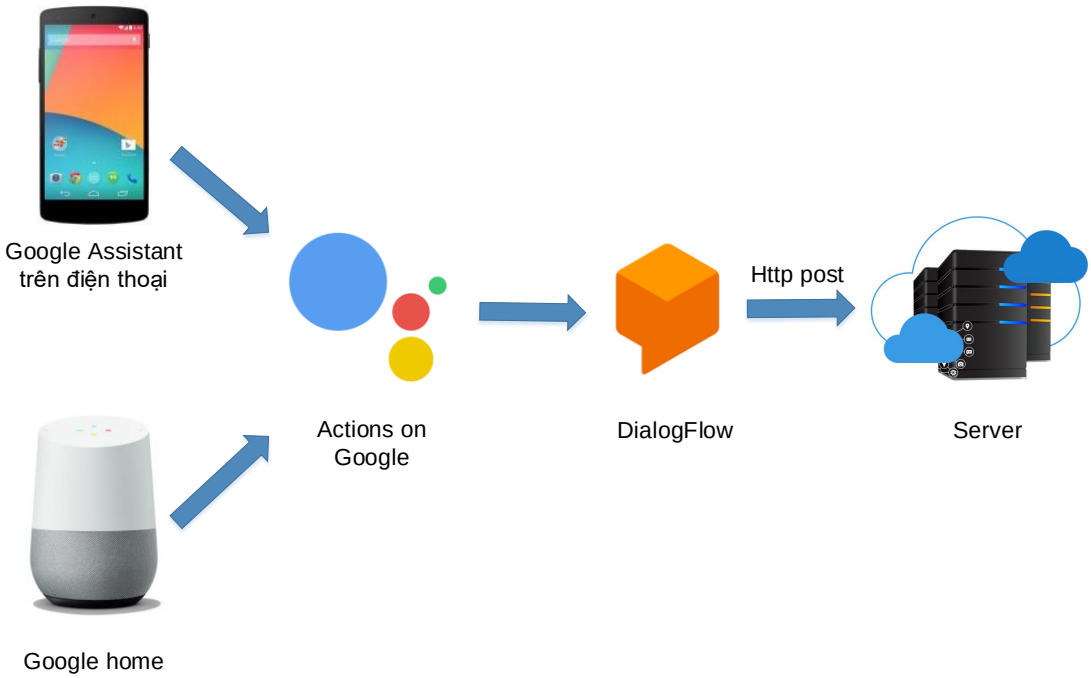
Để phát triển ứng dụng dựa trên Google Assitant, Google đưa ra cho lập trình viên công cụ Actions on Google. Công cụ này cho phép lập trình viên tạo ra một trợ lý ảo cho riêng mình, trợ lý ảo này sẽ chạy dựa trên Google Assistant. Đối với hệ thống này, chúng ta sẽ tạo ra trợ lý ảo có chức năng lắng nghe các lệnh điều khiển tivi của người dùng và thực thi các lệnh đó, đồng thời trả lời lại cho người dùng biết.

Trợ lý ảo của hệ thống được thiết kế với tên gọi **TV Control**. Sau đây là một số mẫu câu lệnh mà người dùng có thể ra lệnh cho trợ lý **TV Control** để điều khiển tivi:

- **Người dùng:** Turn on TV
- **TV Control:** OK, turn on TV now
- **Người dùng:** Volume up
- **TV Control:** OK, volume up now
- **Người dùng:** Turn on 25 channel
- **TV Control:** OK, 25 channel is on now

Trong các câu lệnh của người dùng, hệ thống chia ra làm hai loại câu lệnh: câu lệnh điều khiển các chức năng điều khiển tivi như bật/tắt, tăng/giảm âm lượng,.. và loại câu lệnh điều khiển kênh tivi như bật kênh 25, bật kênh 26,…Với loại câu lệnh điều khiển kênh, người dùng cần nói số kênh muốn bật cho trợ lý. Do Google Assistant chưa hỗ trợ ngôn ngữ tiếng việt nên người dùng chưa thể nói tên kênh cho trợ lý hiểu mà chỉ dùng số kênh, đây cũng là một hạn chế của hệ thống.

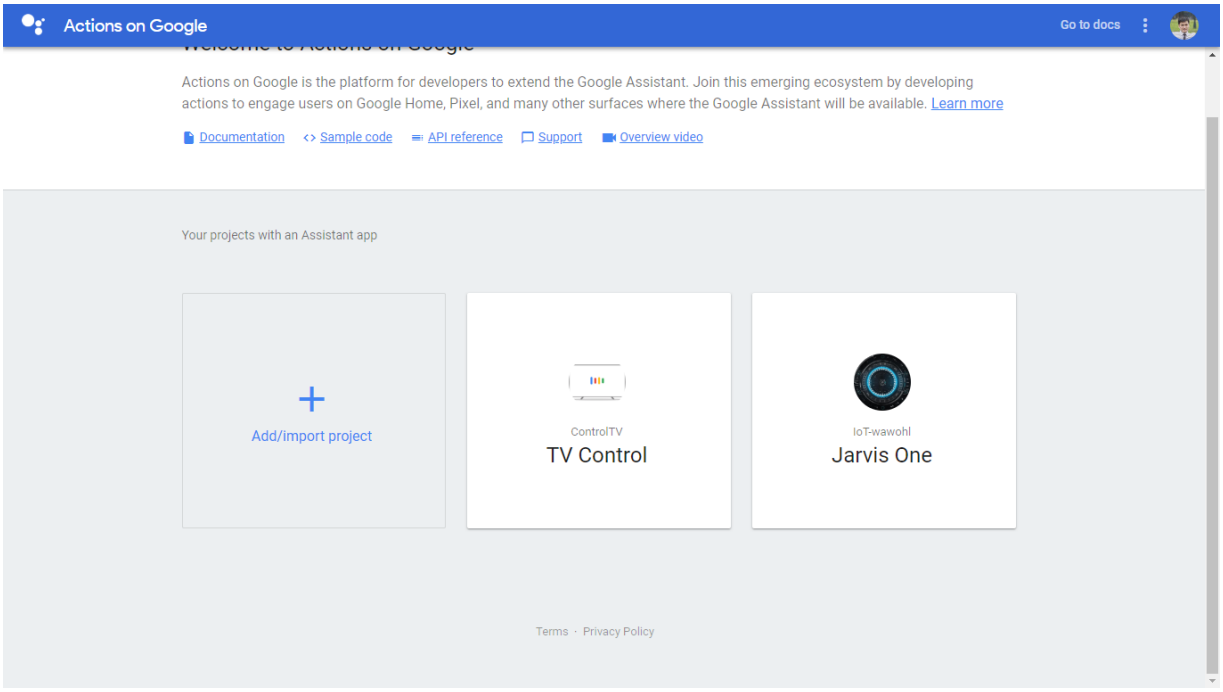
Dưới đây là sơ đồ cách hoạt động của ứng dụng chạy trên nền tảng Google Assistant:



Hình 5.1 Sơ đồ giao tiếp giữa Google Assistant với Server hệ thống

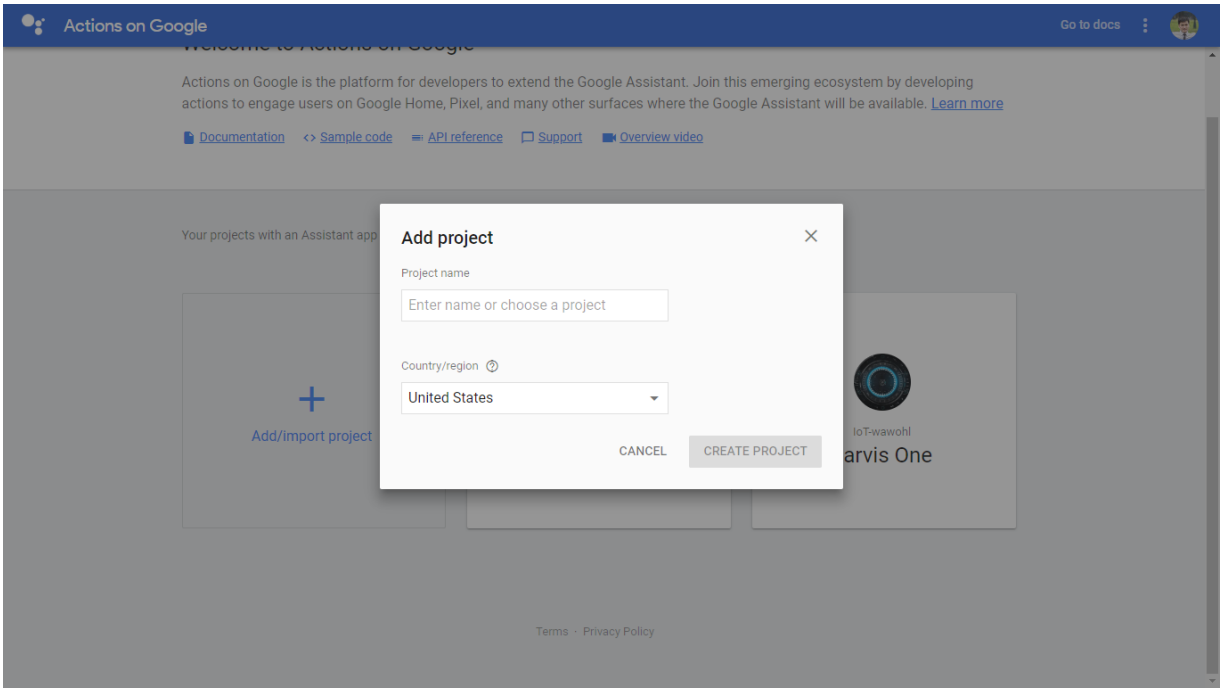
5.2 Thiết kế

Google cho phép lập trình viên thiết kế ứng dụng dựa trên Google Assistant hoàn toàn bằng nền tảng Web, cụ thể là công cụ Actions on Google và DialogFlow. Để tạo ứng dụng trong Actions on Google, vào trang chủ của Actions on Google và đăng nhập tài khoản lập trình viên Google, giao diện trang chủ như sau:



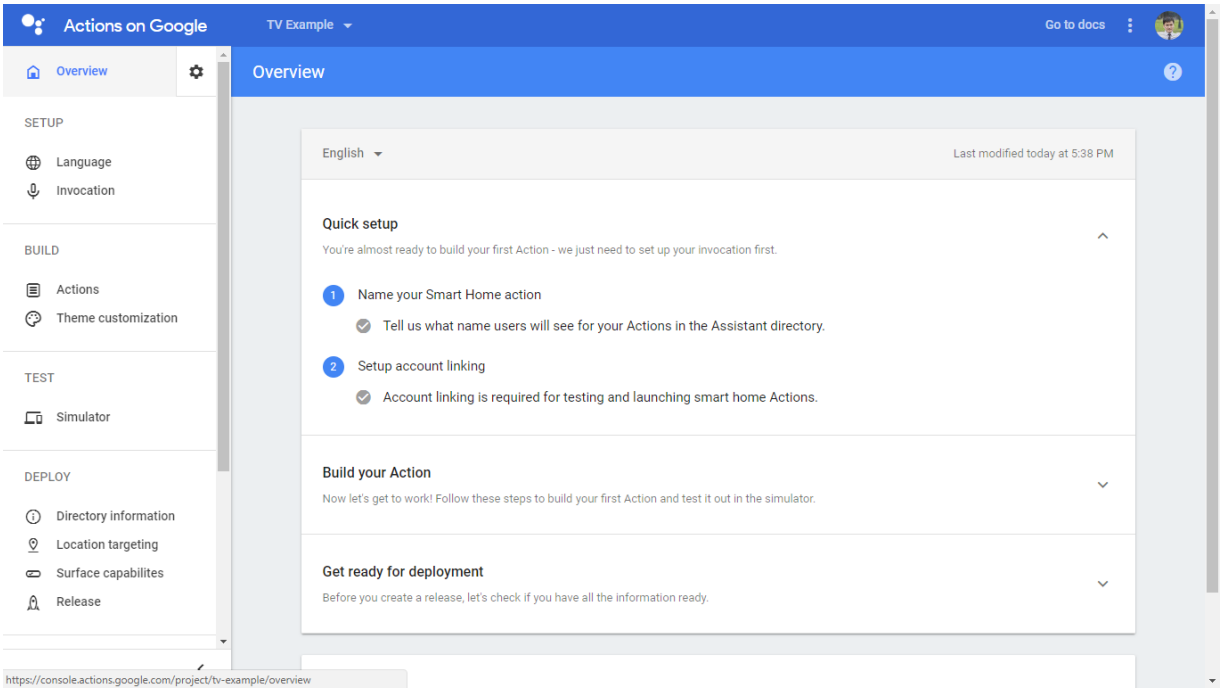
Hình 5.2 Giao diện chính của Actions on Google

Tại đây, lập trình viên có thể tạo cho mình một project mới. Click Add/import project để tạo project mới và đặt tên cho project của mình:

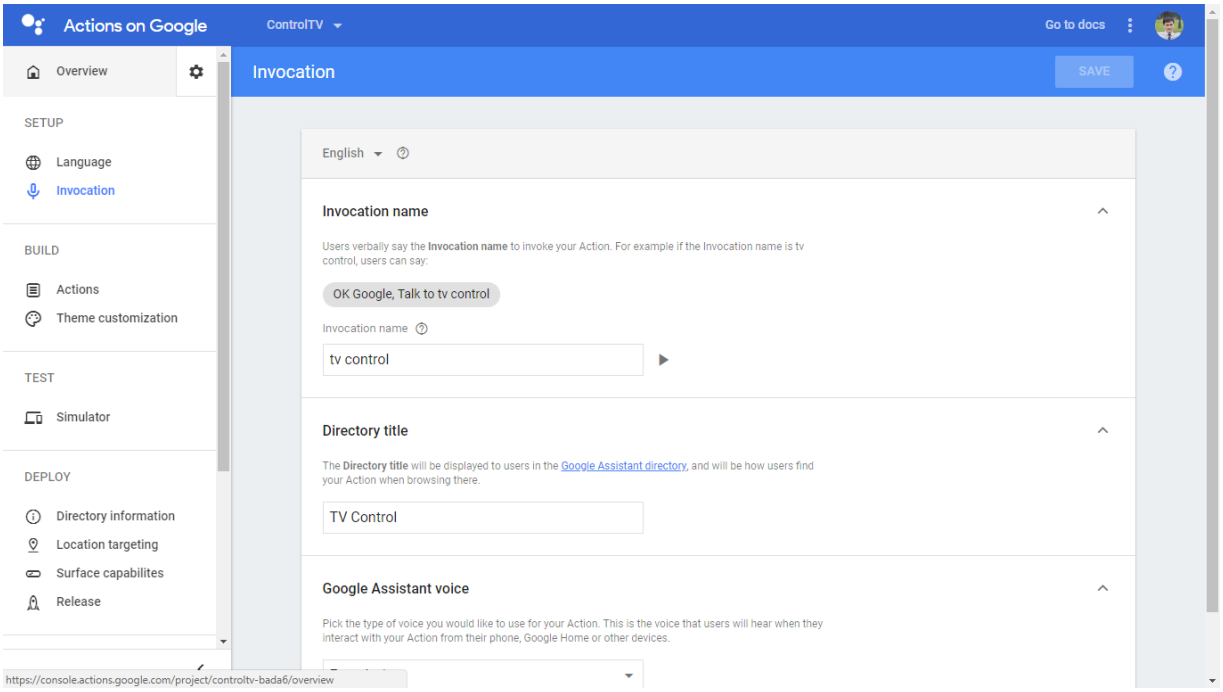


Hình 5.3 Tạo project mới trong Actions on Google

Sau khi tạo được project mới, chúng ta cần cài đặt cho ứng của mình như cài đặt ngôn ngữ, tên gọi cho trợ lý ảo,…

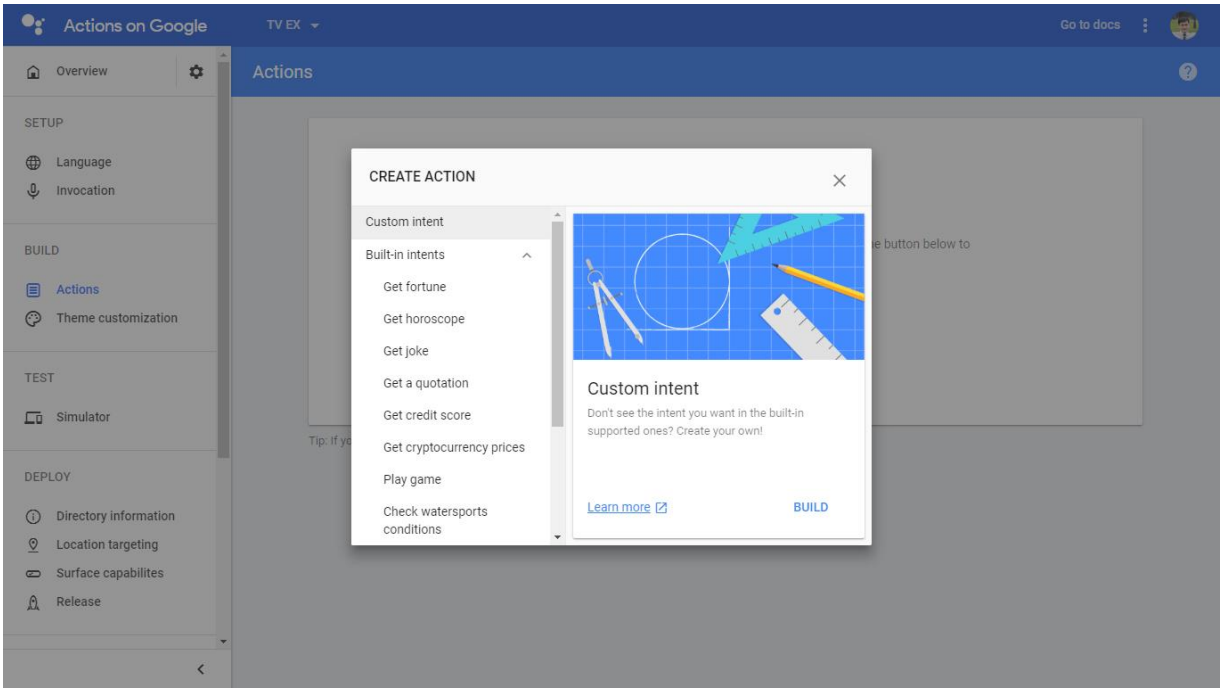


Hình 5.4 Giao diện cài đặt ứng dụng



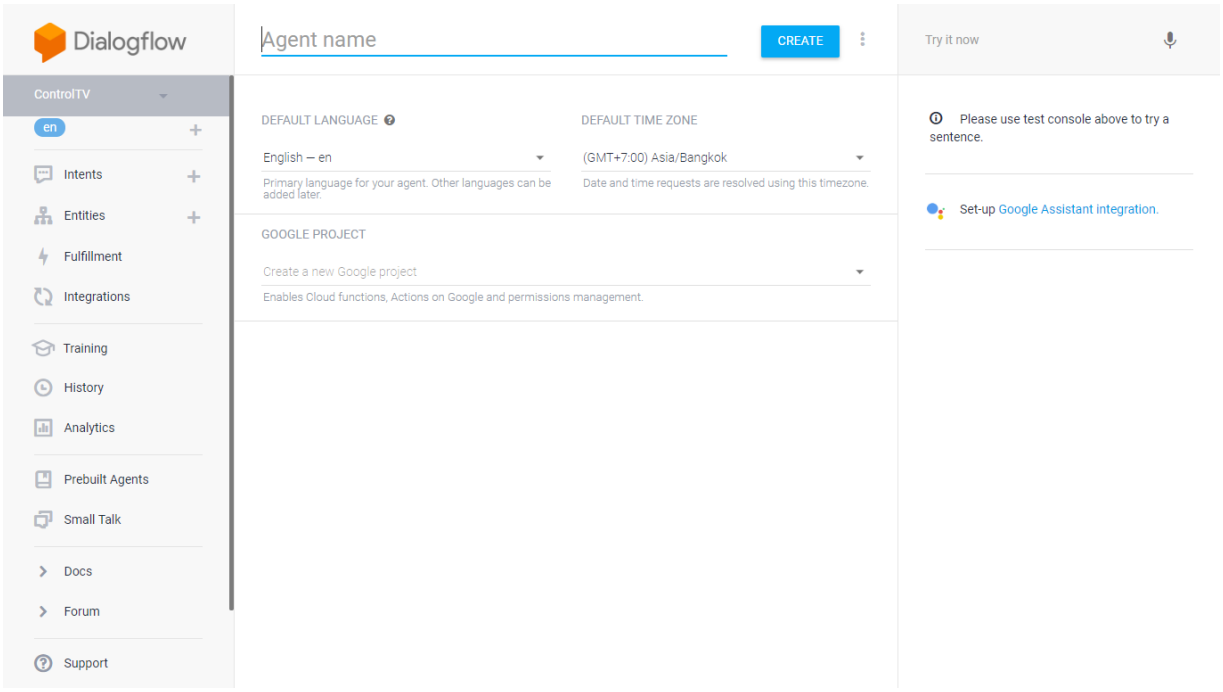
Hình 5.5 Đặt tên và kiểu giọng nói cho trợ lý ảo

Phần quan trọng trong quá trình cài đặt là thêm Actions cho ứng dụng. Việc thêm Actions có tác dụng liên kết ứng dụng tới công DialogFlow. Để thêm Actions, click vào menu Actions ở thanh menu bên trái. Trong cửa sổ mới hiện ra click vào **Add your first action:**



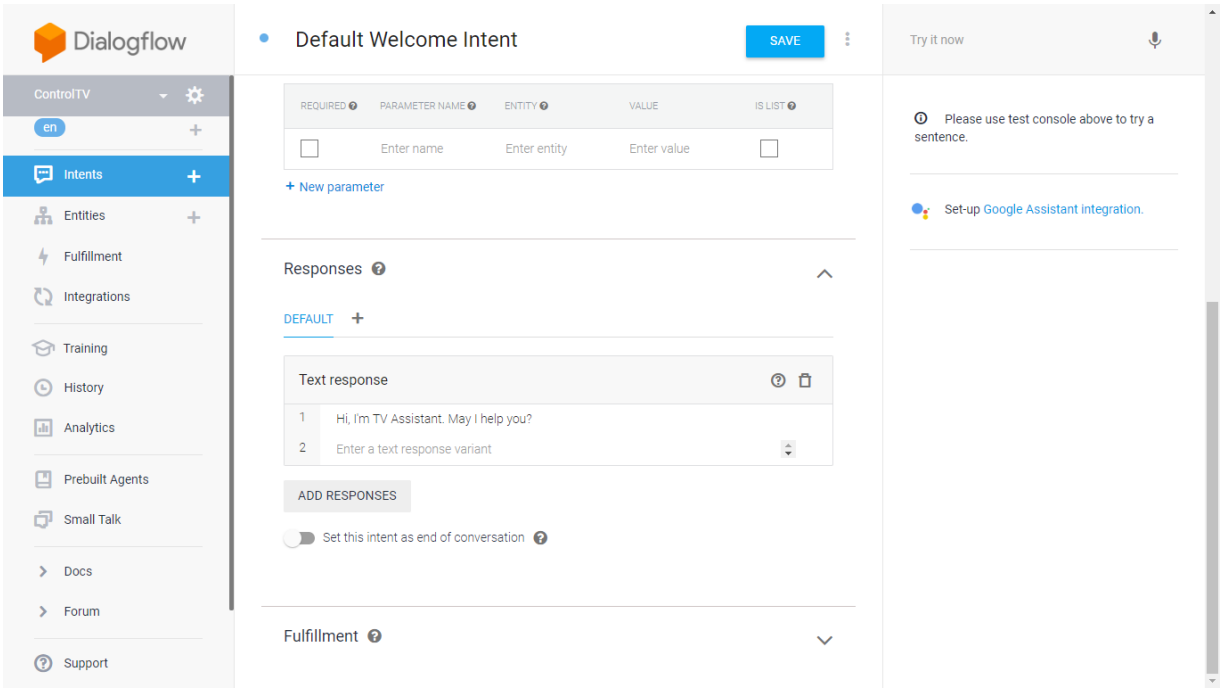
Hình 5.6 Thêm Action mới

Cửa sổ thêm Action mới hiện ra, chọn **Custom intent** và click vào nút **Build**. Khi click vào Build, Actions on Google sẽ link đến trang DialogFlow để tạo ra các đoạn hội thoại giữa người dùng và trợ lý ảo. Tại đây chúng ta sẽ tạo một Agent mới cho ứng dụng của mình:



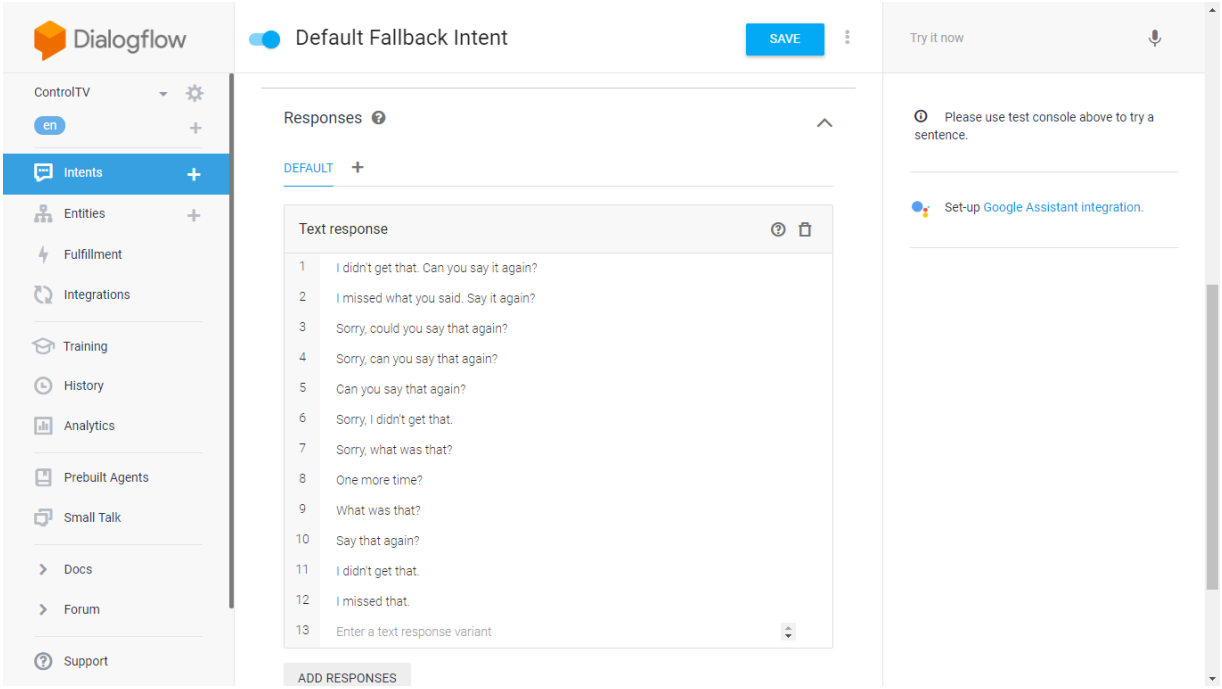
Hình 5.7 Giao diện tạo Agent mới trong DialogFlow

Đặt tên Agent cho ứng dụng là **ControlTV**. Sau khi tạo được Agent mới là ControlTV, chúng ta sẽ thêm các Intents cho Agent bằng cách click vào Intents trong menu bên trái. Có hai Intents nên có trong một Agent là **Default Fallback Intent** và **Default Welcome Intent**. **Default Welcome Intent** sẽ được gọi lần đầu tiên khi người dùng gọi trợ lý ảo, **Default Fallback Intent** sẽ được gọi khi người dùng nói những câu không nằm trong danh sách định trước hay nói cách khác là trợ lý ảo không hiểu yêu cầu của người dùng.



Hình 5.8 Default Welcome Intent

Phần Responses của Default Welcome Intent là những câu trả lời của trợ lý ảo khi người dùng gọi đến.



Hình 5.9 Default Fallback Intent

Phần Responses của Default Fallback Intent là những câu trả lời khi trợ lý ảo không hiểu được câu lệnh của người dùng.

Sau khi tạo được Default Welcome Intent và Default Fallback Intent, chúng ta tạo một Intent khác để phục vụ cho việc giao tiếp giữa người dùng và trợ lý ảo, đặt tên Intent đó là **Control.** Trong Control Intent là những mẫu câu hội thoại giữa người dùng và trợ lý, vì vậy để cho trợ lý hiểu được ta cần phải xác định được các Entities trong từng câu. Ví dụ, người dùng nói:

"Turn on 25 channel now"

Thì Entities trong câu là 25, là số kênh người dùng muốn bật. Từ đó mẫu câu người dùng nói khi muốn bật kênh là:

"Turn on [Entities] channel now"

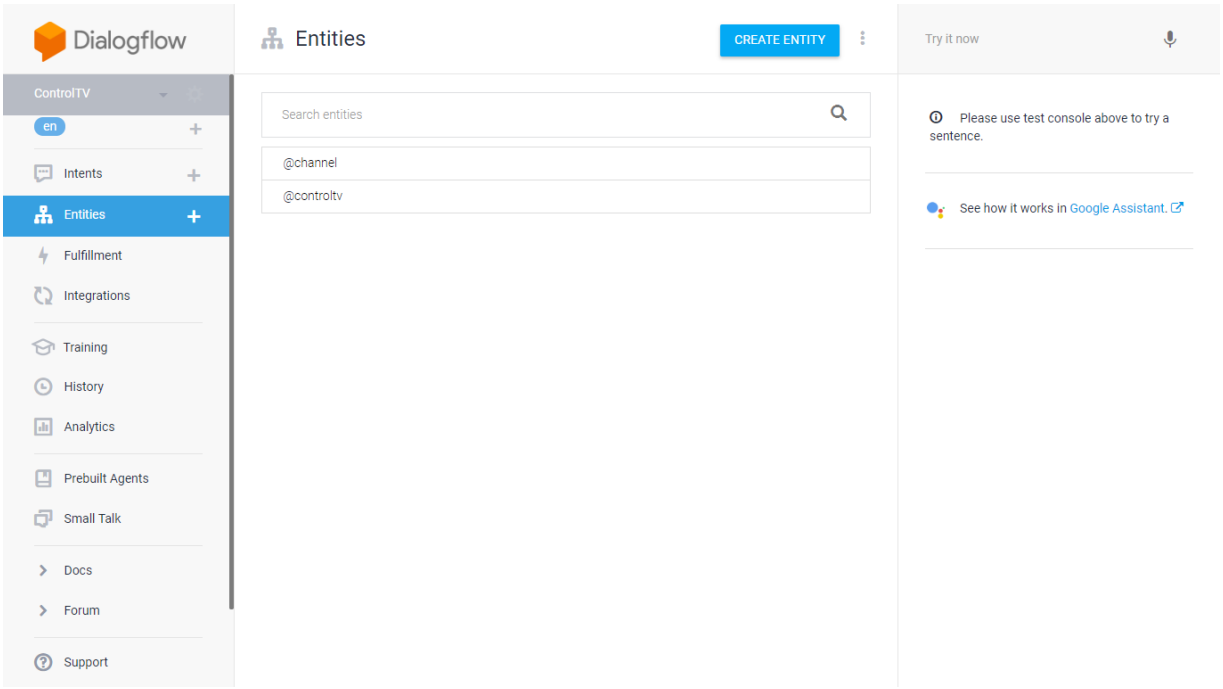
Tương tự với những câu lệnh điều khiển chức năng, ví dụ người dùng nói:

“Volume up now”

Thì Entities trong câu này là ***Volume up.*** Từ đó mẫu câu người dùng khi muốn điều khiển chức năng của tivi là:

“[Entities] now”

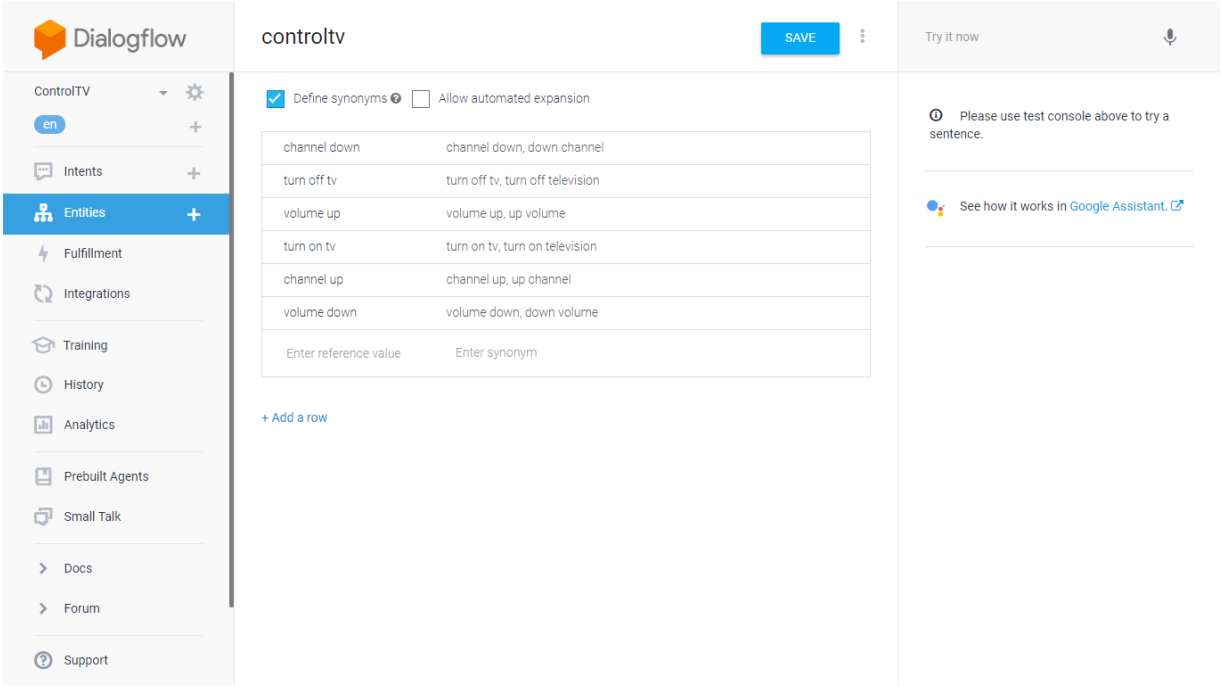
Tuy nhiên các Entities trong hai loại câu lệnh trên là khác nhau. Do đó chúng ta cần tạo ra hai loại Entities tương ứng hai loại mẫu câu trên. Để tạo Entities, trong menu bên trái, click vào Entities:



Hình 5.10 Tạo Entities mới

Tạo hai loại Entities mới là **Channel** và **ControlTV**. Channel Entities tương ứng với mẫu câu điều khiển kênh, ControlTV Entities tương ứng với mẫu câu điều khiển các chức năng của tivi.

Trong ControlTV Entities, ta thêm vào mỗi hàng hai thông số đó là tên các giá trị của Entities vào cột trái và các từ đồng nghĩa của nó vào cột bên phải.



Hình 5.11 ControlTV Entities

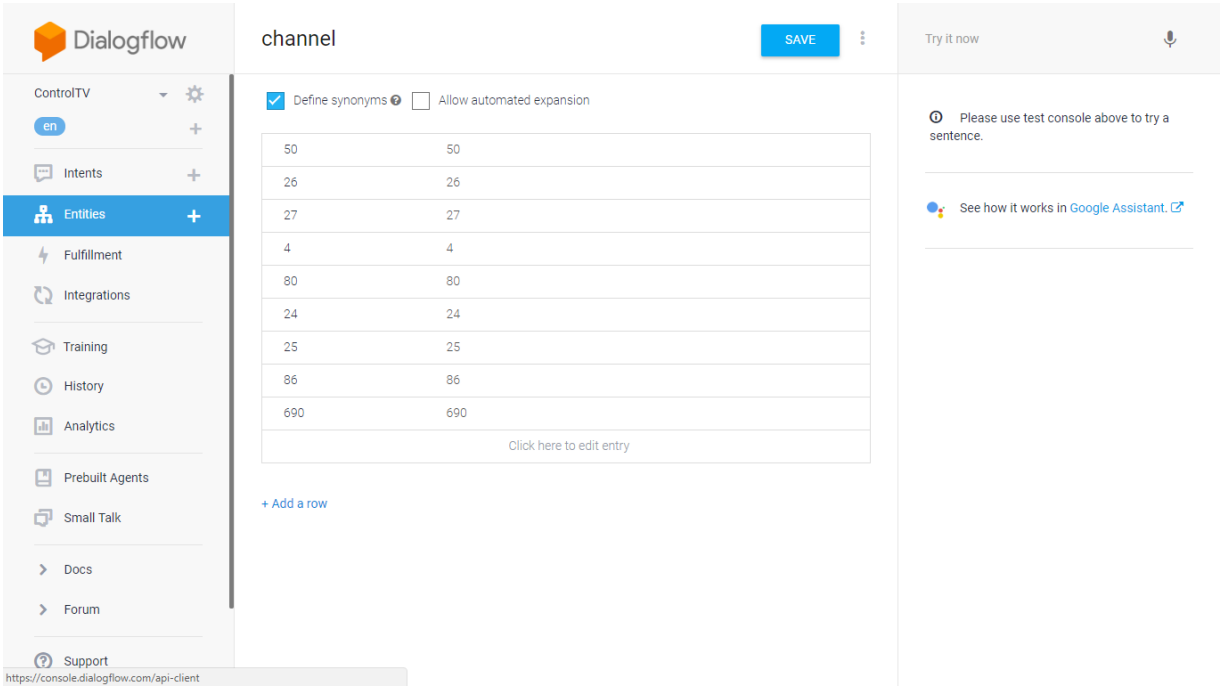
Trong Channel Entities ta sẽ thêm các số kênh trong danh sách kênh của người dùng. Tuy nhiên danh sách kênh của người dùng luôn được chỉnh sửa hoặc thêm mới trên ứng dụng điện thoại, do đó cần giải quyết vấn đề đó là khi người dùng thay đổi danh sách kênh trên ứng dụng điện thoại thì trên DialogFlow cũng phải nhận được sự thay đổi đó.

DialogFlow cho phép lập trình viên thêm các giá trị vào Entities một cách gián tiếp, có nghĩa là lập trình viên có thể viết code trong project của mình để gửi các giá trị Entities lên DialogFlow bằng nhiều ngôn ngữ như Nodejs, C/C++,…Dữ liệu gửi lên là chuỗi JSON có định dạng:

{“value”: “giá trị”}

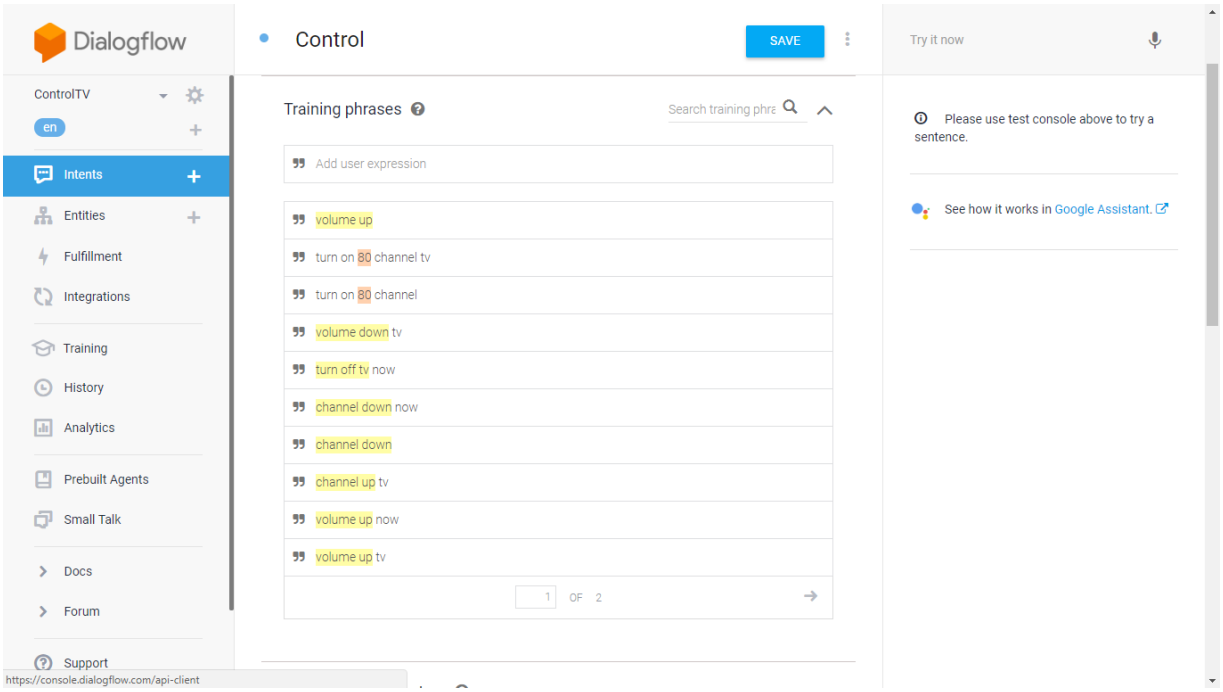
Ví dụ, danh sách kênh gửi lên trong Channel Entities là:

[{"value":"50"},{"value":"26"},{"value":"27"},{"value":"4"},{"value":"80"},{"value":"24"},{"value":"25"},{"value":"86"},{"value":"690"}]



Hình 5.12 Channel Entities

Sau khi đã tạo được các Entities, chúng ta quay lại Control Intent để huấn luyện một số mẫu câu cho trợ lý ảo. Trong mục **Training phrases** chúng ta nhập các mẫu câu mà người dùng nói, DialogFlow sẽ nhận biết được trong một câu đâu là Entities và đánh dấu chúng bằng màu sắc.



Hình 5.13 Training phrases Intents

Để trợ lý ảo có thể trả lời người dùng thì cần thêm vào phần **Responses** những mẫu câu trả lời. Ví dụ:

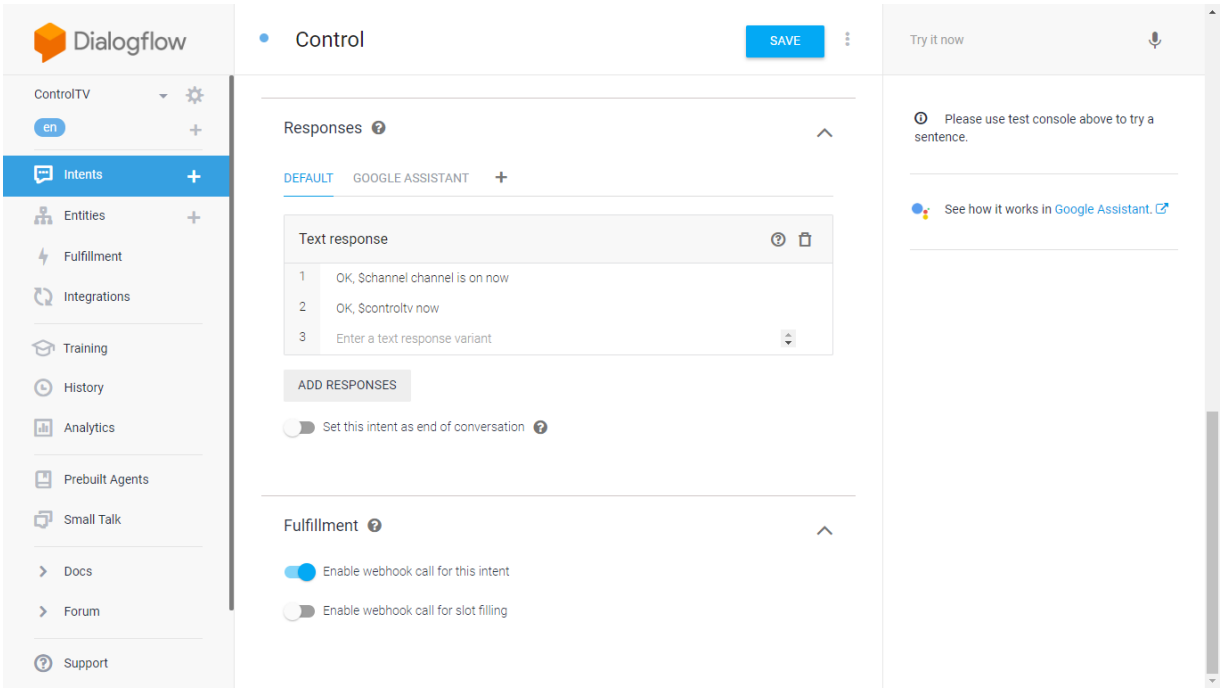
- **Người dùng:** *Turn on 25 channel*
- **TV Control:** *OK, 25 channel is on now*
- **Người dùng:** *Volume up now*
- **TV Control:** *OK, volume up now*

Với những câu lệnh điều khiển kênh thì câu trả lời có dạng:

OK, [Channel Entities] channel is on now.

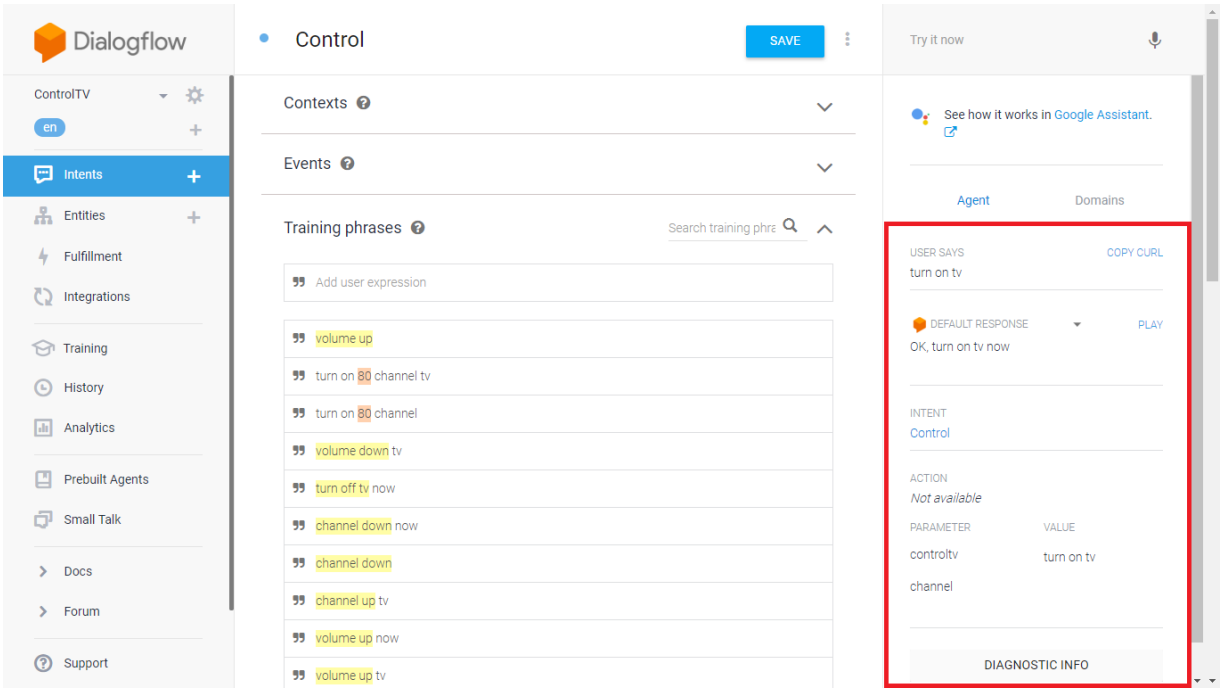
Với những câu lệnh điều khiển chức năng của tivi thì câu trả lời có dạng:

OK, [Controltv Entities] now.



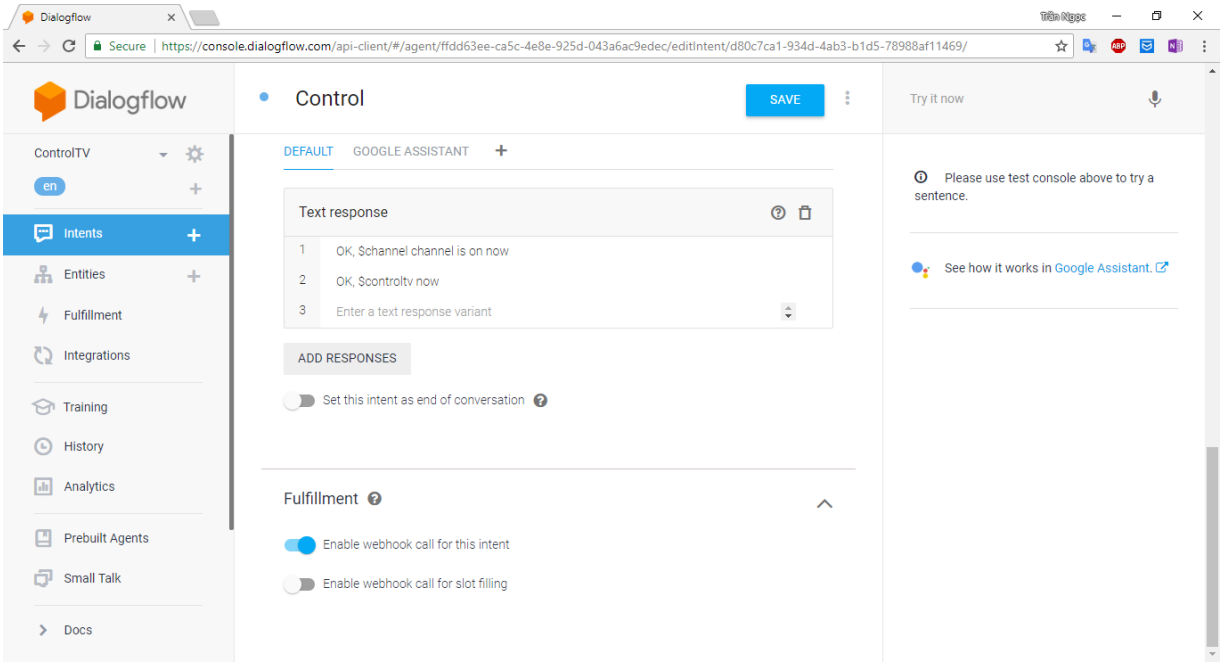
Hình 5.14 Responese Intents

Sau khi hoàn tất quá trình thiết kế đoạn hội thoại, chúng ta tiến hành test đoạn hội thoại vừa thiết kế. Trong phần **Try it now** phía bên phải, DialogFlow cho phép test bằng cách nhập từ bàn phím hoặc nói trực tiếp, kết quả trả về là câu trả lời đã thiết kế trong phần Responses.



Hình 5.15 Kết quả test

Việc tiếp theo là kết nối DialogFlow với Server. DialogFlow cho phép kết nối với Server thông qua phương thức Http. Để bật chức năng này, trong Control Intents phần **Fulfillment,** click vào **Enable Webhook call for this intent:**



Hình 5.16 Bật tính năng Webhook cho Intents

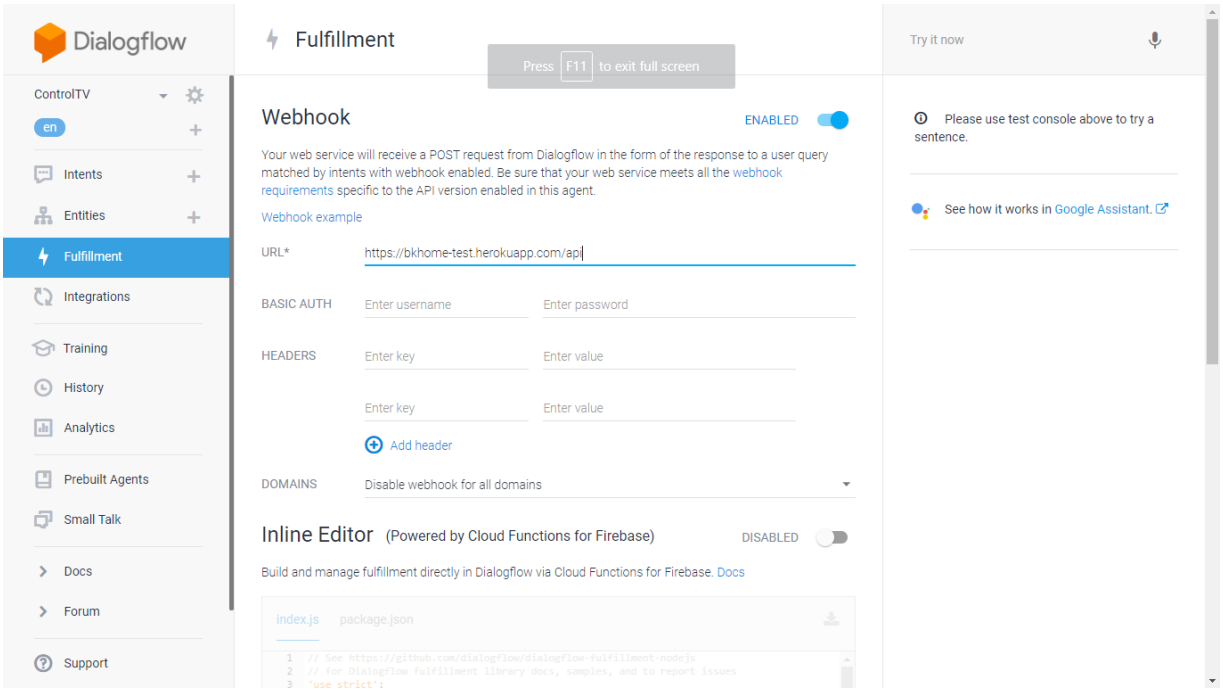
Mỗi khi người dùng ra lệnh thì DialogFlow sẽ trả về kết quả là một chuỗi JSON chứa các thông số của cuộc hội thoại, trong đó tham số quan trọng nhất là ***result*** chứa các kết quả trả về của lệnh từ người dùng, trong đó có các Entities của câu lệnh. Dưới đây là chuỗi JSON do DialogFlow trả về:

```
{
  "id": "0f6a612f-5d53-4ac5-a654-a21bbf97a096",
  "timestamp": "2018-05-17T06:42:22.903Z",
  "lang": "en",
  "result": {
    "source": "agent",
    "resolvedQuery": "turn on tv",
```

```
    "action": "",
    "actionIncomplete": false,
    "parameters": {
      "controltv": "turn on tv",
      "channel": ""
    },
    "contexts": [],
    "metadata": {
      "intentId": "d80c7ca1-934d-4ab3-b1d5-78988af11469",
      "webhookUsed": "true",
      "webhookForSlotFillingUsed": "false",
      "webhookResponseTime": 10001,
      "intentName": "Control"
    },
    "fulfillment": {
      "speech": "OK, turn on tv now",
      "messages": [
        {
          "type": "simple_response",
          "platform": "google",
          "textToSpeech": "OK, turn on tv now"
        },
        {
          "type": 0,
          "speech": "OK, turn on tv now"
        }
      ]
```

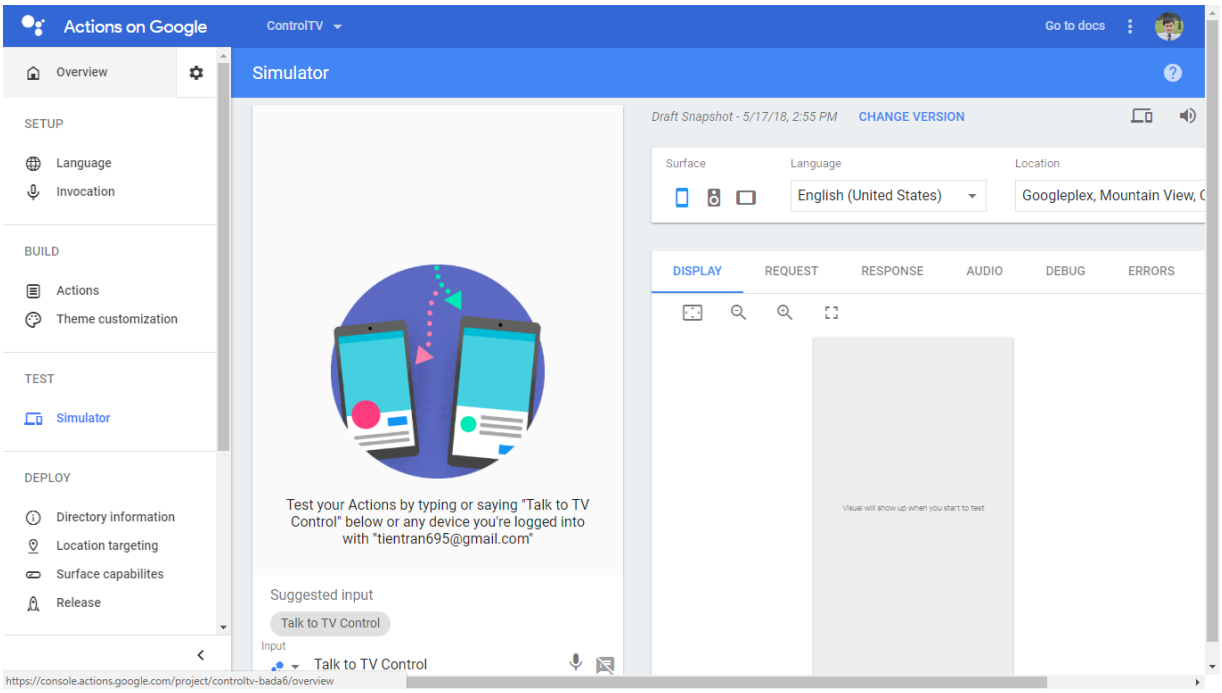
```
    },
    "score": 1
},
  "status": {
    "code": 206,
    "errorType": "partial_content",
    "errorDetails": "Webhook call failed. Error: Request timeout."
  },
  "sessionId": "2f611008-450d-4bc8-84e9-7540d49a770d"
}
```


Khi kết nối tới Server, DialogFlow sẽ gửi chuỗi JSON này tới Server thông qua phương thức HTTP POST request. Để kết nối tới Server, click vào **Fulfillment** ở menu bên trái. Trong Fulfillment, bật chế độ Webhook và nhập địa chỉ của Server vào ô URL. Địa chỉ Server là ***https://bkhome-test.herokuapp.com/api.*** Mỗi khi người dùng ra lệnh thì DialogFlow sẽ gửi chuỗi JSON trên đến Server, và nhiệm vụ của Server là giải mã chuỗi JSON này để biết được yêu cầu từ người dùng thông qua các Entities của câu.



Hình 5.17 Kết nối DialogFlow với Server

Sau khi quá trình cài đặt hoàn tất, chúng ta cần phải cập nhật lại cho Actions on Google hiểu. Trong phần test bên phải, chúng ta click vào **See how it works in Google Assistant** để cập nhật và chuyển đến trình mô phỏng của Google Assistant. Tại đây, Google cho phép lập trình viên mô phỏng ứng dụng của mình giống như trên thực tế.



Hình 5.18 Giao diện mô phỏng Google Assistant

5.3 Kết luận

Ứng dụng trợ lý ảo điều khiển tivi được phát triển dựa trên nền tảng Google Assistant đã có thể chạy trên các thiết bị điện thoại hệ điều hành Android 6.0 trở lên và thiết bị loa thông minh Google home, giúp người dùng có thể điều khiển tivi thông qua giọng nói. Trong phạm vi của đồ án, ứng dụng mới chỉ dừng lại ở mức giao tiếp cơ bản, cụ thể là các câu lệnh điều khiển kênh và điều khiển chức năng của tivi. Ví dụ:

Người dùng: Turn on tv

Control TV: OK, turn on tv now

Người dùng: Volume up

Control TV: OK, volume up now

Người dùng: Turn on 25 channel

Control TV: OK, 25 channel is on now

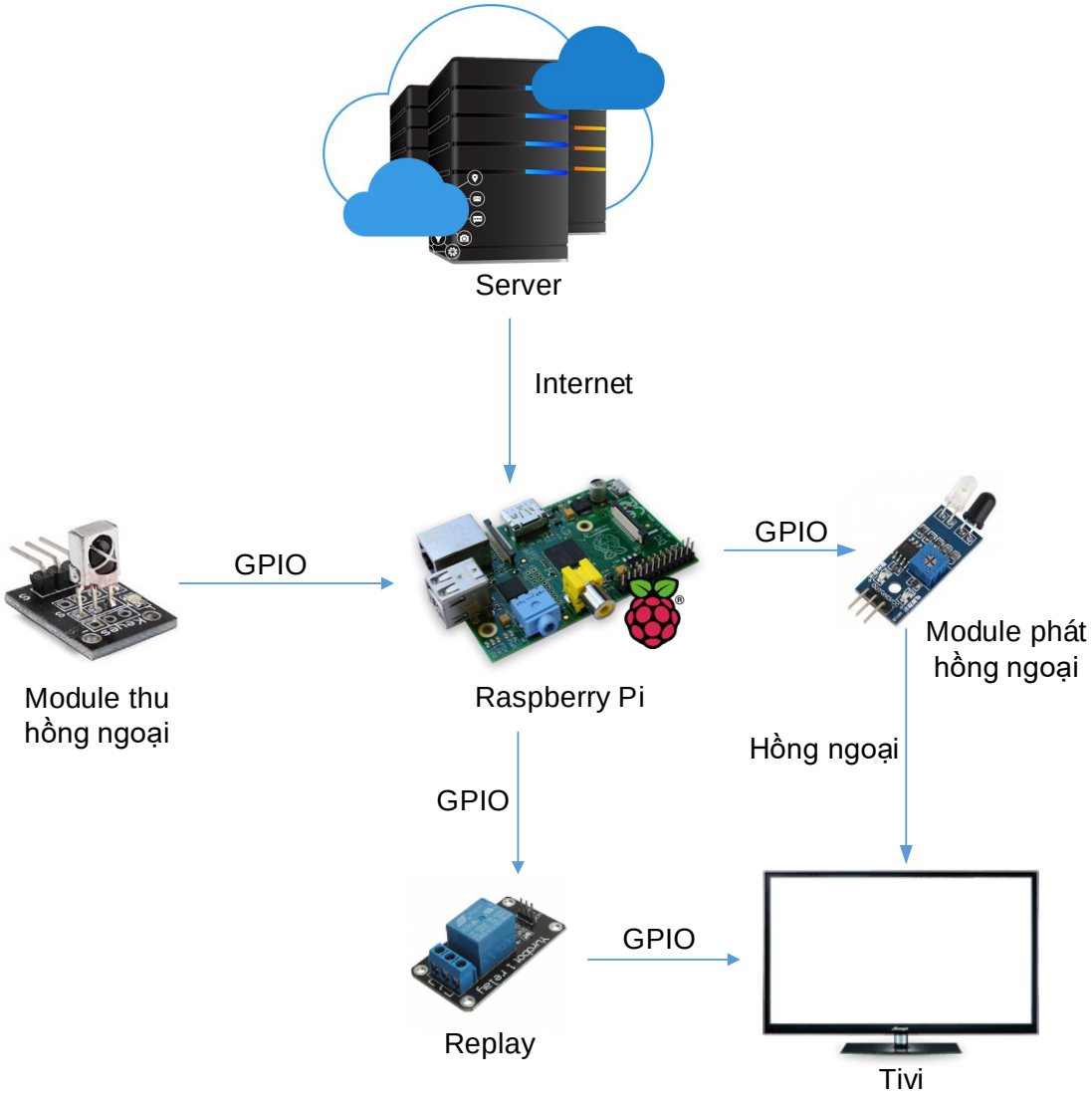
Tuy nhiên, do nền tảng Google Assistant chưa hỗ trợ ngôn ngữ tiếng Việt nên người dùng khi ra lệnh phải sử dụng tiếng Anh. Việc sử dụng tiếng Anh đưa đến một số hạn chế đó là người dùng không ra lệnh chuyển kênh bằng tên kênh mà phải dùng số kênh để ra lệnh bởi vì các tên kênh do người dùng cài đặt trên điện thoại đều dùng tiếng Việt, dẫn đến trợ lý ảo không hiểu được lệnh của người dùng.

CHƯƠNG 6. THIẾT KẾ KHỐI THỰC THI TRÊN RASPBERRY PI

6.1 Sơ đồ khối

Khối thực thi trên Raspberry Pi có chức năng kết nối với Server và nhận lệnh từ Server. Khi có lệnh từ Server, Raspberry Pi sẽ điều khiển hai thiết bị là Module hồng ngoại và Module replay tương ứng với từng lệnh từ Server. Cụ thể, Server sẽ gửi hai loại lệnh đó là lệnh điều khiển tivi và lệnh bật/tắt nguồn điện thông qua Replay. Khi Server gửi lệnh điều khiển tivi, Raspberry sẽ đọc tin nhắn từ Server và điều khiển Module hồng ngoại phát đúng tín hiệu như Server yêu cầu. Khi Server gửi lệnh bật/tắt nguồn điện, Raspberry sẽ điều khiển Module Replay như Server yêu cầu.

Để phát được các tín hiệu hồng ngoại đúng như tín hiệu từ remote tivi thì Raspberry cần phải học được tín hiệu từ remote đó. Chính vì vậy cần phải sử dụng Module thu hồng ngoại để thu các tín hiệu hồng ngoại từ remote tivi phục vụ cho việc phát lại. Tất cả các Module đều kết nối với Raspberry thông qua các chân GPIO. Dưới đây là sơ đồ khối thực thi trên Raspberry Pi:

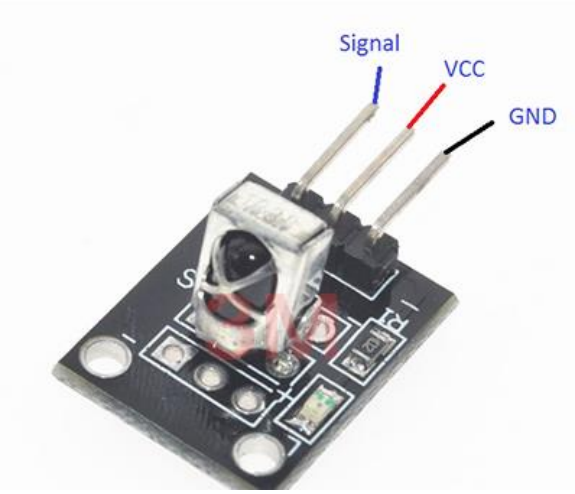


Hình 6.1 Sơ đồ khối thực thi trên Raspberry Pi

6.2 Thiết kế

6.2.1 Thu tín hiệu hồng ngoại

Để thu tín hiệu hồng ngoại từ remote, ta sử dụng mắt thu hồng ngoại TSOP1838 với tần số 38KHz, phù hợp với các loại điều khiển thông dụng.



Hình 6.2 Module thu hồng ngoại TSOP1838

Mắt thu hồng ngoại hoạt động ở mức điện áp 5V vì vậy chân Vcc của TSOP138 được nối với Vcc 5v của Raspberry Pi, chân ra tín hiệu Signal sẽ được nối với **GPIO 23** của Raspberry Pi.

Để cài đặt thư viện LIRC trên Raspberry ta chạy lệnh sau trên terminal [7]:

```
sudo apt-get install lirc
```

Sau đó cần cài đặt chân GPIO nhận tín hiệu hồng ngoại và chân phát tín hiệu hồng ngoại. Ở đây chân thu tín hiệu là chân GPIO 23, chân phát tín hiệu hồng ngoại là chân GPIO 22. Những chỉnh sửa này được thực hiện trong file **/etc/modules**:

```
lirc_rpi gpio_in_pin=23 gpio_out_pin=22
```

Tiếp theo cần phải chỉnh sửa thông số cài đặt hệ thống của Raspberry, mở file **/boot/config.txt** để chỉnh sửa, thêm vào file này dòng cuối cùng:

```
dtoverlay=lirc-rpi,gpio_in_pin=23,gpio_out_pin=22
```

Bước cuối cùng là chỉnh sửa lại file /etc/lirc/hardware.conf của thư viện LIRC như mẫu dưới:

```
# /etc/lirc/hardware.conf

# Arguments which will be used when launching lircd

LIRCD_ARGS="--uinput"

# Don't start lircmd even if there seems to be a good config file

# START_LIRCMD=false

# Don't start irexec, even if a good config file seems to exist.

# START_IREXEC=false

# Try to load appropriate kernel modules

LOAD_MODULES=true

# Run "lircd --driver=help" for a list of supported drivers.

DRIVER="default"

# usually /dev/lirc0 is the correct setting for systems using udev

DEVICE="/dev/lirc0"

MODULES="lirc_rpi"


# Default configuration files for your hardware if any

LIRCD_CONF=""

LIRCMD_CONF=""

############################################################
```

Sau khi hoàn tất quá trình cài đặt thì khởi động lại Raspberry Pi.

Tiếp theo cần phải thu lại tín hiệu hồng ngoại từ Remote và lưu lại vào thư viện với tên phím tương ứng. Để Remote lại gần mắt thu và chạy lệnh sau:

```
sudo /etc/init.d/lirc stop

mode2 -d /dev/lirc0
```

Khi ấn một phím bất kì thì màn hình terminal xuất hiện các thông số của tín hiệu hồng ngoại dạng như sau:

```
space 14529891

pulse 8451

space 4734

pulse 274

space 884

pulse 403

space 532

pulse 635

space 649

pulse 440

space 680

pulse 462
```

Sau khi thư viện đã thu được tín hiệu hồng ngoại từ Remote thì ta tiến hành ghi lại các tín hiệu của từng nút trên remote và đặt tên cho chúng. Tên thường sẽ đặt theo chức năng của nó, chẳng hạn nút tăng âm lượng sẽ đặt là KEY_VOLUMEUP,…Chạy lệnh sau để thực hiên:

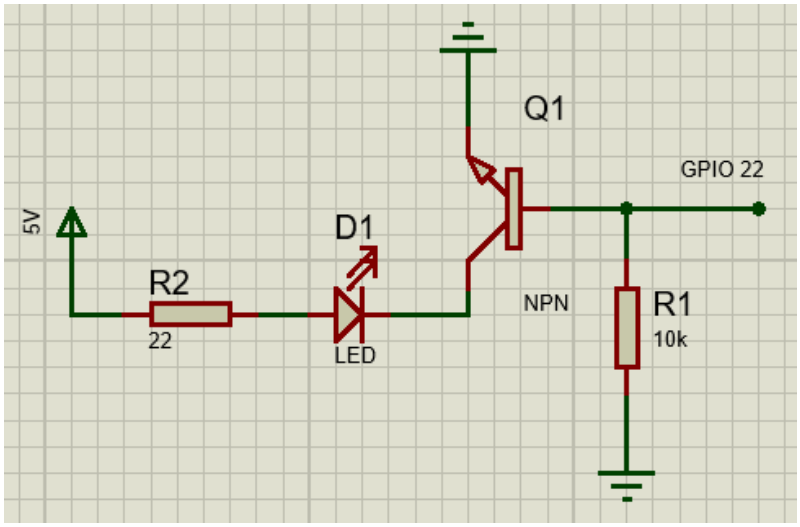
```
sudo /etc/init.d/lirc/stop
```

```
irrecord -d /dev/lirc0 ~/lircd.conf
```

Quá trình trên kết thúc là ta đã ghi lại được các tín hiệu trên remote và lưu lại trong thư viện LIRC, cụ thể là trong file **~/lirc.conf.**

6.2.2 Phát tín hiệu hồng ngoại và điều khiển Relay

Để phát tín hiệu hồng ngoại từ Raspberry, ta sử dụng led phát hồng ngoại, để phát tín hiệu được xa hơn ta sử dụng thêm một transistor để khuếch đại tín hiệu. Dưới đây là sơ đồ mạch phát tín hiệu hồng ngoại:



Hình 6.3 Sơ đồ mạch phát tín hiệu hồng ngoại

GPIO 22 của Raspberry sẽ là đầu ra của tín hiệu hồng ngoại, được nối vào cực B của transistor. Hai led hồng ngoại sử dụng là loại 5mm.

Để phát một tín hiệu ra ngoài module phát, ta sử dụng lênh **irsend:**

```
irsend SEND_ONCE REMOTE_NAME CONTROL
```

Ở đây REMOTE_NAME là tên remote đặt ban đầu, CONTROL là phím cần phát đi mà đã đặt tên lúc thu tín hiệu. Khi lệnh trên thực hiện, module sẽ phát tín hiệu hồng ngoại giống với tín hiệu trên remote với phím tương ứng.

Đối với module relay, kết nối chân tín hiệu vào của relay với chân GPIO 24 của Raspberry, hai chân nguồn và đất nối tương ứng với chân nguồn 5v và chân đất của Raspberry.

6.2.3 Xử lý dữ liệu từ Server

Dữ liệu từ Server gửi đến Raspberry là chuỗi JSON chứa thông tin lệnh của người dùng, bao gồm 2 loại lệnh: **lệnh điều khiển relay** và **lệnh điều khiển tivi.**

❖ ***Lệnh điều khiển relay được gửi thông qua chuỗi JSON có dạng:***

```
{"Status":1}
```

trong đó 1 tương ứng với trạng thái bật, 0 tương ứng trạng thái tắt. Nhiệm vụ của Raspberry là bắt sự kiện từ phía Server và giải mã chuỗi JSON.

```
socket.on("APP_ESP", function(data){
     var status = data.Status;
})
```

Khi giải mã được chuỗi JSON và lấy được trạng thái của relay, ta cần xuất tín hiệu ứng với trạng thái tương ứng ra GPIO của module relay và thư viện ***onoff*** trong Nodejs hỗ trợ điều này.

```
var Gpio = require('onoff').Gpio;
var replay = new Gpio(24, 'out');
```

Để xuất tín hiệu 1 ra chân relay ta dùng câu lệnh:

```
replay.writeSync(1);
```

Từ đó ta có thể xây dựng đoạn mã lắng nghe sự kiện từ Server và điều khiển module relay như sau:

```
socket.on("APP_ESP", function(data){
```

```
    console.log("Mobile app send control: " + JSON.stringify(data));
    var status = data.Status;
    replay.writeSync(status);
})
```

❖ ***Lệnh điều khiển tivi được gửi thông qua chuỗi JSON có dạng:***

```
{"remote": NAME_TV, "code": VOLUMEUP}
```

trong đó thông số ***code*** là thông tin lệnh từ người dùng. Để lắng nghe sự kiện từ Server và giải mã chuỗi JSON ta dùng:

```
socket.on("APP_RASPI", function(data){
    console.log("Mobile app send data: " + JSON.stringify(data));
    IRSend(data.code);
})
```

Ở đây, hàm ***IRSend(data)*** có chức năng điều khiển module phát hồng ngoại tương ứng với dữ liệu ***data*** truyền vào. Để điều khiển được module phát hồng ngoại, ta cần sử dụng thư viện LIRC với câu lệnh:

```
irsend SEND_ONCE REMOTE_NAME CONTROL
```

Dựa vào lệnh phát tín hiệu hồng ngoại từ thư viện LIRC, ta xây dựng được hàm IRSend với chức năng tương tự:

```
function TCL_TV(code){
    if(isNaN(code)){//data is KEY_CONTROL
        if(code == VOLUME_UP || code == VOLUME_DOWN)
            IRSendVolume(code);
        else{
```

```
            lirc.irsend.send_once("TCL_TV", "KEY_" + code, function() {
                console.log("Sent TCL_TV KEY_" + code);
              });
        }
    }else{ //data is KEY_CHANNEL
        IRSendChannel(code);
    }
}
//Send KEY_CHANNEL to lirc
function IRSendChannel(data, index = 0){
    lirc.irsend.send_once("TCL_TV", "KEY_" + data[index]);
    console.log("Sent TCL_TV KEY_" + data[index]);
    index++;
    if(index < data.length){
        setTimeout(function(){
            IRSendChannel(data, index);
        }, TIME_PRESS_MILI);
    }
}
```

6.3 Kết luận

Khối thực thi với trung tâm điều khiển là máy tính nhúng Raspberry Pi có chức năng điều khiển các module thu phát hồng ngoại, module relay thực hiện cho việc điều khiển tivi. Kết nối giữa Raspberry và Server luôn được đảm bảo theo thời gian thực, ổn định. Việc sử dụng Raspberry là thiết bị điều khiển giúp cho quá trình thu tín hiệu hồng ngoại từ remote một cách chính xác nhờ thư viện LIRC hỗ trợ tốt cho hệ điều hành Linux.

Raspberry Pi đã đáp ứng được yêu cầu của hệ thống khi điều khiển được các module hồng ngoại và relay hoạt động đúng yêu cầu đề ra. Tín hiệu hồng ngoại phát ra khớp với tín hiệu trên remote, việc xử lý dữ liệu từ Server được thực hiện chính xác, đúng với yêu cầu của người dùng.

Hệ thống vẫn tồn tại nhiều hạn chế, một trong hạn chế lớn nhất là việc thu tín hiệu hồng ngoại từ remote. Hiện tại, khi muốn điều khiển một loại tivi thì cần phải thu tín hiệu remote vào Raspberry, việc thu tín hiệu được thực hiện trên Raspberry. Để có thể áp dụng hệ thống cho các loại tivi khác nhau, chúng ta cần có dữ liệu remote của từng chủng loại tivi, từ đó hạn chế được việc thu trực tiếp từ remote.

CHƯƠNG 7. KẾT LUẬN CHUNG

7.1 Kết quả đạt được

Hệ thống điều khiển tivi bằng giọng nói sau khi đưa vào thử nghiệm đã thu được một số kết quả nhất định. Ứng dụng trên điện thoại di động giúp người dùng có thể điều khiển tivi của mình một cách dễ dàng ở bất cứ đâu thông qua mạng internet. Ứng dụng cho phép người dùng ra lệnh bằng giọng nói tiếng Việt với một số câu lệnh cơ bản như: ***"bật/tắt tivi"***, ***"Bật kênh VTV1"***, ***"tăng/giảm âm lượng"***,…Ngoài ra ứng dụng còn cho phép người dùng thêm hoặc sửa các kênh tivi yêu thích của mình.

Chức năng ra lệnh thông qua nền tảng Google Assistant hoạt động đúng như thiết kế, người dùng có thể ra lệnh qua trợ lý Google Assistant thông qua các thiết bị hỗ trợ như Google Home hoặc điện thoại có Google Assistant. Để ra lệnh thông qua Google Assistant, người dùng phải sử dụng tiếng Anh, một số mẫu câu người dùng có thể nói như: ***"turn on/off tivi"***, ***"volume up/down"***, ***"turn on 25 channel"***,..

7.2 Những điều chưa làm được

Hệ thống hiện tại đang còn những hạn chế nhất định, một trong những hạn chế lớn nhất đó là việc cài đặt hệ thống còn phức tạp. Để hệ thống có thể hoạt động một cách chính xác, người dùng cần thu được tín hiệu hồng ngoại của tivi thông qua Raspberry, điều này gây khó khăn cho người dùng.

Chức năng ra lệnh bằng giọng nói tiếng Việt trên điện thoại hoạt động còn chậm trong điều kiện môi trường có nhiều tiếng ồn. Người dùng khi ra lệnh phải nói theo đúng cú pháp quy định của ứng dụng, số lượng mẫu câu lệnh còn hạn chế.

Ngoài ra, hệ thống chưa tính năng quản lý người dùng theo tài khoản. Đây là một tính năng quan trọng giúp người dùng có thể quản lý được thiết bị của mình, đồng thời hệ thống có thể phục vụ cho nhiều người dùng khác mà chỉ cần một Server.

7.3 Phương hướng phát triển tiếp theo

Hệ thống hiện tại mới dừng lại ở chức năng điều khiển một một thiết bị duy nhất là tivi do đó hướng phát triển tiếp theo của đề tài là nâng cấp hệ thống thành một hệ

thống điều khiển ngôi nhà thông minh toàn diện. Cụ thể, hệ thống sẽ được tích hợp thêm các thiết bị khác trong ngôi nhà như bóng đèn, quạt, điều hòa,..và những thiết bị được điều khiển bằng sóng hồng ngoại khác. Ngoài ra, các cảm biến cũng sẽ được tích hợp vào trong hệ thống giúp người dùng kiểm soát tình trạng của ngôi nhà một cách chính xác.

Hệ thống trong tương lai sẽ được phát triển theo hướng phục vụ cho nhiều người dùng, quản lý người dùng theo tài khoản, từ đó người dùng sẽ có thể quản lý được thiết bị của mình một cách dễ dàng. Ứng dụng trên điện thoại cũng sẽ được sửa lại phù hợp với yêu cầu của hệ thống mới.

TÀI LIỆU THAM KHẢO

[1] *http://vietjack.com/nodejs/nodejs_la_gi.jsp*, truy cập cuối ngày 10/5/2018.

[2] *https://viblo.asia/p/websocket-la-gi-Ljy5VxkbZra*, truy cập cuối ngày 10/5/2018.

[3] *https://raspberrypi.vn/tin-tuc/raspberry-pi-la-gi-gioi-thieu-ve-raspberry-pi-261.pi*, truy cập cuối ngày 20/5/2018.

[4] *https://vi.wikipedia.org/wiki/Google_Assistant*, truy cập cuối ngày 10/5/2018.

[5] *https://en.wikipedia.org/wiki/Google_Home*, truy cập cuối ngày 20/5/2018.

[6] *https://developers.google.com/actions/dialogflow/*, truy cập cuối ngày 20/5/2018.

[7] *http://alexba.in/blog/2013/01/06/setting-up-lirc-on-the-raspberrypi/*, truy cập cuối ngày 10/5/2018.

PHỤ LỤC

Phụ lục 1. Mã nguồn Nodejs của Server

```
const PORT = 3000;

const ENTITIES = "channel";

const NAME_TV = "TCL_TV";

var ip = require('ip');

var express = require("express");

var exp = express();

var app = require("http").createServer(exp);


var socketio = require('socket.io');

var io = socketio(app);


var esp8266 = io.of('/esp8266'); //Namespace của Esp8266

var mobileApp = io.of('/mobileApp'); //Namespace của Mobile App

var raspi = io.of('/raspi'); //Namespace của Raspberry


//Update entities Dialogflow

var request = require('request');


//Get post request from Dialogflow

var bodyParser = require("body-parser");


app.listen(process.env.PORT || PORT);

console.log("Server running at IP: " + ip.address() + ":" + PORT);


/**
```

```
 * Bắt sự kiện khi esp8266 kết nối tới
 */
esp8266.on('connection', function(socket){
    console.log("Esp8266 connected!");
    //Khi Esp gửi cho app trạng thái của replay
    socket.on("ESP_APP", function(data){
        console.log("Esp send to App: " + JSON.stringify(data));
        mobileApp.emit("ESP_APP", data);
    })
    //Khi esp8266 ngắt kết nối
    socket.on('disconnect', function(){
        console.log("Esp8266 disconnected!");
    })
})

/**
 * Bắt sự kiện khi Mobile App kết nối tới
 */
mobileApp.on('connection', function(socket){
    console.log("Mobile app connected!");
    //Sự kiện App gửi control power cho Esp8266
    socket.on("APP_ESP", function(data){
        console.log("App send to Esp8266: " + JSON.stringify(data));
        esp8266.emit("APP_ESP", data);
    })
    //Sự kiện App gửi cho Raspberry
    socket.on("APP_RASPI", function(data){
```

```
        console.log("App send to Raspberry: " + JSON.stringify(data));
        raspi.emit("APP_RASPI", data);
    })

    //Sự kiện App gửi chuỗi Json chứa số kênh
    socket.on("CHANNEL", function(data){
        console.log("App send Dialogflow list channel: " + JSON.stringify(data)
);
        UpdateEntities(data);
    })

    //Sự kiện App gửi yêu cầu update tới Esp
    socket.on("UPDATE", function(){
        console.log("App request update to Esp");
        esp8266.emit("UPDATE", "");
    })

    //Khi Mobile app ngắt kết nối
    socket.on('disconnect', function(){
        console.log("Mobile app disconnected!");
    })
})

/**
 * Bắt sự kiện khi Raspberry kết nối tới
 */
raspi.on('connection', function(socket){
```

```
    console.log("Raspberry connected!");
    //Khi Raspberry ngắt kết nối
    socket.on('disconnect', function(){
        console.log("Raspberry disconnected!");
    })
})

//Update entities cho Dialogflow
function UpdateEntities(data){
    // Set the headers
    var headers = {
        'Authorization':    'Bearer b90129cb0a5a4b568bc2e960e1397168',
        'Content-Type':     'application/json'
    }
    //Json to update entities, if not create -> create new entities
    var bodyJson =
    [
        {
            "entries": data,
            "name": ENTITIES
        }
    ]
    //Sap xep lai bodyJson
    var body = JSON.stringify(bodyJson);
    // Configure the request
    var options = {
        url: 'https://api.dialogflow.com/v1/entities?v=20150910&lang=en',
```

```
        method: 'PUT',
        headers: headers,
        body: body
    }
    // Start the request
    request(options, function (error, response, body) {
        if (!error && response.statusCode == 200) {
            // Print out the response body
            console.log("Dialogflow response: " + JSON.stringify(body));
        }
    })
}

//Get post request from Dialogflow
exp.use(bodyParser.json({extended: true}));
exp.post("/api", function(request, response){
    var channel = request.body.result.parameters.channel;
    var controltv = request.body.result.parameters.controltv;
    console.log("Dialogflow send channel: " + channel);
    console.log("Dialogflow send control: " + controltv);
    if(channel.length != 0)
        raspi.emit("APP_RASPI", {"remote": NAME_TV, "code": channel});
    if(controltv.length != 0){
        var code = "";
        switch(controltv){
            case "turn on tv":
                code = "POWER";
```

```
                break;
            case "turn off tv":
                code = "POWER";
                break;
            case "volume up":
                code = "VOLUMEUP";
                break;
            case "volume down":
                code = "VOLUMEDOWN";
                break;
            case "channel up":
                code = "CHANNELUP";
                break;
            case "channel down":
                code = "CHANNELDOWN";
                break;
        }
        raspi.emit("APP_RASPI", {"remote": NAME_TV, "code": code});
    }
})
```

Phụ lục 2. Mã nguồn java của ứng dụng trên điện thoại Android

❖ MainActivity.java

```
package com.example.tientran.bkhome;


import android.app.PendingIntent;

import android.content.Intent;

import android.os.Bundle;

import android.speech.RecognizerIntent;

import android.support.design.widget.FloatingActionButton;

import android.support.v7.app.AlertDialog;

import android.support.v7.app.AppCompatActivity;

import android.support.v7.widget.SwitchCompat;

import android.util.Log;

import android.view.ActionMode;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.widget.AbsListView;

import android.widget.AdapterView;

import android.widget.Button;

import android.widget.CompoundButton;

import android.widget.EditText;

import android.widget.GridView;

import android.widget.ImageView;

import android.widget.ListView;

import android.widget.Switch;

import android.widget.Toast;
```

```
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.net.URISyntaxException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Locale;

import io.socket.client.IO;
import io.socket.client.Socket;
import io.socket.emitter.Emitter;

public class TiviActivity extends AppCompatActivity {

    private static final int NOT_FIND = -1;
    private static final int REQ_CODE_SPEECH_INPUT = 100;

    private FloatingActionButton btnVoice;
    private Switch sw_power;
    private ListView listView;
    private ArrayList<Kenh> arrayChannel = new ArrayList<Kenh>();
    private ArrayList<Kenh> selectedChannel = new ArrayList<Kenh>();
    private KenhAdapter adapter;
    {
```

```
        try {
            mSocket = IO.socket("https://bkhome-test.herokuapp.com/mobileApp");
        } catch (URISyntaxException e) {
        }
    }
    //Array Logo channel
    int[] idIcon = {
            R.drawable.tivi, R.drawable.vtv1, R.drawable.vtv2,
            R.drawable.vtv3, R.drawable.vtv4, R.drawable.vtv5, R.drawable.vtv6,
            R.drawable.vtv7, R.drawable.vtv8, R.drawable.vtv9,
            R.drawable.hanoi1, R.drawable.hanoi2, R.drawable.vinhphuc, R.drawab
le.bibi,
            R.drawable.hbo, R.drawable.thanhhoa
    };
    int idIconAdded = idIcon[0];
    //Save arrayChannel to file
    private SaveChannel saveChannel = new SaveChannel(this);
    private Socket mSocket;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tivi);
        //Emit event update status
        mSocket.emit("UPDATE");
        //Listen even update status power
        mSocket.on("ESP_APP", SetStaus);
```

```
        //Connect to server
        mSocket.connect();

        btnVoice = (FloatingActionButton) findViewById(R.id.btnVoice);
        sw_power = (Switch) findViewById(R.id.swpower);
        listView = (ListView) findViewById(R.id.listView);

        //Check file "ListChannel.txt"
        if (saveChannel.isSaveChannel()) {
            arrayChannel = saveChannel.ReadChannel();
        } else {
            arrayChannel.add(new Kenh("VTV1", "1", R.drawable.vtv1));
            arrayChannel.add(new Kenh("VTV2", "2", R.drawable.vtv2));
            arrayChannel.add(new Kenh("VTV3", "253", R.drawable.vtv3));
            arrayChannel.add(new Kenh("VTV4", "4", R.drawable.vtv4));
            arrayChannel.add(new Kenh("VTV5", "5", R.drawable.vtv5));
            saveChannel.WriteChannel(arrayChannel);
        }
        adapter = new KenhAdapter(TiviActivity.this, R.layout.kenhlv, arrayChan
nel);
        listView.setAdapter(adapter);
        //Longclick on ListView
        listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
        listView.setMultiChoiceModeListener(new AbsListView.MultiChoiceModeList
ener() {
            @Override
            public void onItemCheckedStateChanged(ActionMode mode, int position
, long id, boolean checked) {
```

```
                SlectChannel(position);
            }

            @Override
            public boolean onCreateActionMode(ActionMode mode, Menu menu) {
                mode.getMenuInflater().inflate(R.menu.action_mode, menu);
                return true;
            }

            @Override
            public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
                return false;
            }

            @Override
            public boolean onActionItemClicked(ActionMode mode, MenuItem item)
{
                switch (item.getItemId()) {
                    case R.id.btnDelete:
                        for (Kenh kenh : selectedChannel)
                            arrayChannel.remove(kenh);
                        selectedChannel.clear();
                        adapter.notifyDataSetChanged();
                        mode.finish();
                        saveChannel.WriteChannel(arrayChannel);
                        SendNumberChannelToServer();
                        break;
```

```
                }
                return true;
            }

            //Khi ket thuc ActionMode thi update ListView
            @Override
            public void onDestroyActionMode(ActionMode mode) {
                for (int i = 0; i < arrayChannel.size(); i++)
                    if (arrayChannel.get(i).isChecked())
                        arrayChannel.get(i).setCheck(false);
                adapter.notifyDataSetChanged();
            }
        });
        //Click on ListView
        listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, final int
position, long id) {
                UpdateChannel(position);
            }
        });
        sw_power.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeL
istener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isC
hecked) {
                ControlPower(isChecked);
            }
```

```
        });
    }//end onCreate


    //Set status power for switch
    private Emitter.Listener SetStaus = new Emitter.Listener() {
        @Override
        public void call(final Object... args) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    JSONObject data = (JSONObject)args[0];
                    String status = "";
                    try {
                        status = data.getString("Status");
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                    if(status.equals("1")){
                        sw_power.setChecked(true);
                    }else
                        sw_power.setChecked(false);
                }
            });
        }
    };


    //Send control power to Esp8266
```

```
    private void ControlPower(boolean isChecked){
        JSONObject status = new JSONObject();
        try {
            if(isChecked)
                status.put("Status", 1);
            else
                status.put("Status", 0);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        mSocket.emit("APP_ESP", status);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.action_bar, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.btnAdd:
                AddChannel();
                adapter.notifyDataSetChanged();
                break;
            case R.id.btnSortName:
```

```
                Collections.sort(arrayChannel, ComparatorNameChannel);
                adapter.notifyDataSetChanged();
                saveChannel.WriteChannel(arrayChannel);
                break;
            case R.id.btnSortNum:
                Collections.sort(arrayChannel, ComparatorNumChannel);
                adapter.notifyDataSetChanged();
                saveChannel.WriteChannel(arrayChannel);
                break;
        }
        return super.onOptionsItemSelected(item);
    }

    //onClick Add menu
    public void AddChannel() {
        AlertDialog.Builder builder = new AlertDialog.Builder(TiviActivity.this
);
        final View customLayout = getLayoutInflater().inflate(R.layout.add_dial
og, null);
        builder.setView(customLayout);
        final AlertDialog dialog = builder.create();
        //Ánh xạ
        final EditText tenKenh = (EditText) customLayout.findViewById(R.id.edtT
enKenh);
        final EditText soKenh = (EditText) customLayout.findViewById(R.id.edtSo
Kenh);
        final ImageView imageIconAddDialog = (ImageView) customLayout.findViewB
yId(R.id.imageAddDialog);
```

```
        Button selectIconAddDialog = (Button) customLayout.findViewById(R.id.bt
nIconAddDialog);
        Button addChannel = (Button) customLayout.findViewById(R.id.btnAddDialo
g);
        imageIconAddDialog.setImageResource(R.drawable.tivi);
        //Sự kiện ấn Thêm
        addChannel.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String kenh = tenKenh.getText().toString().trim().toUpperCase()
;
                String so = soKenh.getText().toString().trim().toUpperCase();
                if (kenh.equals("") || so.equals(""))
                    Toast.makeText(TiviActivity.this, "Nhập thông tin kênh", To
ast.LENGTH_SHORT).show();
                else if (SearchChannel(kenh, so, arrayChannel))
                    Toast.makeText(TiviActivity.this, "Kênh đã tồn tại", Toast.
LENGTH_SHORT).show();
                else {
                    arrayChannel.add(new Kenh(kenh, so, idIconAdded));
                    idIconAdded = idIcon[0];
                    saveChannel.WriteChannel(arrayChannel); //save channel
                    SendNumberChannelToServer();
                    dialog.dismiss();
                }
            }
        });
        //Su kien khi an SelectIcon
```

```
        selectIconAddDialog.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SelecIconAddDialog(imageIconAddDialog);
            }
        });
        dialog.show();
    }

    //Display add Icon on Dialog
    public void SelecIconAddDialog(final ImageView imageView) {
        AlertDialog.Builder builder = new AlertDialog.Builder(TiviActivity.this
);
        final View customLayout = getLayoutInflater().inflate(R.layout.activity
_icon, null);
        builder.setView(customLayout);
        final AlertDialog dialog = builder.create();
        IconAdapter adapter = new IconAdapter(TiviActivity.this, idIcon);
        GridView grid = (GridView) customLayout.findViewById(R.id.grid);
        grid.setAdapter(adapter);
        grid.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int posit
ion, long id) {
                idIconAdded = idIcon[position];
                imageView.setImageResource(idIconAdded);
                dialog.dismiss();
            }
```

```
        });
        dialog.show();
    }

    //Display update icon on Dialog
    public void SelecIconUpdateDialog(final ImageView imageView, final int posi
tionArrayList) {
        AlertDialog.Builder builder = new AlertDialog.Builder(TiviActivity.this
);
        final View customLayout = getLayoutInflater().inflate(R.layout.activity
_icon, null);
        builder.setView(customLayout);
        final AlertDialog dialog = builder.create();
        IconAdapter adapter = new IconAdapter(TiviActivity.this, idIcon);
        GridView grid = (GridView) customLayout.findViewById(R.id.grid);
        grid.setAdapter(adapter);
        grid.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int posit
ion, long id) {
                arrayChannel.set(positionArrayList, new Kenh(arrayChannel.get(p
ositionArrayList).nameChannel,
                        arrayChannel.get(positionArrayList).numberChannel,
                        idIcon[position]));
                imageView.setImageResource(idIcon[position]);
                dialog.dismiss();
            }
        });
```

```
        dialog.show();
    }


    //Sort name channel
    public static Comparator<Kenh> ComparatorNameChannel = new Comparator<Kenh>
() {
        public int compare(Kenh app1, Kenh app2) {
            Kenh channel1 = app1;
            Kenh channel2 = app2;
            return channel1.nameChannel.compareTo(channel2.nameChannel);
        }
    };
    //Sort number channel
    public static Comparator<Kenh> ComparatorNumChannel = new Comparator<Kenh>(
) {
        public int compare(Kenh app1, Kenh app2) {
            Kenh channel1 = app1;
            Kenh channel2 = app2;
            return Integer.valueOf(channel1.numberChannel).compareTo(Integer.va
lueOf(channel2.numberChannel));
        }
    };


    //Put channel selected on SelectChannel array
    public void SlectChannel(int position) {
        if (arrayChannel.get(position).isChecked()) {
            arrayChannel.get(position).setCheck(false);
            selectedChannel.remove(arrayChannel.get(position));
```

```
        } else {
            arrayChannel.get(position).setCheck(true);
            selectedChannel.add(arrayChannel.get(position));
        }
        adapter.notifyDataSetChanged();
    }

    //Update channel when click on listview
    public void UpdateChannel(final int position) {
        AlertDialog.Builder builder = new AlertDialog.Builder(TiviActivity.this
);
        View customLayout = getLayoutInflater().inflate(R.layout.update_dialog,
null);
        builder.setView(customLayout);
        final AlertDialog dialog = builder.create();
        //Ánh xạ
        final EditText tenKenh = (EditText) customLayout.findViewById(R.id.edtT
enKenh);
        final EditText soKenh = (EditText) customLayout.findViewById(R.id.edtSo
Kenh);
        Button update = (Button) customLayout.findViewById(R.id.btnUpdate);
        final ImageView imageIconUpdateDialog = (ImageView) customLayout.findVi
ewById(R.id.imageUpdateDialog);
        Button selectIconUpdateDialog = (Button) customLayout.findViewById(R.id
.btnIconupdateDialog);
        tenKenh.setText(arrayChannel.get(position).nameChannel);
        soKenh.setText(arrayChannel.get(position).numberChannel);
        imageIconUpdateDialog.setImageResource(arrayChannel.get(position).idIma
ge);
```

```
        update.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String kenh = tenKenh.getText().toString().trim().toUpperCase()
;
                String so = soKenh.getText().toString().trim().toUpperCase();
                if (kenh.equals("") || so.equals(""))
                    Toast.makeText(TiviActivity.this, "Nhập thông tin kênh", To
ast.LENGTH_SHORT).show();
                else if (SearchChannelExceptPosition(kenh, so, arrayChannel, po
sition))
                    Toast.makeText(TiviActivity.this, "Kênh đã tồn tại", Toast.
LENGTH_SHORT).show();
                else {
                    arrayChannel.set(position, new Kenh(kenh, so, arrayChannel.
get(position).idImage));
                    adapter.notifyDataSetChanged();
                    saveChannel.WriteChannel(arrayChannel);//save channel
                    SendNumberChannelToServer();
                    dialog.dismiss();
                }
            }
        });
        selectIconUpdateDialog.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SelecIconUpdateDialog(imageIconUpdateDialog, position);
            }
```

```
        });
        dialog.show();
    }

    //Cilck on Voice button
    public void VoiceClick(View v) {
        Intent voiceIntent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        voiceIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
        voiceIntent.putExtra(RecognizerIntent.EXTRA_PROMPT,"Ra lệnh để điều khiển tivi");
        voiceIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, "vi");
        startActivityForResult(voiceIntent, REQ_CODE_SPEECH_INPUT);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        switch (requestCode) {
            case REQ_CODE_SPEECH_INPUT: {
                if (resultCode == RESULT_OK && null != data) {
                    ArrayList<String> result = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
                    try {
                        processResult(result.get(0));
                    } catch (JSONException e) {
```

```
                        e.printStackTrace();
                    }
                }
                break;
            }
        }
    }

    //Ham tim kiem kenh
    private boolean SearchChannel(String nameChannel, String numberChannel, Arr
ayList<Kenh> arrayList) {
        for (int i = 0; i < arrayList.size(); i++)
            if (numberChannel.equalsIgnoreCase(arrayList.get(i).numberChannel)
||
                    nameChannel.equalsIgnoreCase(arrayList.get(i).nameChannel))
        return true;
        return false;
    }

    //Ham tim kiem kenh loai tru mot vi tri
    private boolean SearchChannelExceptPosition(String nameChannel, String numb
erChannel, ArrayList<Kenh> arrayList, int position) {
        int i = 0;
        while (i < arrayList.size() && i != position) {
            if (numberChannel.equalsIgnoreCase(arrayList.get(i).numberChannel)
||
                    nameChannel.equalsIgnoreCase(arrayList.get(i).nameChannel))
                return true;
```

```
            i++;
        }
        return false;
    }

    //Ham put so kenh trong arrayChannel sang arrayJson va gui len server
    private void SendNumberChannelToServer(){
        JSONArray jsonArray = new JSONArray();
        for(int i = 0; i < arrayChannel.size(); i++){
            JSONObject jsonObject = new JSONObject();
            try {
                jsonObject.put("value", arrayChannel.get(i).numberChannel);
            } catch (JSONException e) {
                e.printStackTrace();
            }
            jsonArray.put(jsonObject);
        }
        //Emit len server
        mSocket.emit("CHANNEL", jsonArray);
    }

    //Xu ly data voice
    private void processResult(String textResult) throws JSONException {
        String textResultUpCase = textResult.trim().toUpperCase();
        Toast.makeText(this, "" + textResultUpCase, Toast.LENGTH_SHORT).show();
        JSONObject data = new JSONObject();
        data.put("remote", getString(R.string.TCL_TV));
```

```
        switch (textResultUpCase) {
            case "TĂNG ÂM LƯỢNG":
                data.put("code", getString(R.string.volume_up));
                mSocket.emit(getString(R.string.app_emit), data);
                break;
            case "GIẢM ÂM LƯỢNG":
                data.put("code", getString(R.string.volume_down));
                mSocket.emit(getString(R.string.app_emit), data);
                break;
            case "TĂNG KÊNH":
                data.put("code", getString(R.string.channel_up));
                mSocket.emit(getString(R.string.app_emit), data);
                break;
            case "GIẢM KÊNH":
                data.put("code", getString(R.string.channel_down));
                mSocket.emit(getString(R.string.app_emit), data);
                break;
            case "BẬT TIVI":
                data.put("code", getString(R.string.power));
                mSocket.emit(getString(R.string.app_emit), data);
                break;
            case "TẮT TIVI":
                data.put("code", getString(R.string.power));
                mSocket.emit(getString(R.string.app_emit), data);
                break;
            case "KÊNH":
                break;
```

```
            default:
                //Find index of "KENH" in textResultUpCase
                int lastIndex = textResultUpCase.lastIndexOf("KÊNH");
                //Get name channel in textResultUpCase
                String resultChannel = textResultUpCase.substring(lastIndex + 5
);
                int resultIndex = SearchIndexChannel(resultChannel, arrayChanne
l);
                if (resultIndex != NOT_FIND) {
                    data.put("code", arrayChannel.get(resultIndex).numberChanne
l);
                    mSocket.emit(getString(R.string.app_emit), data);
                    Toast.makeText(this, "OK, bật kênh " + resultChannel, Toast
.LENGTH_SHORT).show();
                } else
                    Toast.makeText(this, "Không tìm thấy kênh", Toast.LENGTH_SH
ORT).show();
        }
    }

    /**
     * Find name channel or number channel in arrayList
     * Return NOT_FIND if not find
     */
    private int SearchIndexChannel(String channel, ArrayList<Kenh> arrayList) {
        for (int i = 0; i < arrayList.size(); i++) {
            if (channel.equals(arrayList.get(i).nameChannel)
                    || channel.equals(arrayList.get(i).numberChannel))
```

```
                return i;
            }
            return NOT_FIND;
        }

        @Override
        protected void onDestroy() {
            super.onDestroy();
            if(mSocket != null)
                mSocket.disconnect();
        }
    }
```

❖ **SaveChannel.java**

```
package com.example.tientran.bkhome;


import android.content.Context;

import android.util.Log;

import android.widget.Toast;


import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.OutputStreamWriter;

import java.io.Serializable;

import java.util.ArrayList;


/**

 * Created by tientran on 27/03/2018.

 */


public class SaveChannel {

    Context context;


    public SaveChannel(Context context) {

        this.context = context;

    }
```

```
    //Ham ghi arraylist vao file "ListChannel"
    public void WriteChannel(ArrayList<Kenh> arrayList) {
        try {
            FileOutputStream fileOutputStream = context.openFileOutput("ListCha
nnel.txt", context.MODE_PRIVATE);
            ObjectOutputStream objectOutputStream = new ObjectOutputStream(file
OutputStream);
            objectOutputStream.writeObject(arrayList);
            objectOutputStream.close();
            fileOutputStream.close();
        } catch (FileNotFoundException e) {
        } catch (IOException e) {
        }
    }

    //Ham doc arraylist tu file "ListChannel"
    //Return ArrayList
    public ArrayList<Kenh> ReadChannel() {
        ArrayList<Kenh> result = null;
        FileInputStream fileInputStream = null;
        ObjectInputStream objectInputStream = null;
        try {
            fileInputStream = context.openFileInput("ListChannel.txt");
            objectInputStream = new ObjectInputStream(fileInputStream);
            result = (ArrayList<Kenh>) objectInputStream.readObject();
            Log.d("ReadChannel", result.toString());
            objectInputStream.close();
```

```
            fileInputStream.close();
        } catch (FileNotFoundException e) {
        } catch (IOException e) {
        } catch (ClassNotFoundException e) {
        }
        return result;
    }

    //Ham kiem tra file "ListChannel" ton tai chua
    //Return true neu ton tai
    //Return false neu chua ton tai
    public boolean isSaveChannel() {
        try {
            FileInputStream fileInputStream = context.openFileInput("ListChanne
l.txt");
            fileInputStream.close();
        } catch (FileNotFoundException e) {
            return false;
        } catch (IOException e) {
            e.printStackTrace();
        }
        return true;
    }
}
```

❖ KenhAdapter.java

```
package com.example.tientran.bkhome;


import android.content.Context;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.BaseAdapter;

import android.widget.ImageView;

import android.widget.TextView;


import java.util.List;


/**

 * Created by tientran on 07/03/2018.

 */


public class KenhAdapter extends BaseAdapter {

    Context context;

    int layout;

    List<Kenh> arrayChannel;


    public KenhAdapter(Context context, int layout, List<Kenh> arrayChannel) {

        this.context = context;

        this.layout = layout;

        this.arrayChannel = arrayChannel;

    }
```

```
    @Override
    public int getCount() {
        return arrayChannel.size();
    }

    @Override
    public Object getItem(int position) {
        return null;
    }

    @Override
    public long getItemId(int position) {
        return 0;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater) context.getSystemService(Con
text.LAYOUT_INFLATER_SERVICE);
        convertView = inflater.inflate(layout, null);
        //anh xa va gan gia tri
        TextView nameChannel = (TextView)convertView.findViewById(R.id.tvTenKen
h);
        TextView numberChannel = (TextView)convertView.findViewById(R.id.tvSoKe
nh);
        ImageView imageView = (ImageView)convertView.findViewById(R.id.imageVie
w);
```

```
        nameChannel.setText(arrayChannel.get(position).nameChannel);
        numberChannel.setText(arrayChannel.get(position).numberChannel);
        //Set icon cho kenh
        if(arrayChannel.get(position).isChecked())
            imageView.setImageResource(R.drawable.checked);
        else
            imageView.setImageResource(arrayChannel.get(position).idImage);
        return convertView;
    }
}
```

❖ **Kenh.java**

```
package com.example.tientran.bkhome;


import java.io.Serializable;


/**
 * Created by tientran on 07/03/2018.
 */


public class Kenh implements Serializable{
    public String nameChannel;
    public String numberChannel;
    private boolean Checked = false;
    public int idImage;


    public Kenh(String nameChannel, String numberChannel, int idImage) {
        this.nameChannel = nameChannel;
        this.numberChannel = numberChannel;
        this.idImage = idImage;
    }
    public boolean isChecked() {
        return Checked;
    }
    public void setCheck(boolean check){
        Checked = check;
    }
}
```

❖ IconAdapter.java

```
package com.example.tientran.bkhome;


import android.content.Context;

import android.util.Log;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.BaseAdapter;

import android.widget.ImageView;

import android.widget.TextView;


import java.util.List;


/**
 * Created by tientran on 13/03/2018.
 */


public class IconAdapter extends BaseAdapter {
    private Context mContext;
    private final int[] Imageid;


    public IconAdapter(Context c,int[] Imageid ) {
        mContext = c;
        this.Imageid = Imageid;
    }
```

```
    @Override
    public int getCount() {
        return Imageid.length;
    }

    @Override
    public Object getItem(int position) {
        return null;
    }

    @Override
    public long getItemId(int position) {
        return 0;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View grid;
        LayoutInflater inflater = (LayoutInflater) mContext.getSystemService(Co
ntext.LAYOUT_INFLATER_SERVICE);

        if (convertView == null) {
            grid = new View(mContext);
            grid = inflater.inflate(R.layout.grid_item, null);
            ImageView imageView = (ImageView)grid.findViewById(R.id.grid_image)
;
            imageView.setImageResource(Imageid[position]);
        } else {
```

```
            grid = (View) convertView;
        }
        return grid;
    }
}
```

Phụ lục 3. Mã nguồn Nodejs của Raspberry Pi

```
const TIME_PRESS_MILI = 500;

const NUMBER_PRESS_VOLUME = 5;

const VOLUME_UP = "VOLUMEUP";

const VOLUME_DOWN = "VOLUMEDOWN";

var lirc = require('lirc_node');

var io = require('socket.io-client')

var socket = io('https://bkhome-test.herokuapp.com/raspi');

lirc.init();


//Listen event connet to Server

socket.on('connect', function(){

    console.log("Connected to Server!");

})


//Listent event: App send to Raspi

socket.on("APP_RASPI", function(data){

    console.log("Mobile app send data: " + JSON.stringify(data));

    IRSend(data);

})

//Listent event: App send to Esp8266

socket.on("UPDATE", function(){

    console.log("Mobile app send data: UPDATE");

    var status = replay.readSync();

    socket.emit("ESP_APP", {"Status": status});

})
```

```
socket.on("APP_ESP", function(data){
    console.log("Mobile app send control: " + JSON.stringify(data));
    var status = data.Status;
    replay.writeSync(status);
})

//Listent event: Raspi disconnect server
socket.on("disconnect", function(){
    console.log("Disconnected to Server!");
})

/**
 * Process data from Server send to
 * @param {JSON} data: example {'remote': 'TCL_TV', 'code': 'VOLUMEUP'}
 */
function IRSend(data){
     switch(data.remote){
              case "TCL_TV":
                       TCL_TV(data.code);
              //Other remote:
     }
}
/**
 * Process data of remote TCL_TV
 * @param {String} code : code of TCL_TV remote
 */
function TCL_TV(code){
```

```
     if(isNaN(code)){//data is KEY_CONTROL
              if(code == VOLUME_UP || code == VOLUME_DOWN)
                       IRSendVolume(code);
              else{
                       lirc.irsend.send_once("TCL_TV", "KEY_" + code, function()
{
                                console.log("Sent TCL_TV KEY_" + code);
                         });
              }
     }else{ //data is KEY_CHANNEL
              IRSendChannel(code);
     }
}

//Send KEY_CHANNEL to lirc
function IRSendChannel(data, index = 0){
     lirc.irsend.send_once("TCL_TV", "KEY_" + data[index]);
     console.log("Sent TCL_TV KEY_" + data[index]);
     index++;
     if(index < data.length){
              setTimeout(function(){
                       IRSendChannel(data, index);
              }, TIME_PRESS_MILI);
     }
}

//Send KEY_VOLUME_xxx to lirc
```

```
function IRSendVolume(data, index = 0){
     lirc.irsend.send_once("TCL_TV", "KEY_" + data);
     console.log("Sent TCL_TV KEY_" + data);
     index++;
     if(index < NUMBER_PRESS_VOLUME){
              setTimeout(function(){
                       IRSendVolume(data, index);
              }, TIME_PRESS_MILI);
     }
}
```