UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Otávio Henrique Gotardo Piton

AUTOMAÇÃO RESIDENCIAL UTILIZANDO A PLATAFORMA EM NUVEM IBM BLUEMIX

São Carlos

2017

Otávio Henrique Gotardo Piton

AUTOMAÇÃO RESIDENCIAL UTILIZANDO A PLATAFORMA EM NUVEM IBM BLUEMIX

Trabalho de conclusão de curso apresentado à escola de engenharia de São Carlos da Universidade de São Paulo.

Curso de Engenharia elétrica com ênfase em sistemas de energia e automação

Orientador: Prof. Dennis Brandão

São Carlos

2017

Piton, Otávio Henrique Gotardo
P681a Automação residencial utilizando a plataforma em nuvem IBM Bluemix / Otávio Henrique Gotardo Piton; orientador Dennis Brandão. São Carlos, 2017.

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2017.

1. Internet das coisas. 2. Computação em nuvem. 3. Automação residencial. 4. IBM Bluemix. 5. Protocolo MQTT. 6. Controle de voz. I. Título.

FOLHA DE APROVAÇÃO

Nome: Otávio Henrique Gotardo Piton

Título: "Automação residencial utilizando a plataforma em nuvem IBM Bluemix"

Trabalho de Conclusão de Curso defendido e aprovado em 28 / 11 / 17,

com NOTA 9,0 (nove, zero), **pela Comissão Julgadora:**

Prof. Associado Dennis Brandão - Orientador - SEL/EESC/USP

Prof. Associado Evandro Luis Linhari Rodrigues - SEL/EESC/USP

Mestre Guilherme Serpa Sestito - Doutorando - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

RESUMO

PITON, O. H. G. **Automação residencial utilizando a plataforma em nuvem IBM Bluemix**. 2017. 63p. Trabalho de conclusão de curso - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

O objetivo deste trabalho foi a implementação de um sistema de automação residencial utilizando plataformas em nuvem, o IBM Bluemix foi utilizado como base para a sua implementação. O sistema foi capaz de simular o controle de iluminação, do alarme, informar a temperatura do ambiente, trancar e destrancar a porta. O controle é feito via internet pelo *smartphone* através de painel de comando em um aplicativo ou via controle de voz e pode ser realizado de qualquer lugar do mundo com acesso a internet. O Projeto também possui um sistema de conversação sendo capaz de criar um diálogo simples com o usuário, como cumprimentos e despedidas ou até mesmo responder aos comandos. Foi realizada a implementação do sistema utilizando o IBM Bluemix com os serviços Watson Conversation para a conversação, Node-Red para o processamento de informações e comunicação entre os seviços e hardware, Cloudant para o armazenamento de informações e Weather Insights para informações sobre o clima. A comunicação entre os serviços em nuvem, o *smartphone* e o hardware foi feita via protocolo MQTT. O hardware utilizado foi uma placa de desenvolvimento Wemos D1 R2 com componentes conectados a ela para simular os equipamentos controlados. O sistema atingiu sua motivação de agregar praticidade, segurança e simplicidade à tarefas realizadas diariamente pelos usuários.

Palavras-chave: Internet das coisas. Computação em nuvem. Automação residencial. IBM Bluemix. Protocolo MQTT. Controle de voz.

ABSTRACT

PITON, O. H. G. **Home automation using the IBM Bluemix cloud plataform**. 2017. 63p. Trabalho de conclusão de curso - Escola de Engenharia de São Carlos, Universidade de São Paulo,São Carlos, 2017.

The goal of this paper was the implementation of a home automation system using cloud platforms, IBM Bluemix was the basis for the implementation. The system was able to simulate the lighting control, alarm control, information about the ambient temperature and lock and unlock the door, the control is done via internet by smartphone through control panel in an application or via voice control and can be done from anywhere in the world with internet access. The system also has a conversation system able to create a simple dialog with the user, such compliments and goodbyes or even respond to commands. The system implementation was made using IBM Bluemix with the services Watson Conversation for conversation, Node Red for processing information and communication between the services and hardware, Cloudant for information storage and Weather Insights for information about the weather. The communication between the cloud services, smartphone and hardware has been done via MQTT protocol. The hardware used was a development card Wemos D1 R2 with components attached to it to simulate the equipment controlled. The system has reached its motivation to add practicality, security and simplicity to the tasks done daily by users.

Keywords: Internet of things. Cloud computing. Home Automation. IBM Bluemix. MQTT Protocol. Voice control.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

IBM	International Business Machines
PaaS	Platform as a Service
MQTT	Message Queuing Telemetry Transport
M2M	Machine to machine
TCP	Transmission Control Protocol
IP	Internet Protocol
API	Application Programming Interface
QoS	Quality of Service
USB	Universal Serial Bus
WPA	*Wi-Fi* Protected Access
GPS	Global Positioning System
IoT	Internet of Things
RAM	Random Access Memory
UIMA	Unstructured Information Management Architecture
DBaaS	Database as a Service
JSON	JavaScript Object Notation
Led	Light Emiting Diode

SUMÁRIO

1 INTRODUÇÃO

1.1 Automação residencial

A automação residencial, também conhecida como domótica, é a automatização e o controle aplicados à residência. Estes se realizam mediante o uso de equipamentos que dispõem de capacidade para se comunicar interativamente entre eles e com capacidade de seguir as instruções de um programa previamente estabelecido pelo usuário da residência e com possibilidades de alterações conforme seus interesses. Em consequência, a domótica permite maior qualidade de vida, reduz o trabalho doméstico, aumenta o bem-estar e a segurança, racionaliza o consumo de energia e, além disso, sua evolução permite oferecer continuamente novas aplicações (MURATORI; Bó, 2011).

Como se percebe, o principal fator que define uma instalação residencial automatizada é a integração entre os sistemas aliada à capacidade de executar funções e comandos mediante instruções. A integração pode abranger todos os sistemas tecnológicos da residência,(MURATORI; Bó, 2011) como:

- Instalação elétrica, que compreende: iluminação, persianas e cortinas, gestão de energia e outros;
- Sistema de segurança: alarmes de intrusão, alarmes técnicos (fumaça, vazamento de gás, inundação), circuito fechado de TV, monitoramento, controle de acesso;
- Sistemas multimídia: áudio e vídeo, som ambiente, jogos eletrônicos, além de vídeos, imagens e sons sob demanda;
- Sistemas de comunicações: telefonia e interfonia, redes domésticas, TV por assinatura;
- Utilidades: irrigação, aspiração central, climatização, aquecimento de água, bombas e outros.

Atualmente é possível desenvolver sistemas de automação residencial com um alto nível de controle utilizando sistemas embarcados, que estão cada vez mais potentes. Utilizando também a internet ou sistema de telefonia móvel que é cada vez mais utilizado com a grande popularização dos *smartphones*.

Os sistemas de iluminação das residências têm como base a utilização de interruptores, com a automação é esperado que a utilização destes seja reduzida drasticamente e que todo o controle seja feito a partir de controles no *smartphone* ou por controles de voz com o principal objetivo de aumentar o conforto dos moradores.

Os sistemas de alarme e segurança residenciais sempre foram desejados pelos moradores, já que melhoram a segurança da residência, mas atualmente com os índices de violência se elevando cada vez mais estes sistemas vêm ganhando cada vez mais procura e mercado.

Outro diferencial de sistemas que utilizam a internet como base para automação residencial é que existe a possibilidade da realização de controles de qualquer local do mundo com acesso a internet. Mesmo durante uma viagem ou durante o trabalho é possível controlar sua residência e com a instalação de câmeras este sistema poderia ficar ainda mais preciso e interessante.

A proposta deste trabalho é o controle de processos residenciais simples e recorrentes, como o controle de iluminação, alarmes e outros equipamentos, por meio do *smartphone*. Será utilizado como base para o controle dos processos e a comunicação entre os equipamentos a plataforma em nuvem do IBM Bluemix.

1.2 Internet das coisas

O conceito da internet das coisas é basicamente conectar qualquer dispositivo com um botão *ON/OFF* a internet, isto inclui tudo, *smartphones*, cafeteiras, máquinas de lavar, interruptores e também se aplica a componentes de máquinas. Qualquer dispositivo com um botão *ON/OFF* pode ser parte da internet das coisas. Segundo a Gartner, empresa de tecnologia da informação, pesquisas e consultoria, em 2020 serão mais de 26 bilhões de dispositivos conectados (alguns estimam este número em mais de 100 bilhões). Uma rede gigante de "coisas" conectadas (que incluem também pessoas), assim como a relação entre pessoa-pessoa, pessoa-coisas e coisas-coisas (MORGAN, 2014).

As aplicações são diversas e englobam o monitoramento da saúde de um indivíduo, o controle de um sistema de automação e o uso de dispositivos pessoais conectados (MARTINS, 2017).

A nova lei do futuro será, "Qualquer coisa que pode ser conectada, será conectada". Em uma escala mais ampla, há possibilidade de aplicações em redes de transporte, "cidades inteligentes" podem nos ajudar a reduzir desperdícios e aumentar a eficiência ajudando a entender e aprimorar como vivemos e trabalhamos. A realidade é que infinitas oportunidades e possibilidades são permitidas e muitas delas não podemos nem mesmo pensar ou entender completamente o impacto que resultaria hoje (MORGAN, 2014).

Discussões sobre a internet das coisas estão acontecendo por todo o mundo e procurando entender como isso poderá impactar vidas. Também busca-se entender como as muitas oportunidades e desafios ocorrerão quando mais e mais dispositivos começarem a se juntar a rede. Por enquanto estudar e informar-se sobre o que é e seu potencial de impactar como vivemos e trabalhamos é o mais possível de ser realizado (MORGAN, 2014).

Comunicação é a parte central da internet das coisas. Tecnologias de rede habilitam dispositivos a se comunicar com outros dispositivos, aplicações e serviços que estão rodando na nuvem. A internet depende de protocolos padronizados para garantir que a comunicação entre dispositivos diferentes ocorra com segurança e confiáveis. Estes protocolos especificam as regras e formatos que dispositivos usam para estabilizar e gerenciar as redes, assim como a transmissão de informação entre as redes (GERBER, 2017).

No Brasil já existe a discussão de uma politica nacional voltada para o desenvolvimento

do mercado de internet das coisas, trata-se do Plano Nacional de Internet das Coisas, que visa nortear ações politicas e públicas para o setor. A ABB, empresa de tecnologia de energia e automação, já trabalha neste segmento de internet das coisas com o "ABB Ability^{TM}" que está sendo desenvolvido sobre a plataforma em nuvem Azure da Microsoft, assim um sistema que antes era centralizado em uma única localidade possa ser colaborativo, possibilitando que diversas pessoas em lugares diferentes possam trabalhar juntas em um mesmo projeto (MARTINS, 2017).

A Philips, empresa voltada à tecnologia e produtos de consumo, já conta com lâmpadas inteligentes, a Philips HUE com elas é possível acender lâmpadas por aplicativo no celular, escolher a intensidade da luz, acor da iluminação ou podem ser configuradas com temporizador (MARTINS, 2017).

O ecossistema de internet das coisas no Brasil deve movimentar mais de 13 bilhões de dólares e o mercado brasileiro deve dobrar até 2020 (MARTINS, 2017).

1.3 Objetivo

O objetivo deste trabalho é desenvolver aplicações de automação e também armazenar o estado das aplicações em nuvem e obter informações do funcionamento das aplicações, utilizando para isso as plataformas em nuvem do IBM Bluemix.

Existe o intuito de se desenvolver exemplos de aplicações como controle de iluminação ou equipamentos por meio de comandos ou ações automáticas e sistemas de segurança. Com a finalidade de gerar às pessoas mais praticidade, facilidade, segurança e conforto.

2 EMBASAMENTO TEÓRICO

Neste capítulo será apresentada a fundamentação teórica para o desenvolvimento deste projeto, como componentes, plataformas e serviços utilizados que necessitam de um aprofundamento teórico a respeito deles.

2.1 IBM Bluemix

IBM Bluemix é a oferta de nuvem mais recente da IBM. Permite que as organizações e os desenvolvedores criem, implementem e gerenciem aplicativos de maneira fácil e rápida.

Similares ao IBM Bluemix existem por exemplo o Amazon AWS e o Microsoft Azure. Provavelmente estes outros serviços também seriam suficientes para a realização do mesmo projeto. A escolha do Bluemix foi feita já que foi possível conseguir uma licença de pesquisa para a utilização do serviço.

O IBM Bluemix é composto por um painel apresentado na Figura 1 de onde é possível acessar todos seus serviços e plataformas, além de um catálogo onde é possível configurar novos serviços e criar novos aplicativos. Entre os serviços encontrados no Bluemix existem serviços de infraestrutura, cognitivos e de internet das coisas. O IBM Bluemix, utilizado juntamente com a plataforma Node-Red , foi a base para este trabalho, a parte central de todas as aplicações, que realiza tanto ações de controle como comunicação entre todos os dispositivos.

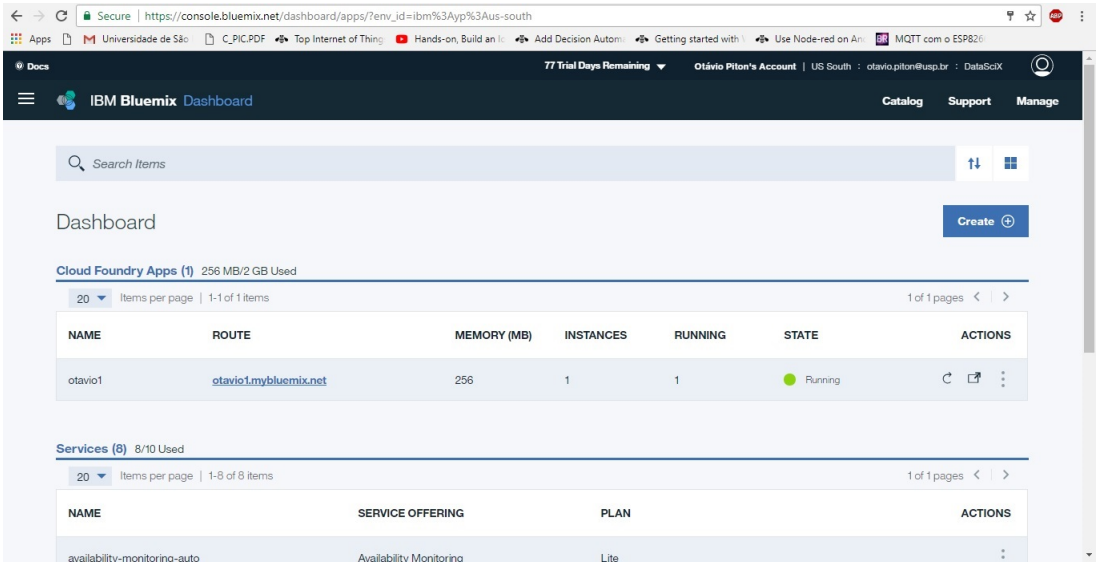


Figura 1: Painel do IBM Bluemix

O Bluemix é uma implementação da Arquitetura de Nuvem Aberta da IBM baseada em *Cloud Foundry*, uma plataforma como serviço (PaaS) de código aberto (REYES, 2014).

Cloud Foundry é uma plataforma como serviço de código aberto que permite criar e

implementar aplicativos rapidamente na nuvem. Devido às suas raízes de código aberto, o *Cloud Foundry* não é específico para o provedor e não o limita a softwares de propriedade intelectual ou infraestrutura de nuvem. O *Cloud Foundry* extrai a infraestrutura de código implícita da nuvem para realizar a operação, permitindo se concentrar mais no desenvolvimento de aplicativos e menos na codificação (REYES, 2014).

O IBM Bluemix oferece serviços de nível básico e empresarial de que as organizações precisam para que seus aplicativos estejam prontos e disponíveis para quando e onde seus clientes mais necessitarem. Devido às suas tecnologias de código aberto implícitas, o IBM Bluemix oferece flexibilidade para integrar desenvolvimento e serviços que se adaptem às suas necessidades (REYES, 2014).

2.1.1 IBM Watson

O IBM Watson é um supercomputador que combina inteligência artificial e software analítico para oferecer serviços diversos. O supercomputador foi nomeado em homenagem ao fundador da IBM, Thomas J. Watson. Ele tem uma taxa de processamento de 80 *teraflops* (um trilhão de operações com ponto flutuante por segundo), na tentativa de replicar ou ultrapassar um cérebro humano (ROUSE, 2016).

Os componentes principais do IBM Watson são: Apache UIMA (*Unstructured Information Management Architecture*), estrutura necessária para análise de informações dispersas; Apache's Hadoop uma estrutura de programação baseada em Java que suporta processamento de grandes quantidades de informações em um ambiente de computação distribuída; SUSE Enterprise Linux Server 11, o mais rápido sistema operacional disponível; 2880 núcleos de processador; 15 terabytes de RAM; 50 gigabytes de informação processada; IBM's DeepQA software, designado para aquisição informação que incorpora processamento de linguagem natural e aprendizado de máquina (ROUSE, 2016).

O Watson anteriormente contava com um serviço de perguntas e repostas (*IBM Watson Question Answer*) que foi impossibilitado de novas criações em 20 de novembro de 2015 e totalmente retirado de funcionamento em 16 de dezembro de 2015. E esta função deixada para os serviços de Natural Language Classifier, Conversation, Retrieve and Rank e Document Conversion, que são serviços que incluem o poder e a flexibilidade de dar respostas a perguntas, entretanto para aplicações mais específicas. Neste projeto o serviço do IBM Watson mais utilizado é o Watson Conversation.

2.1.1.1 Watson Conversation

Com o serviço Watson Conversation, é possível construir um diálogo que entende linguagem natural e utiliza o aprendizado de máquina para responder quem utiliza de forma a simular uma conversação entre humanos (MILLER, 2017).

Funciona da seguinte maneira, o usuário interage com a aplicação por uma interface que pode ser uma janela de chat ou uma interface de voz, a aplicação envia a informação de entrada do usuário para o serviço, a este conecta-se um *workspace* (área de trabalho) que contem

o diálogo e informações de treinamento, o serviço interpreta a entrada e encontra a resposta correspondente, a aplicação pode agora com a informação adquirida interagir novamente com o sistema de interface e realizar ações determinadas.

A Figura 2 mostra o painel do Watson Conversation onde ficam os *workspaces* e dentro de cada um destes todas as informações referentes aos diálogos.

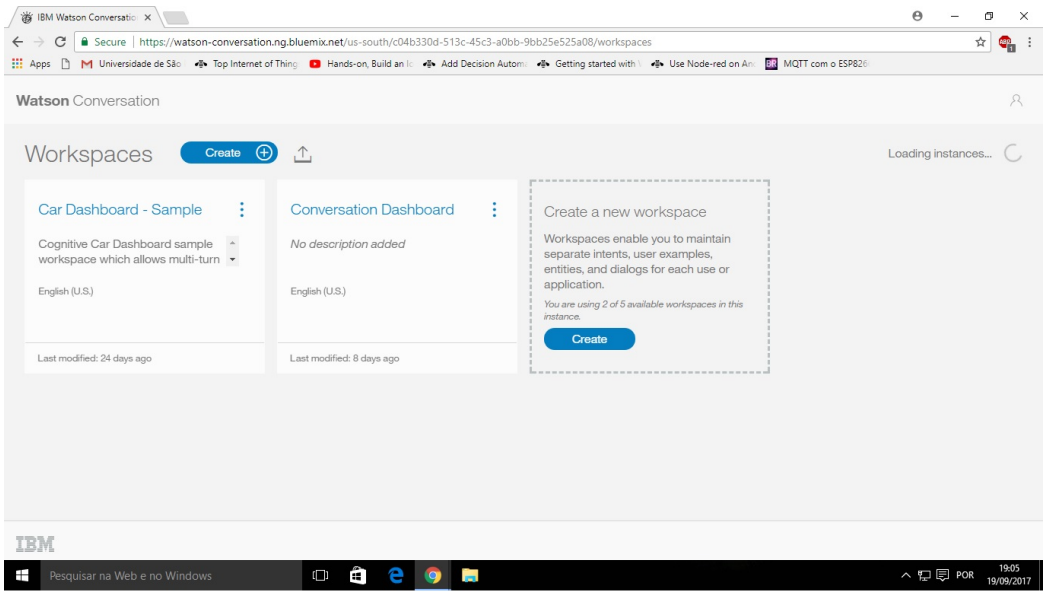


Figura 2: Painel do serviço Watson Conversation

Para implementar o serviço é necessário configurar um *workspace*, é possível fazer isto utilizando uma interface gráfica. Em seguida é preciso inserir a informação de treinamento que são chamadas de *Intents* (Intenções) e *Entities* (Entidades). As Intenções definem em que tipo de ação o usuário está interessado, são literalmente as intenções do usuário com o serviço, por exemplo se o usuário está falando sobre o clima ou sobre as luzes. As Entidades são termos que dão contexto a Intenção, por exemplo sobre qual luz o usuário está falando ou sobre o clima de qual dia o usuário está se referindo.

Conforme informações são adicionadas, um qualificador de linguagem natural é adicionado ao *workspace*, e é treinado para entender os tipos de solicitações que estas informações indicam, para que o serviço entenda estas solicitações e responda a elas.

A ferramenta de diálogo é utilizada para construir o diálogo que incorpora as intenções e entidades. O diálogo é representado graficamente como uma árvore, é possível adicionar ramos ao diálogo que correspondem às intenções e deste ramo saírem novos ramos que lidam com possíveis variações desta intenção e estas variações podem ser por exemplo as entidades.

2.1.2 Plataforma Node-Red

Node-Red é uma ferramenta de programação para conectar dispositivos de hardware, APIs, e serviços online de maneiras diferentes e interessantes. Fornece um editor baseado em navegador web que deixa o processo fácil e tem um vasto diretório de nós que podem ser utilizados

com um simples clique (NODERED.ORG, 2017). A Figura 3 mostra a interface da ferramenta com o diretório de nós a esquerda, o editor de *Flows* no centro e a direita abas de informação sobre os nós e de "*Debug*" para testes do *Flow*.

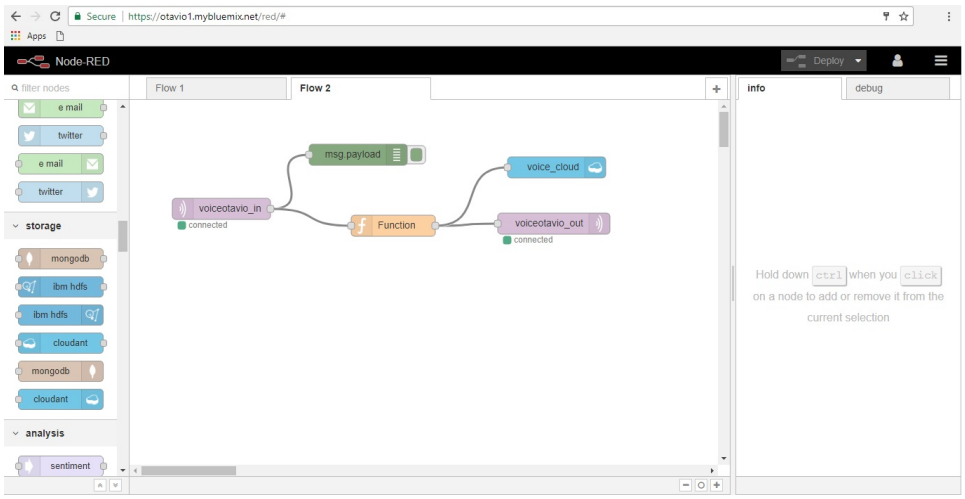


Figura 3: Interface da ferramenta Node-Red

A ferramenta é construída em Node.js que a faz ideal para rodar em hardwares de baixo custo e também na nuvem. Ainda funções em *JavaScript* podem ser criadas dentro do editor utilizando um editor de texto (NODERED.ORG, 2017).

A ferramenta Node-Red, neste projeto, é utilizada para conectar a plataforma em nuvem com os dispositivos de hardware, a comunicação em sua maioria é feita pelo protocolo MQTT, a ferramenta também processa as informações recebidas e envia os comandos correspondentes a tais informações para o hardware.

A Figura 4 apresenta exemplos de nós da ferramenta Node-Red, estes são os nós mais utilizados no projeto.

O nó "inject" que basicamente injeta alguma informação no *flow* quando pressionado, assim sendo útil para fazer testes de como a programação irá se comportar com determinadas entradas.

Os nós "mqtt" tanto de entrada quanto de saída são para comunicação com o *smartphone* e com o hardware. Eles são configurados basicamente com o servidor, o tópico e a QoS. Depois disso já são capazes de receber e transmitir informações respectivamente.

O nó "debug" é um nó de teste utilizado para verificar as informações que fluem e assim conferir se a programação está funcionando corretamente.

O nó "weather insights" é utilizado para obter informações sobre o clima e pode retornar informações sobre o clima no instante atual, de hora em hora, ou de dia em dia. E precisa ser configurado com o intervalo de tempo da previsão e com as coordenadas do local da previsão.

Os nós do IBM Watson são os nós referentes a conversação, o nó "conversation" é o nó utilizado para utilizar o serviço *Watson Conversaion* e precisa ser configurado com o *workspace*

referente ao diálogo predeterminado. O serviço de tradução (*language translator*) também pode ser utilizado para mudar o idioma em que o sistema irá trabalhar.

O nó "function" é utilizado quando é necessário escrever uma função dentro do fluxo de informação para processar a mensagem, a função deve ser escrita em Java. O nó "switch" é utilizado para controlar o fluxo de informação escolhendo para qual caminho a informação deve seguir dependendo do tipo de informação que entrou no nó.

Os nós "cloudant" são utilizados para armazenar informação em nuvem, basicamente armazenam a informação que entrar no nó e esta informação fica disponível para acesso posterior.

Os nós de *dashboard* são nós utilizados para passar e apresentar as informações do sistema em um painel que pode ser acessado via internet.

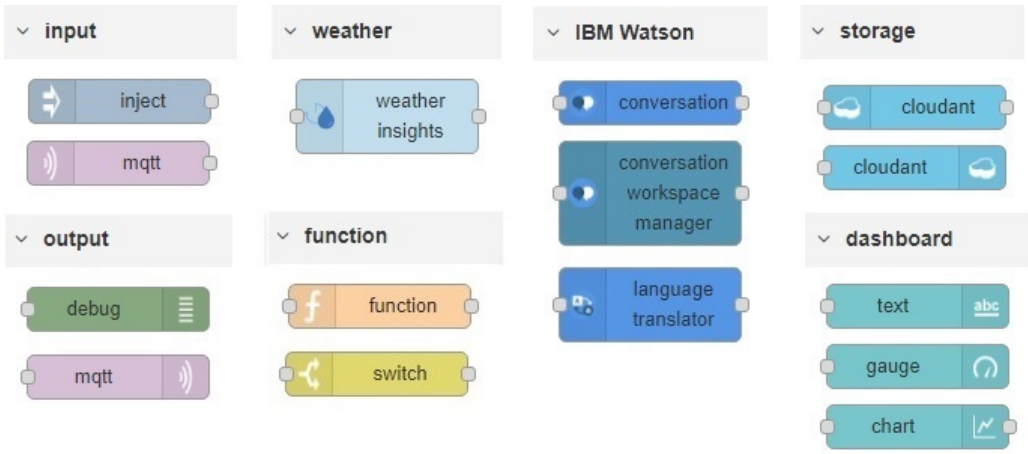


Figura 4: Nós mais utilizados no projeto

2.1.3 Sistema de armazenamento Cloudant

O IBM Cloudant é um sistema de gerenciamento de banco de dados, é nada mais do que um armazenador de documentos JSON totalmente gerenciado para aplicativos móveis e da web progressivos, que mantém acesso e disponibilidade de dados em máxima escala.

É um DBaaS *(Database as a Service)* de documento JSON totalmente gerenciado e otimizado para disponibilidade, durabilidade e mobilidade de dados - perfeito para aplicativos móveis e da web de rápido crescimento. O que torna o Cloudant exclusivo é sua indexação avançada e a capacidade de enviar dados até a extremidade da rede, em múltiplos *datacenters* e dispositivos, para acesso mais rápido e de maior tolerância a falhas. Ele permite que os usuários acessem dados a qualquer momento, em qualquer lugar.

No projeto o Cloudant é utilizado para armazenar em nuvem as informações dos estados das aplicações do sistema.

2.2 Protocolo MQTT

O principal protocolo utilizado para a comunicação entre as aplicações é o MQTT *(Message Queuing Telemetry Transport)*. Principalmente entre o *smartphone* e a plataforma, e entre a placa Wemos D1 R2 e a plataforma. Este protocolo foi utilizado neste projeto devido a sua praticidade de utilização, ser de fácil processamento e ser muito utilizado em internet das coisas.

O protocolo MQTT é um protocolo de mensagem leve e simples baseado em *publish/subscribe* (publicar/se inscrever), os seus princípios são minimizar a largura de banda e os recursos dos dispositivos, enquanto também assegura confiabilidade e segurança de comunicação. Estes princípios o fazem ideal para a comunicação *"machine-to-machine" (M2M)* ou máquina-a-máquina, para a internet das coisas e para aplicações móveis (MQTT.ORG, 2017).

Para os dispositivos da internet das coisas, a conexão com a internet é um requisito. Esta comunicação permite os dispositivos a trabalhar com os outros e com os serviços. O protocolo fundamental da internet é o TCP/IP, e construído neste protocolo temos o MQTT que se tornou o padrão para comunicação na internet das coisas (YUAN, 2017).

O protocolo define dois tipos de indivíduos na rede, um *message broker* e um número de clientes. O *message broker* é um servidor que recebe todas as mensagens dos clientes e encaminha estas mensagens aos clientes destinatários. Um cliente é qualquer dispositivo que pode interagir com o *broker* e receber ou enviar mensagens. Um cliente pode ser um sensor no campo ou uma aplicação na plataforma que processa informação (YUAN, 2017).

As mensagens MQTT são organizadas por *topics* (tópicos), assim existe a flexibilidade de especificar quais clientes tem acesso a certas mensagens. O cliente se conecta ao *broker* e pode se inscrever ou publicar para qualquer tópico do *broker*. Na Figura 5 sensores publicam suas mensagens no tópico de *"sensor_data"* e se inscrevem no tópico *"config_change"*, aplicações de processamento de dados se inscrevem no tópico *"sensor_data"* e um *"Admin Console"* recebe comandos e ajusta as configurações dos sensores e publica no tópico *"config_change"* (YUAN, 2017).

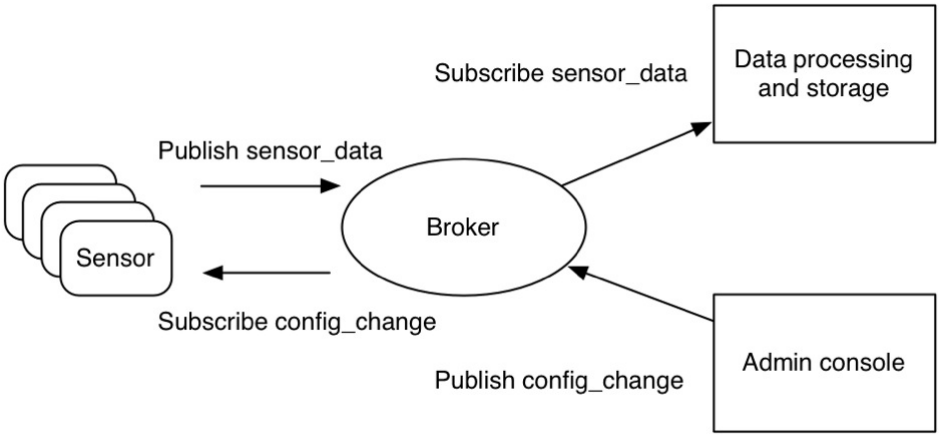


Figura 5: O modelo de *publish/subscribe* para sensores IoT (YUAN, 2017)

3 MATERIAIS E MÉTODOS

Na Figura 6 temos o diagrama de blocos do projeto completo, onde são representados os componentes do projeto. Basicamente o *smartphone*, a placa Wemos D1 R2 e a plataforma Node-Red se comunicam pela internet. A placa Wemos e a plataforma Node-Red ainda se comunicam com seus respectivos periféricos e serviços, para completar o sistema.

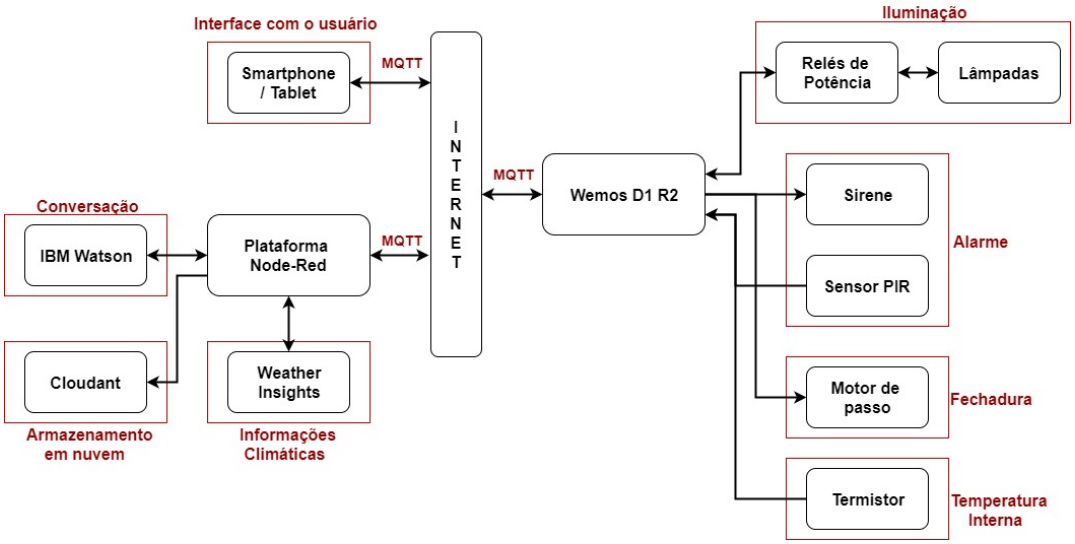


Figura 6: Diagrama de blocos do projeto

As principais funcionalidades do sistema são:

- Controlar as luzes;
- Controlar o alarme;
- Trancar e destrancar a porta;
- Informar a temperatura;
- Informações sobre o clima;
- Criar um diálogo com o usuário;
- Responder sobre capacidades do sistema;

3.1 Aplicação com Node-Red no *smartphone*

Esta aplicação é uma demonstração da plataforma Node-Red e do Bluemix em funcionamento foi baseada no trabalho de Stuart Arnell na página DeveloperWorks da IBM (ARNELL, 2017). Onde o *smartphone* Android se conecta ao Watson IOT via Node-Red e funciona simulando um dispositivo IOT com sensores de temperatura e umidade. E o computador também conectado

ao Watson IOT pelo Node-Red, funcionando como o servidor e neste há um painel que apresenta os estados da simulação feita no *smartphone*.

Para rodar o Node-Red no *smartphone* Android também é necessário ter instalado um aplicativo chamado Termux e também o aplicativo Termux-API. O Termux é um emulador de terminal e ambiente Linux, e na aplicação ele é utilizado para fazer a plataforma Node-Red funcionar dentro do Android. A inicialização não é fácil e requer uma sequência de códigos inseridos no aplicativo. Na Figura 7 é apresentada a interface do aplicativo Termux no Android.

Figura 7: Interface do aplicativo Termux no Android

Depois disso o Node-Red será iniciado no celular e poderá ser acessado pelo navegador acessando *localhost:1880*. Entretanto existe um problema neste método que se o aplicativo for fechado no celular, não será mais possível acessar o Node-Red. E será necessário reabrir o aplicativo e repetir os 2 últimos comandos para inicializar o Node-Red novamente.

Este método apesar de aparentemente promissor e útil se mostrou complicado de se utilizar pela necessidade da utilização do aplicativo Termux e por cada inicialização necessitar de refazer uma parte da codificação para que tudo volte a funcionar.

3.2 Sistema utilizado no *smartphone* para realizar comunicação com a plataforma em nuvem

O *smartphone* é utilizado como interface do usuário com o sistema por meio de comandos de voz, ou por meio de um aplicativo que contém um painel de controle.

3.2.1 Controle de voz via *smartphone*

Primeiramente o *smartphone* Android precisa ser programado para realizar um ciclo de ações, isto pode ser feito utilizando um aplicativo chamado Automate e *plug-ins* necessários. O

ciclo funciona da seguinte maneira:

1. Espera-se até que o *smartphone* receba um comando de voz pelo próprio aplicativo do Google;
2. O comando é enviado à plataforma Node-Red em formato de texto;
3. O texto será analisado e será enviada uma resposta;
4. O *smartphone* espera até que a mensagem de resposta seja recebida e então a reproduz;

O comando de voz precisa ter a palavra chave programada para ser enviado à plataforma em nuvem, se ele não contem esta palavra o *smartphone* reconhece como um comando qualquer para o aplicativo do Google. Neste projeto a palavra chave é “home”, então qualquer comando deve começar com esta palavra, um comando pode ser por exemplo: “home turn on the kitchen lights” ou “home how is the temperature”.

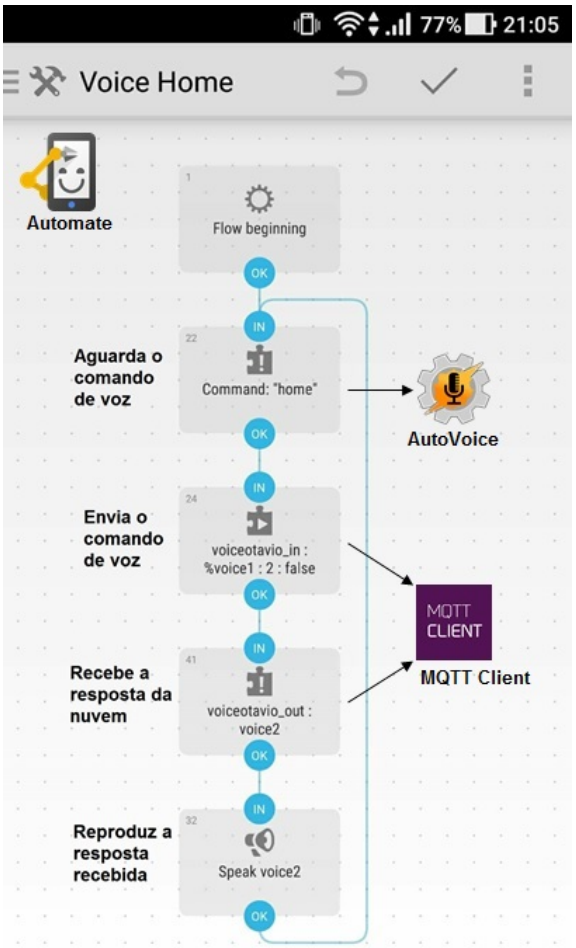


Figura 8: Programação no aplicativo Automate

A Figura 8 contém a programação feita no aplicativo Automate e os respectivos *plug-ins* utilizados, funciona da maneira descrita na imagem. O primeiro bloco é somente uma inicialização. O segundo bloco é responsável por aguardar o comando de voz programado, este bloco chama

um aplicativo de *plug-in* chamado “AutoVoice”, este aplicativo reconhece uma palavra chave falada e então o comando é reconhecido e enviado a plataforma Node-Red.

O terceiro bloco é responsável por enviar o comando de voz reconhecido no bloco anterior para a plataforma Node-Red, isto é feito utilizando um aplicativo de *plug-in* chamado “MQTT Client (Tasker Plugin)”, a mensagem é publicada através deste utilizando o protocolo MQTT para a comunicação.

O quarto bloco aguarda a mensagem em nuvem ser recebida, utilizando o protocolo MQTT e o mesmo *plug-in* “MQTT Client (Tasker Plugin)”. No quinto bloco a resposta recebida no bloco anterior é reproduzida pelo *smartphone* e assim é finalizada a interação com o usuário.

Em alternativa ao Automate seria possível utilizar um aplicativo chamado “Tasker”, este realiza as mesmas funções e tem mais plug-ins que funcionam em conjunto, seu problema é que é pago, e então foi feita a escolha do Automate que é gratuito.

3.2.2 Controle por meio de painel de comando

Para o comando manual do sistema é utilizado um aplicativo chamado “MQTT Dash”, este aplicativo permite criar painéis com funções de botões, barras ou texto, que podem enviar e receber mensagens. Assim é possível apresentar o estado das aplicações e controlá-las no mesmo painel. Na Figura 9 o painel feito no aplicativo que foi utilizado como interface com o sistema é apresentado.

Figura 9: Painel do aplicativo “MQTT Dash”

Cada botão no painel do aplicativo pode ser utilizado para controlar um respectivo

equipamento, este contém comando para as luzes da sala de estar, cozinha e sala de jantar, controle sobre o alarme e a fechadura da porta, além de um mostrador da temperatura indicada pelo hardware.

Outros aplicativos como “IoT MQTT Dashboard” e “My MQTT” realizam ações similares com o “MQTT Dash”, no entanto estes aplicativos não recebem informações e modificam indicadores da maneira necessária e quando recebem informações, se mais de um botão estiver conectado à um mesmo tópico, haverão modificações não esperadas nos estados dos botões, sendo que o “MQTT Dash” consegue fazer da maneira correta.

3.3 Aplicação na plataforma em nuvem

O *smartphone* envia as informações por meio do protocolo MQTT que vão para a nuvem, mais precisamente para a plataforma Node-Red que processa a informação da maneira requerida em conjunto com os serviços Watson Conversation, Weather Insights e Cloudant.

3.3.1 Aplicação na plataforma Node-Red

No Apêndice A está o *flow* completo utilizado no projeto, para fins de apresentação neste capítulo serão apresentadas as partes que compõem o *flow* completo.

A parte inicial do *flow* se encontra na Figura 10, primeiramente a informação chega na plataforma Node-Red por um nó de entrada do protocolo MQTT, o nó “otavio_digital” recebe a informação digital de botões do aplicativo “MQTT Dash”.

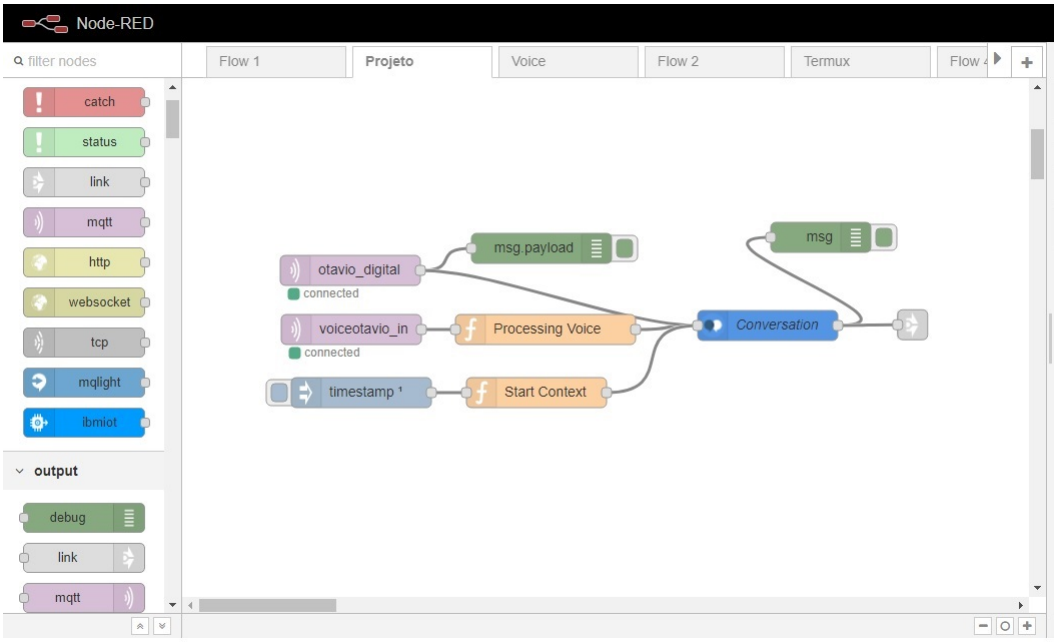


Figura 10: Parte inicial da aplicação em Node-Red

O nó “voiceotavio_in” recebe a informação de voz enviada pelo smartphone que passa por um nó de função para processamento da informação, as informações de ambos os nós são entregues ao serviço Watson Conversation que processa as informações conforme o seu respectivo *workspace*.

Na subseção 3.3.2 a aplicação no Watson Conversation será mais explicada. O nó "timestamp[1]" injeta sinal toda vez que o *flow* é iniciado, para que o nó de função mande informação para o serviço de conversação colocar as variáveis de contexto em seu estado inicial.

Na Figura 11 está a parte do *flow* onde as informações do estado das aplicações são enviadas e armazenadas em nuvem. A cada comando enviado ao serviço de conversação, os estados das aplicações são enviados a nuvem e armazenados, primeiramente passam por um nó de função onde são adicionadas informações como data e hora a mensagem, então as mensagens vão para os nós de "Cloudant" onde são armazenadas. As informações também são apresentadas em gráficos em um painel acessado diretamente no navegador, os blocos de gráfico que estão mais a direita são os responsáveis por enviar estas mensagens.

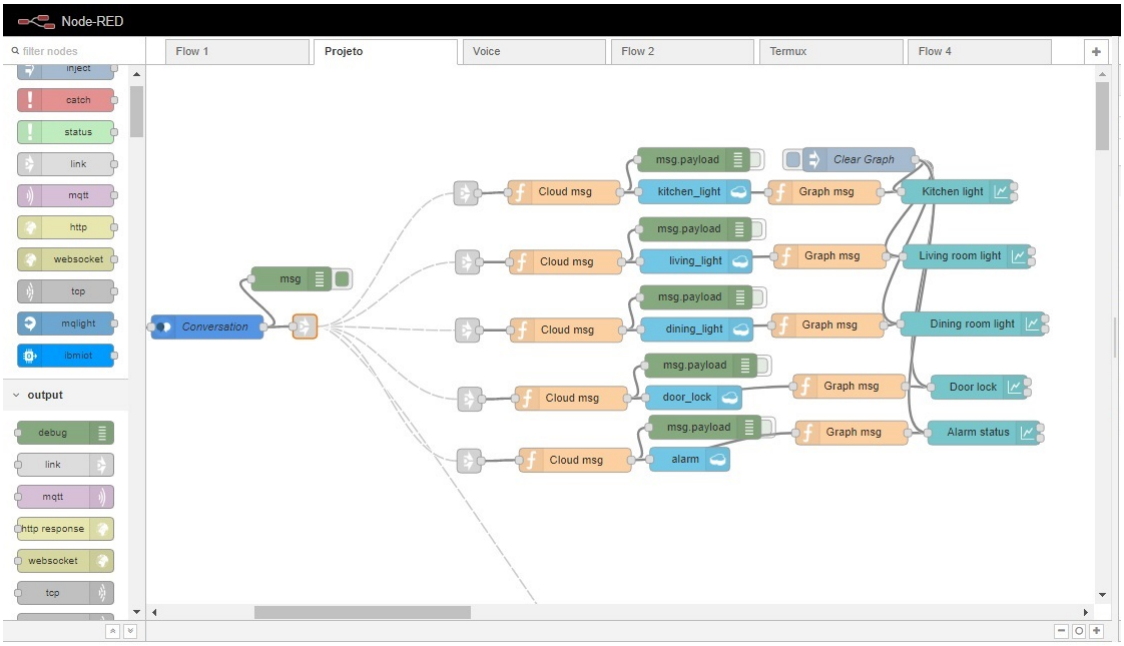


Figura 11: Parte de armazenamento de informações

A Figura 12 mostra o nó "Switch" principal, as informações de saída do serviço de conversação chegam neste nó que direciona cada informação para seu devido caminho. Acontece da seguinte maneira:

1. O serviço de conversação identifica a mensagem do usuário;

2. Armazena em uma variável a função requerida por esta mensagem;

3. Este nó lê esta variável;

4. Então redireciona a mensagem para seu devido fluxo.

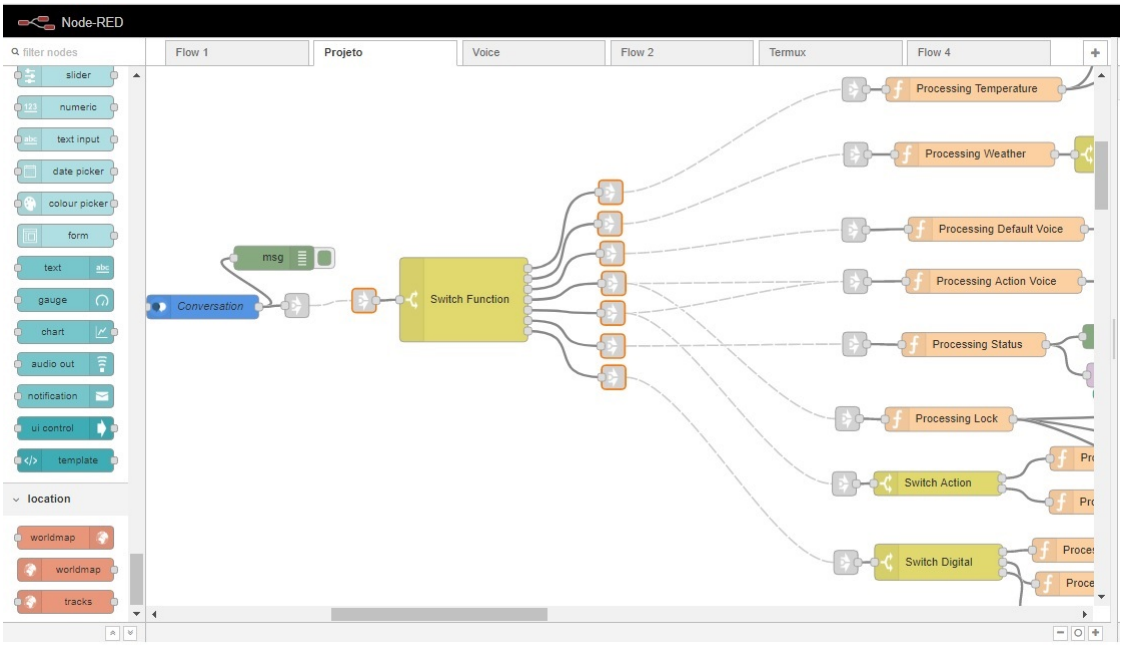


Figura 12: Switch principal

Na Figura 13 é apresentada a parte de processamento das informações, que dependendo da solicitação do usuário algum destes caminhos é tomado e as ações necessárias são realizadas.

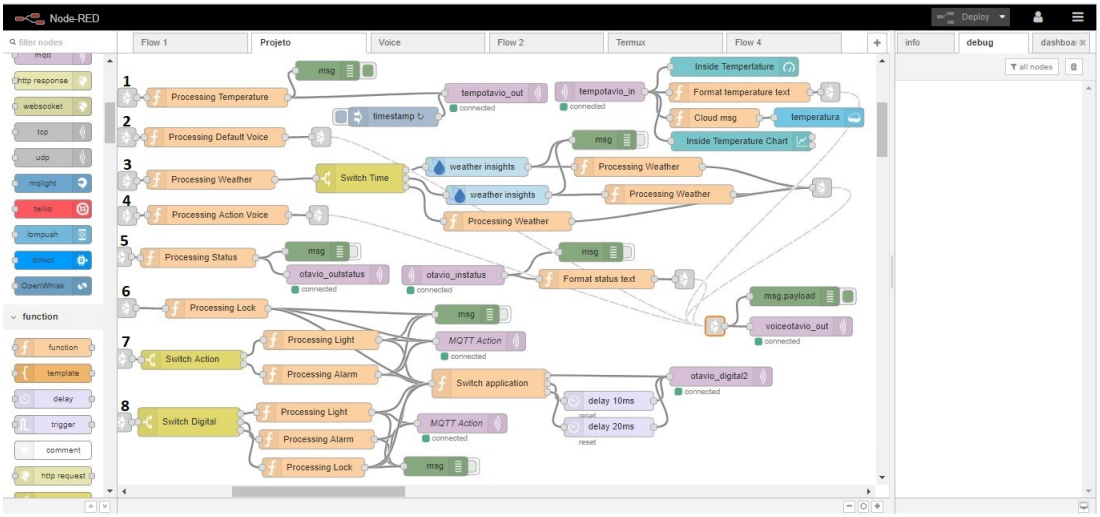


Figura 13: Parte de processamento das informações

O primeiro caminho é referente a solicitação de temperatura interna, nesta função basicamente a plataforma envia uma informação por meio do protocolo MQTT para a placa Wemos D1 R2, então esta retorna um valor de temperatura medido pelo termistor, que é enviado para o painel acessado no navegador, para o *smartphone* e para o armazenamento em nuvem.

O segundo caminho somente processa mensagens de voz padrão, somente para finalidade de diálogo com o usuário. O terceiro caminho processa informações relacionadas a previsão do tempo, o serviço de “Weather insights” é acessado pelos respectivos nós e a informação recebida

é transmitida ao *smartphone* por meio de MQTT.

O quarto caminho é basicamente o processamento de mensagens de voz que serão resposta para o usuário em decorrência de comandos de ação. O quinto caminho processa solicitações de estado das aplicações, quando o usuário perguntar sobre o estado de uma aplicação este caminho será tomado, a plataforma Node-Red envia um pedido para a placa Wemos e esta responde com o estado da aplicação solicitada.

O sexto caminho processa as ações solicitadas por comando de voz para o comando de trancar e destrancar a porta, ele basicamente verifica se o comando foi para trancar ou destrancar a porta e encaminha a respectiva informação para o nó MQTT que envia o comando a placa Wemos.

O sétimo caminho faz basicamente o mesmo que o anterior, mas para ações de luzes e alarme, comandos em que o usuário utiliza as palavras "*turn on*"(ligar) e "*turn off*"(desligar).

O último caminho processa as informações recebidas por meio do aplicativo "MQTT Dash" do *smartphone*, essas informações são consideradas digitais e solicitam as ações digitais de luzes, alarme e fechadura.

3.3.2 Aplicação no serviço Watson Conversation

O serviço Watson Conversation é o principal serviço utilizado, todas as mensagens de comando passam por ele. Este serviço basicamente recebe a mensagem enviada pelo usuário e a responde com outra mensagem para formar um diálogo. Utiliza variáveis de contexto para reconhecer o que foi dito na mensagem e o que precisa ser realizado e repassar para o sistema. No Apêndice B temos a árvore de programação completa utilizada no serviço.

Como apresentado no subseção 2.1.1.1, a primeira parte de reconhecimento das mensagens utiliza as *Intents* ou intenções, na Figura 14 está o painel onde são definidas as intenções do serviço, é possível observar nesta por exemplo as intenções de entradas digitais (#digital_on e #digital_off) , de abrir e trancar a porta (#lock e #unlock) e a intenção de estado (#status).

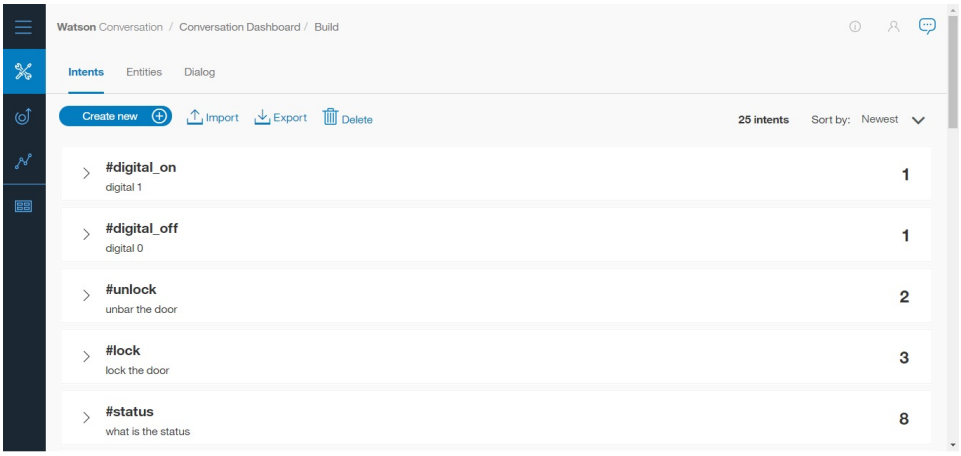


Figura 14: Painel de *Intents* do serviço

A segunda parte de reconhecimento utiliza as *Entities* ou entidades, o painel onde são definidas as entidades do serviço está na Figura 15 , podemos observar nesta imagem as entidades de aplicação (@appliance), local (@place e @place_bad) e tempo (@time).

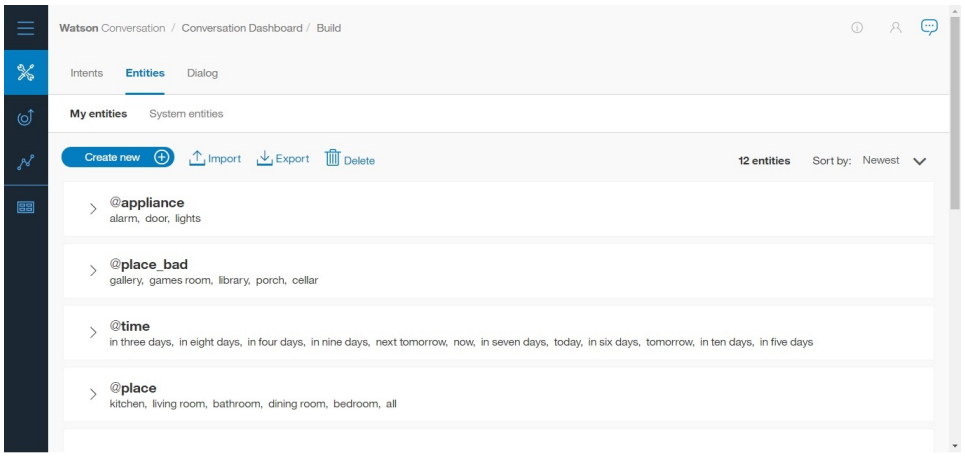


Figura 15: Painel de *Entities* do serviço

Com as intenções e entidades definidas é necessário criar um diálogo utilizando o painel da Figura 16, é possível ver nesta imagem um exemplo de ramo do diálogo. Neste ramo o primeiro nó "#turn_on" reconhece a intenção de ligar algo, então o segundo nó "@appliance:light" reconhece se a aplicação a ser ligada é uma lâmpada, existe também em paralelo com este um nó "@appliance:alarm" que reconhece se a aplicação a ser ligada é o alarme. Se a aplicação for uma lâmpada, o terceiro nó "@place" reconhece o local da lâmpada e então com todas estas informações o serviço responde o usuário com uma mensagem e modifica as variáveis de saída conforme necessário para que a ação seja realizada.

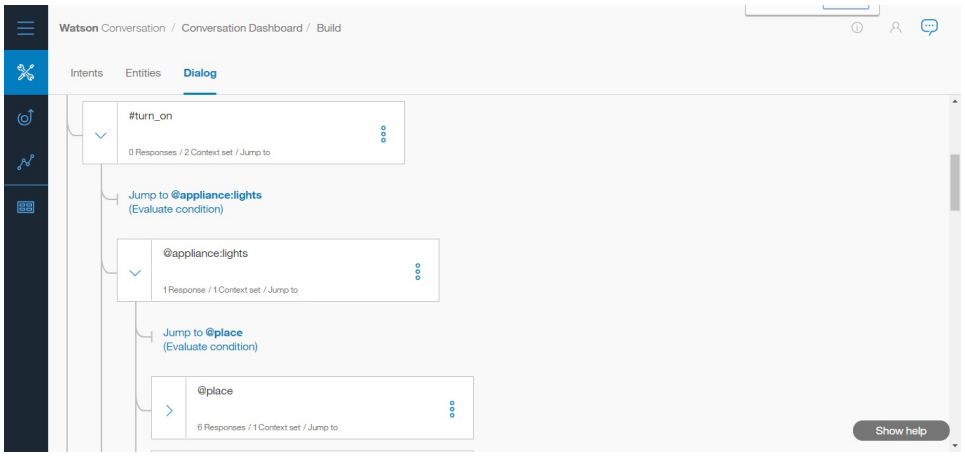


Figura 16: Painel de diálogo do serviço

A Figura 17 contém um exemplo de nó que modifica uma variável, podemos observar que quando a condição de "@place" é satisfeita, se for reconhecida a condição de "@place:kitchen" o serviço modifica a variável de contexto "appl_place" para "kitchen", na saída do serviço esta variável indicará que o local reconhecido e onde a ação deve ser realizada é a cozinha, neste

mesmo bloco temos as condições para outros locais como “@place:living room” e “@place:dining room” que reconhecem respectivamente a sala de estar e a sala de jantar.

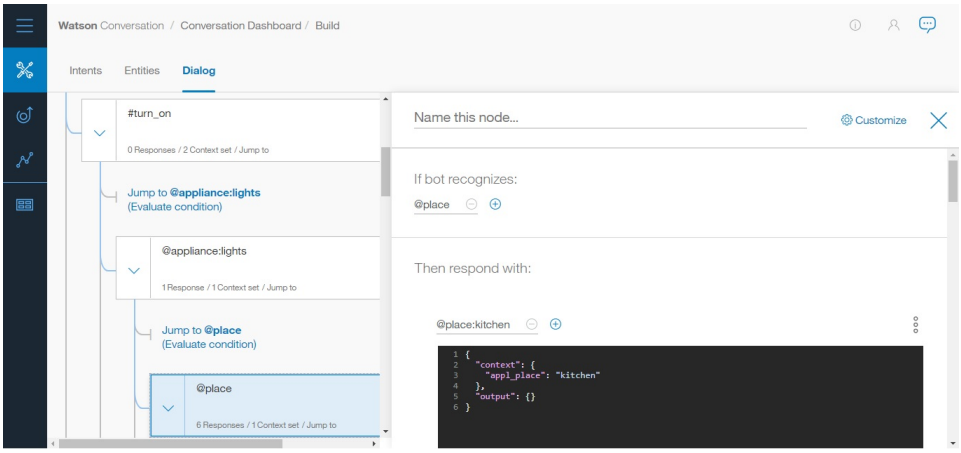


Figura 17: Aba de programação do serviço

Desta mesma maneira são reconhecidas todas as funções do sistema, como ligar e apagar luzes, controlar o alarme, a fechadura, informar a temperatura, informar a previsão climática, responder cumprimentos e despedidas e informar o estado de aplicações.

3.3.3 Armazenamento em nuvem

As informações de estado de todas as aplicações assim como a temperatura medida pelo hardware são armazenados em nuvem. Utiliza-se para isso o sistema de armazenamento Cloudant da IBM que foi melhor apresentado na subseção 2.1.3. A Figura 11 mostra a parte do *flow* do Node-Red que envia as informações para a nuvem.

A Figura 18 mostra as *databases* do serviço, cada uma destas corresponde a um nó de Cloudant da plataforma Node-Red, assim cada um destes *databases* corresponde ao armazenamento dos estados de um aplicação.

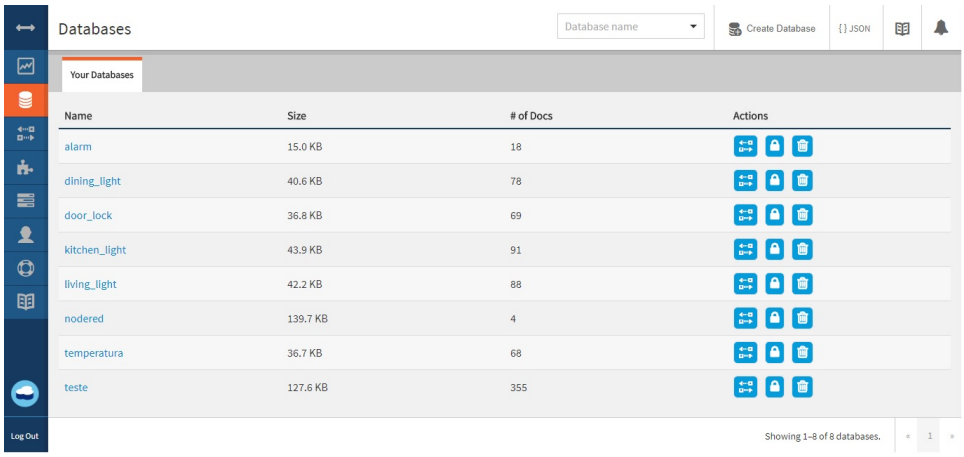


Figura 18: Painel de *databases* do Cloudant

Com as *databases* prontas é necessário agora criar uma *warehouse* e incluir suas *databases* como na Figura 19. Depois disso é necessário abrir com o IBM DashDB e abrir a tabela correspondente a *database* requerida e então poderá exportar ou filtrar as informações.

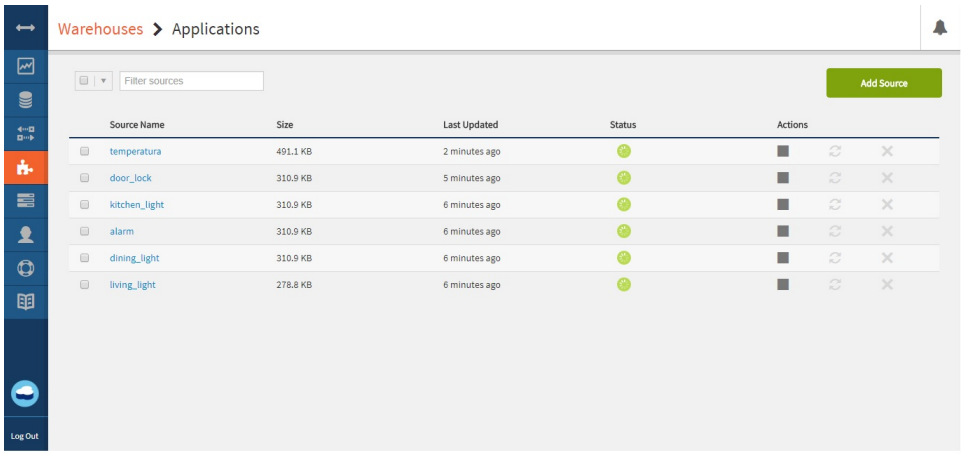


Figura 19: Painel de *warehouses* do Cloudant

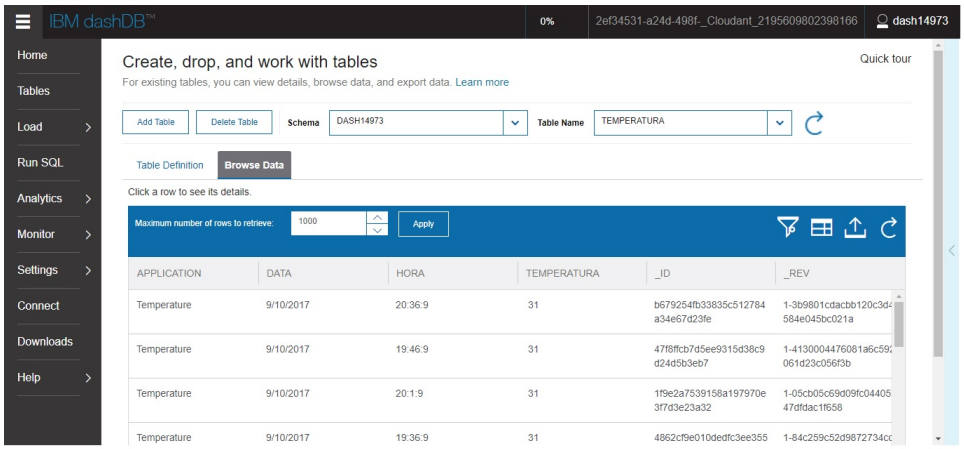


Figura 20: Painel de tabelas do IBM DashDB

3.4 Hardware utilizado

Foi utilizada a placa de desenvolvimento Wemos D1 R2 que é compatível com o Arduino, ela foi escolhida por ser ideal para projetos de robótica e automação que utilizam a internet das coisas, pois possui suporte embutido à rede Wi-Fi. A programação dela é feita através da Arduino IDE da mesma maneira que uma placa Arduino.

A Figura 21 mostra a Wemos D1 R2. Ela foi utilizada basicamente para conectar o hardware com a nuvem, ela se conecta pela internet aos serviços em nuvem do Bluemix através da plataforma Node-Red e assim controla o hardware do projeto.

Figura 21: Wemos D1 R2 (PENA, 2016)

A Wemos D1 R2 foi escolhida entre as placas que poderiam ser utilizadas em projetos deste tipo como Arduino ou o NodeMcu. O Arduino foi descartado devido a sua conectividade via Wi-Fi necessitar a adição de um módulo ou shield ao circuito. Já o NodeMcu se mostra similar a Wemos D1 R2 somente com a diferença de formato entre as placas, sendo possível realizar o mesmo projeto utilizando o NodeMcu somente fazendo alterações necessárias no código.

A Wemos D1 R2 é uma plataforma de software e hardware voltada para a desenvolvimento de aplicações de internet das coisas. Possui um controlador ESP-8266EX, porta micro USB para a alimentação e programação, pino para alimentação externa de 9 a 24V, 9 portas digitais que operam em nível lógico de 3,3V e uma entrada analógica com resolução de 10 bits limitada a 3,2V. O seu módulo ESP8266 consiste em um microprocessador ARM de 32 bits com suporte embutido à rede Wi-Fi (suporte as redes Wi-Fi 802.11 b/g/n e criptografia WPA e WPA2), baixo consumo de energia e memória flash integrada, que permite a ele ser programado de forma independente sem a necessidade de outras placas.

O hardware utilizado para a simulação do circuito está na Figura 23, este sistema foi utilizado para fazer os testes do sistema e verificar se está tudo funcionando como deveria. Nesta figura temos a placa Wemos D1 R2 que foi programada utilizando o programa Arduino IDE, a interface deste se encontra na Figura 22. A placa se comunica com a plataforma em nuvem. E assim controla os demais componentes o código utilizado para a programação está no Apêndice D.

Para a programação da placa Wemos via Arduino IDE é necessário adicionar a biblioteca correspondente a placa “ESP8266WiFi.h”, esta permite a conexão via internet pelo processador ESP8266 da placa. Para a comunicação via protocolo MQTT com a plataforma é necessário utilizar a biblioteca “PubSubClient.h” com ela é possível publicar e se inscrever em tópicos do *broker*.

No código em si, primeiramente a função “setup” faz um *setup* geral do hardware

conectando-o a internet e , depois a função “ntc” extrai a temperatura utilizando o termistor, para calcular a temperatura informada pelo termistor é utilizada a equação de Steinhart–Hart que descreve a temperatura de um dispositivo semicondutor em dada temperatura.

A função “callback” é a principal função do código onde o hardware recebe as informações via MQTT, quando uma mensagem é recebida esta função é chamada e identifica qual é o tópico em que esta mensagem chegou e qual a mensagem, assim executa a ação respectiva. A função “reconnect” serve para reconectar o hardware à internet quando o sistema se desconecta por algum motivo. A função “loop” é o laço principal do programa onde a conexão à internet é testada, o cliente da biblioteca “PubSubClient.h” é rodado novamente para verificar as inscrições aos tópicos, e o alarme quando ligado é monitorado para que seja acionado quando necessário.

Figura 22: Interface de programação

Para fins de teste da aplicação foi utilizado um módulo de relés de 2 canais onde cada relé é responsável pelo acionamento de uma respectiva lâmpada, assim estas lâmpadas podem ser controladas pelo sistema. A Lâmpada 1 simula as luzes da sala de estar e a Lâmpada 2 simula as luzes da cozinha.

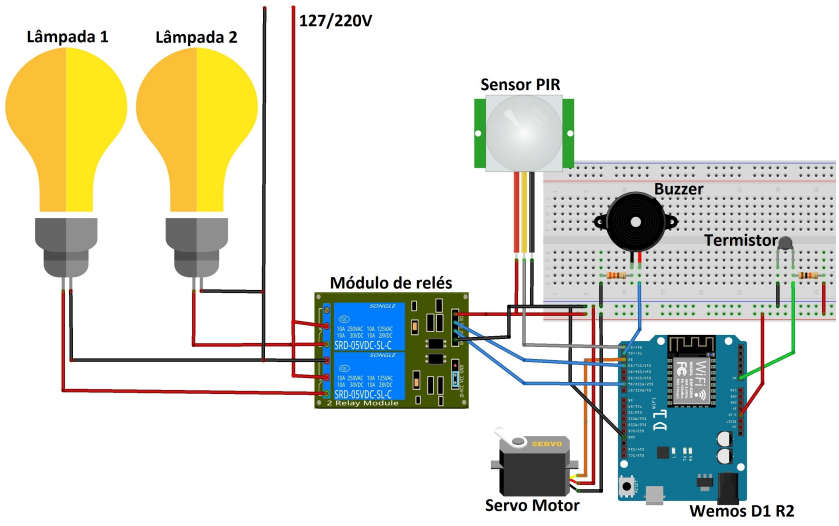


Figura 23: Hardware utilizado para a simulação

O termistor mede a temperatura do ambiente. Temos o Sensor PIR e o Buzzer que são parte do alarme, o Sensor PIR reconhece movimentos e o Buzzer simula a sirene do alarme.

Quando a placa Wemos recebe um comando para ligar ou desligar uma das lâmpadas, seja via painel de controle ou via comando de voz, o relé correspondente a esta será ligado ou desligado, assim acendendo ou apagando a lâmpada correspondente.

Se a placa recebe um sinal para informar a temperatura, a leitura do termistor é enviada para a plataforma e então enviada para o smartphone como resposta, podendo ser uma resposta de voz ou o número de amostra no painel de comando.

Se o sinal recebido é para acionar o alarme, então o alarme é acionado e a placa espera um sinal do Sensor PIR, se esse sinal existir, o Buzzer é acionado e simula uma sirene, mostrando que o alarme funciona corretamente.

Quando é recebido um sinal para que a porta seja trancada ou destrancada a placa envia a informação correspondente ao comando para o servo motor que então se desloca para a posição indicada simulando um trancamento ou abertura da porta.

O código utilizou na placa Wemos D1 R2 238713 bytes (22%) de espaço de armazenamento para programas, o máximo são 1044464 bytes. Variáveis globais usam 32716 bytes (39%) de memória dinâmica, deixando 49204 bytes para variáveis locais, o máximo são 81920 bytes. Assim ainda existe espaço para realizar aplicações maiores, ou adicionar dispositivos.

4 RESULTADOS E DISCUSSÕES

No projeto em geral, o acesso ao hardware via painel de comando foi realizado com sucesso sem nenhum tipo de falha em inúmeros testes com o sistema final. O acesso realizado via comando de voz apresentou problema no próprio reconhecimento de voz do *smartphone* quando a palavra de comando "home"não era reconhecida no início da frase, entretanto se reconhecido corretamente o sistema não apresentou falha em nenhum momento.

O controle do sistema via painel de comando tem um tempo de resposta praticamente imperceptível, para medição deste foram medidos 20 valores com um cronômetro, a média obtida foi de 0,66s e o desvio padrão de 0,07s. Já o controle de voz tem um tempo de resposta do ciclo completo (desde o envio da mensagem até que a mensagem comece a ser reproduzida pelo *smartphone*) em torno de 4 segundos, foram realizadas novamente 20 medidas, a média obtida foi de 3,82s e o desvio padrão foi 0,33s.

O sistema utilizado no *smartphone* para fazer o papel de painel de comando foi executado sem problemas maiores, no entanto a escolha do aplicativo que exerceu este papel é fundamental e exigiu pesquisa e testes, pois cada aplicativo tem um método de funcionamento e suas limitações. O aplicativo utilizado “MQTT Dash” foi o mais completo encontrado durante as pesquisas, ele é capaz de enviar as informações e receber informações, sendo assim possível ter botões interativos que mudam de estado com o estado das aplicações. Isto evita um grande problema apresentado durante a execução do projeto, que é o estado da aplicação mudar e o status do botão não, assim seria necessário apertar o botão para mudar o estado da aplicação para o correto e depois disso alterar o estado para o almejado.

O serviço de conversação utilizado foi o Watson Conversation, a programação deste requer uma boa noção de como o serviço funciona, o serviço faz o papel de diálogo e resposta aos comandos de voz, um grande obstáculo na utilização deste serviço é o mesmo problema que o aplicativo do *smartphone* conseguiu evitar.

O problema é que se o *smartphone* envia um comando direto para o hardware e o serviço de conversação não estiver a par deste comando, o serviço continuará entendendo o estado da aplicação como o anterior, assim para o *smartphone* o estado da aplicação será correto e o serviço de conversação terá conhecimento do estado como errado e quando receber um comando a resposta será errada e não gerará a ação requerida.

Como o serviço de conversação entende o estado das aplicações por variáveis próprias dentro do serviço, só é possível mudar estas variáveis com comandos enviados ao serviço. Em um primeiro momento no projeto o *smartphone* envia comandos diretamente para a placa Wemos, então o comando não passa pelo serviço de conversação que não fica a par do comando e entende que não existem modificações no sistema.

Alterar as variáveis do serviço individualmente não é possível, então não é possível alterar a variável correspondente quando um comando é enviado. Então para resolver o problema,

necessariamente todos os comandos devem passar pelo serviço de conversação e o este os repassar para a plataforma Node-Red que os processa, não havendo mais comunicação direta entre o *smartphone* e o hardware.

Na aplicação do hardware a programação da placa Wemos para a comunicação via protocolo MQTT com a plataforma é necessário utilizar a biblioteca "PubSubClient.h" com ela é possível publicar e se inscrever em tópicos do *broker*, mas existe um problema, em que a inscrição utilizando a função "callback" e comparando o tópico de que a mensagem foi recebida, descobrir de qual tópico a mensagem se originou, entretanto se existirem muitos tópicos diferentes, observou-se que o sistema fica mais lento, demorando para responder aos comandos. Então para este problema não existir foi necessário reduzir os tópicos na medida do possível e em vez de fazer um tópico para cada aplicação, os tópicos foram divididos por tipo de aplicação, e cada aplicação por uma mensagem recebida. Por exemplo o tópico ao invés de ser um tópico para cada luz, existe um tópico para todas as luzes e a mensagem recebida por este corresponde as luzes de cada cômodo.

Na Figura 24 temos o painel da plataforma Node-Red que é acessado pelo próprio navegador no endereço "http://otavio1.mybluemix.net/ui/#/0", ele nos mostra o funcionamento do sistema, mostrando o status de cada componente do sistema em cada instante de tempo por meio de gráficos e também a temperatura. Nesta imagem o sistema ficou em funcionamento um dia inteiro, assim foi simulado como o sistema se comportaria nesse período. Os dados nos mostram como as aplicações foram ligadas e desligadas durante o período.

Na Figura 25 temos um gráfico feito utilizando o Excel, onde podemos ver os estados das luzes da sala de estar armazenados em nuvem utilizando o serviço Cloudant, comparando este gráfico com o estado das luzes da sala de estar da Figura 24 é possível observar que os dados armazenados correspondem exatamente com os dados apresentados no painel da plataforma Node-Red, podendo assim validar os dados recebidos tanto pelo painel quanto os armazenados em nuvem.

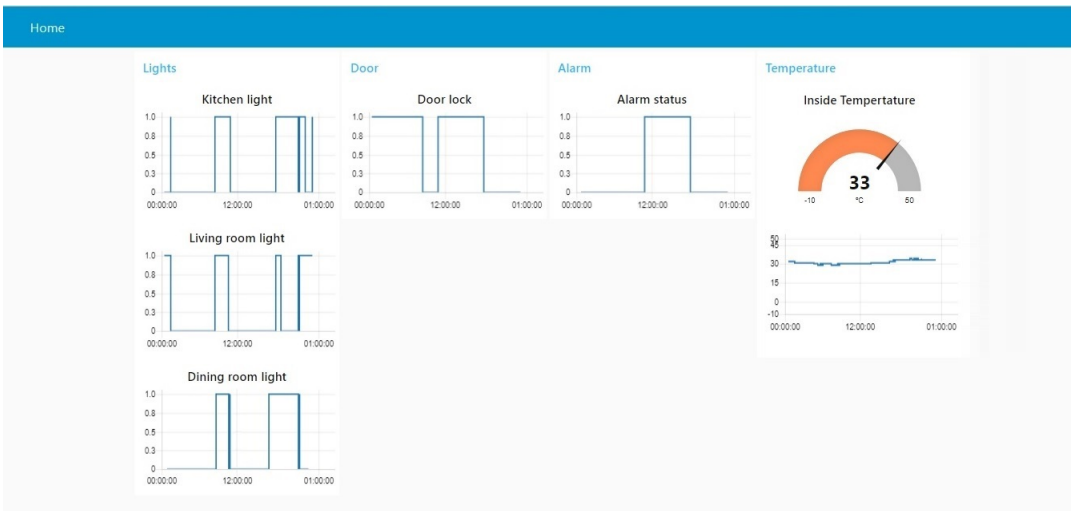


Figura 24: Painel da plataforma Node-Red em funcionamento

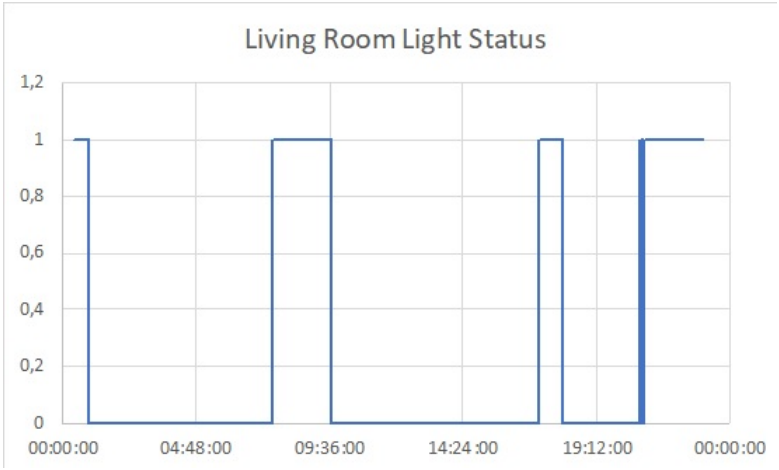


Figura 25: Gráfico dos estados das luzes da sala de estar armazenados em nuvem

Na Figura 26 e na Figura 27 temos respectivamente o painel de comando e o hardware em funcionamento no mesmo instante. O sistema e o painel estão em sincronia, o painel mostrando as luzes da sala de estar e da cozinha ligadas, enquanto o módulo de relés está com os 2 relés acionados, que seriam a ligação das respectivas luzes.

Figura 26: Painel de comando em funcionamento

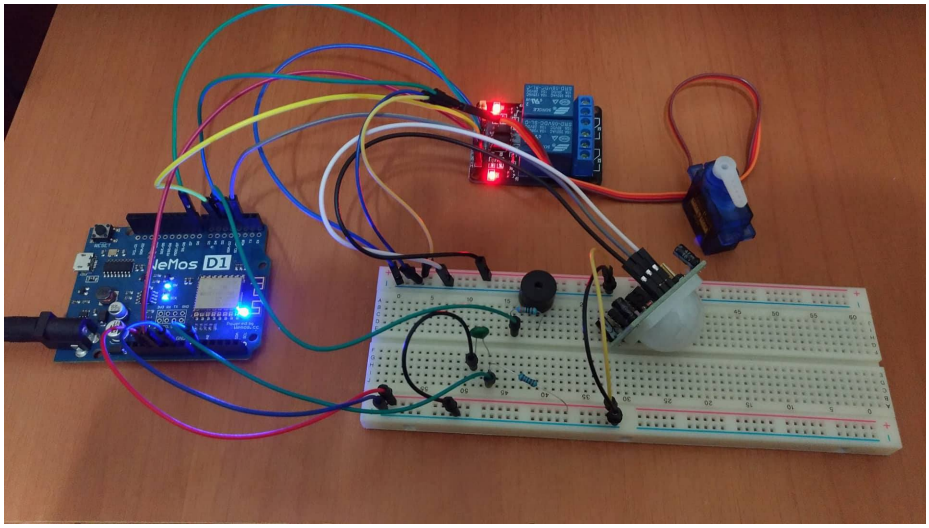


Figura 27: Hardware em funcionamento respondendo ao comando

Na Figura 28 temos o comando de voz dito para o *smartphone* e na Figura 29 temos a resposta do hardware para o comando de voz enviado, o relé correspondente a luz da sala de estar é desligado, pelo comando de voz “home turn off the living room lights”. Na Figura 30 é possível ver na plataforma Node-Red a resposta que foi enviada e reproduzida pelo *smartphone* destacada em vermelho.

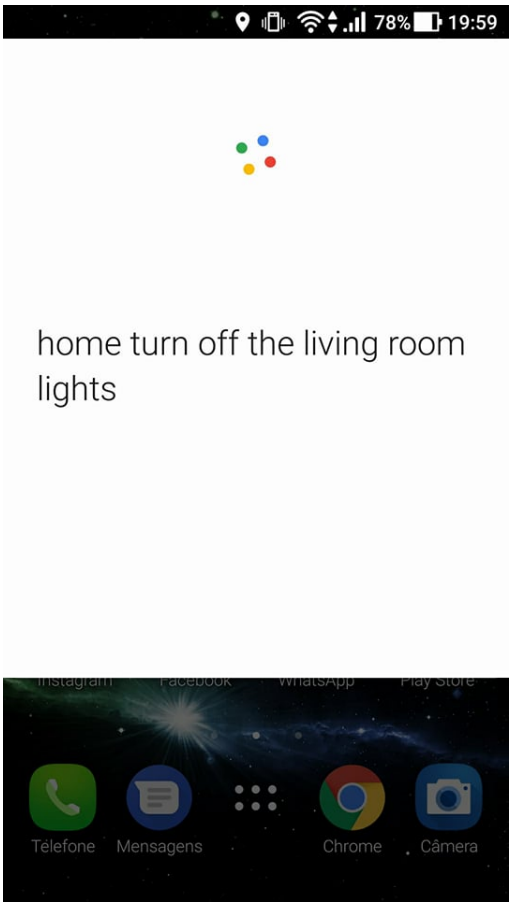


Figura 28: Comando de voz dito ao *smartphone*

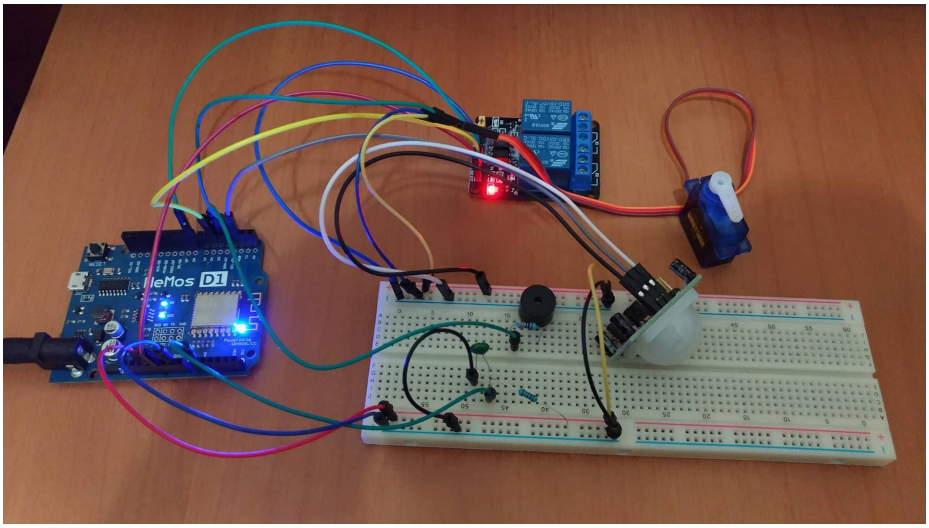


Figura 29: Hardware em funcionamento respondendo ao comando de voz

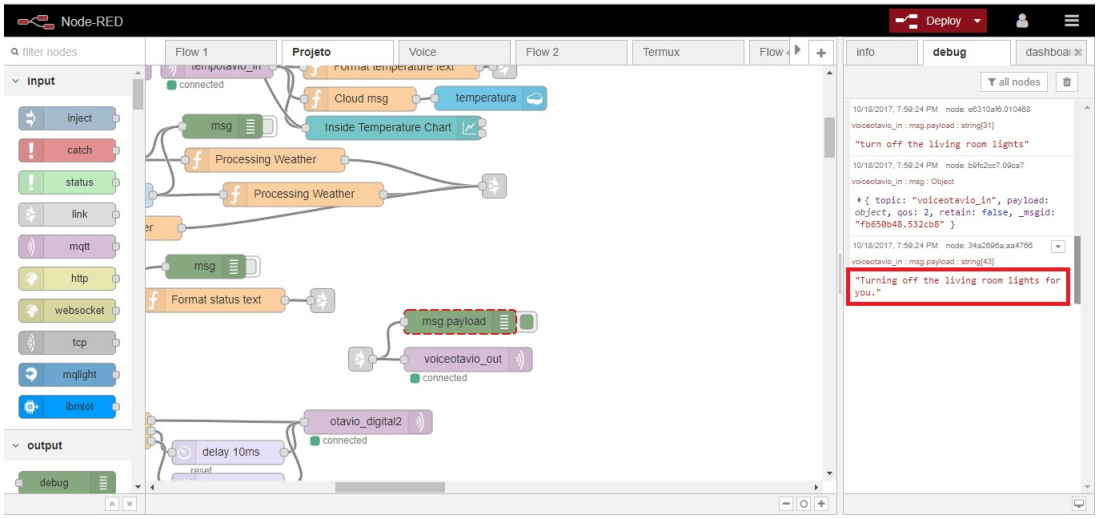


Figura 30: Mensagem enviada ao *smartphone* como resposta ao comando anterior

O sistema é capaz de responder às mensagens de voz dos tipos:

- Cumprimentos e despedidas.
- Ligar e desligar as luzes da sala de estar, cozinha e sala de jantar.
- Perguntas sobre a temperatura interna.
- Perguntas sobre o clima no dia atual e nos 10 dias seguintes.
- Trancar e destrancar a porta.
- Ligar e desligar o alarme.
- Perguntas sobre os estados de aplicações.

- Perguntas sobre suas capacidades.
- Responder quando a mensagem não for reconhecida.

5 CONCLUSÃO

Com a execução do projeto e os resultados relatados, conclui-se que foi possível alcançar o objetivo inicial deste por meio dos métodos propostos. A utilização de plataformas em nuvem é um método muito utilizado atualmente. As plataformas em nuvem utilizadas se mostraram eficientes.

Foi possível concluir que a plataforma Node-Red se mostrou muito útil, ela foi o elo principal do sistema, a sua capacidade de ligar diferentes serviços, processar a informação durante as ligações e se comunicar com diferentes sistemas garante uma flexibilidade para integrar vários sistemas, mesmo que se comuniquem por maneiras diferentes. Se existisse acesso a mais equipamentos que são capazes de se comunicar, o que no futuro será uma realidade, a plataforma ainda seria capaz de integrar todos os equipamentos e processar a comunicação entre eles e adicionar os serviços em nuvem disponíveis a eles.

O serviço Watson Conversation demonstrou eficiência na sua finalidade de realmente criar diálogos, além de simplesmente responder as mensagens com texto, tem ferramentas capazes de entender o contexto da mensagem e assim ser possível gerar uma ação sobre a mensagem recebida.

Em geral a utilização de plataformas em nuvem voltadas para automação residencial, se mostrou promissora. E com o crescente número de equipamentos que são capazes de se conectar a aplicação em nuvem pode ser um caminho para evolução do segmento. Com a utilização deste método pode não ser necessária a utilização de hardwares para processamento de informação, isso se os equipamentos forem capazes de se comunicar, deixando para que todo o processamento seja feito em nuvem e o hardware seja somente necessário para realizar as ações e enviar informações de monitoramento, além de permitir que toda a comunicação seja feita sem fio a partir do Wi-Fi.

REFERÊNCIAS

ARNELL, S. **Use Node-red on Android to Simulate IOT device connecting to Watson IOT**. 2017. Disponível em: <https://developer.ibm.com/recipes/tutorials/use-nodered-on-android-to-simulate-iot-device-connecting-to-watson-iot/>. Acesso em: 22.08.2017.

GERBER, A. **Connecting all the things in the Internet of Things**. 2017. Disponível em: <https://www.ibm.com/developerworks/library/iot-lp101-connectivity-network-protocols/index.html>. Acesso em: 19.06.2017.

MARTINS, P. Mundo digital. **Potência**, v. 13, n. 136, p. 10–23, 2017.

MILLER, M. **IBM Bluemix Docs Conversation**. 2017. Disponível em: <https://console.bluemix.net/docs/services/conversation/index.html#about>. Acesso em: 19.09.2017.

MORGAN, J. **A Simple Explanation Of 'The Internet Of Things'**. 2014. Disponível em: <https://www.forbes.com/forbes/welcome/?toURL=https://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/&refURL=https://www.google.com.br/&referrer=https://www.google.com.br/>. Acesso em: 19.06.2017.

MQTT.ORG. **Frequently Asked Questions**. 2017. Disponível em: <http://mqtt.org/faq>. Acesso em: 19.06.2017.

MURATORI, J. R.; Bó, P. H. D. **Automação residencial:histórico, definições e conceitos**. 2011. Disponível em: <http://www.osetoreletrico.com.br/wp-content/uploads/2011/04/Ed62_fasc_automacao_capI.pdf>. Acesso em: 19.06.2017.

NODERED.ORG. **Flow-based programming for the Internet of Things**. 2017. Disponível em: <https://nodered.org/#features>. Acesso em: 20.06.2017.

PENA, M. **Iniciando em IoT com a placa D1 da Wemos.cc, um ESP8266 compatível com Arduino - Parte 1**. 2016. Disponível em: <http://blog.robotto.com.br/2016/01/iniciando-em-iot-com-placa-d1-da.html>. Acesso em: 12.08.2017.

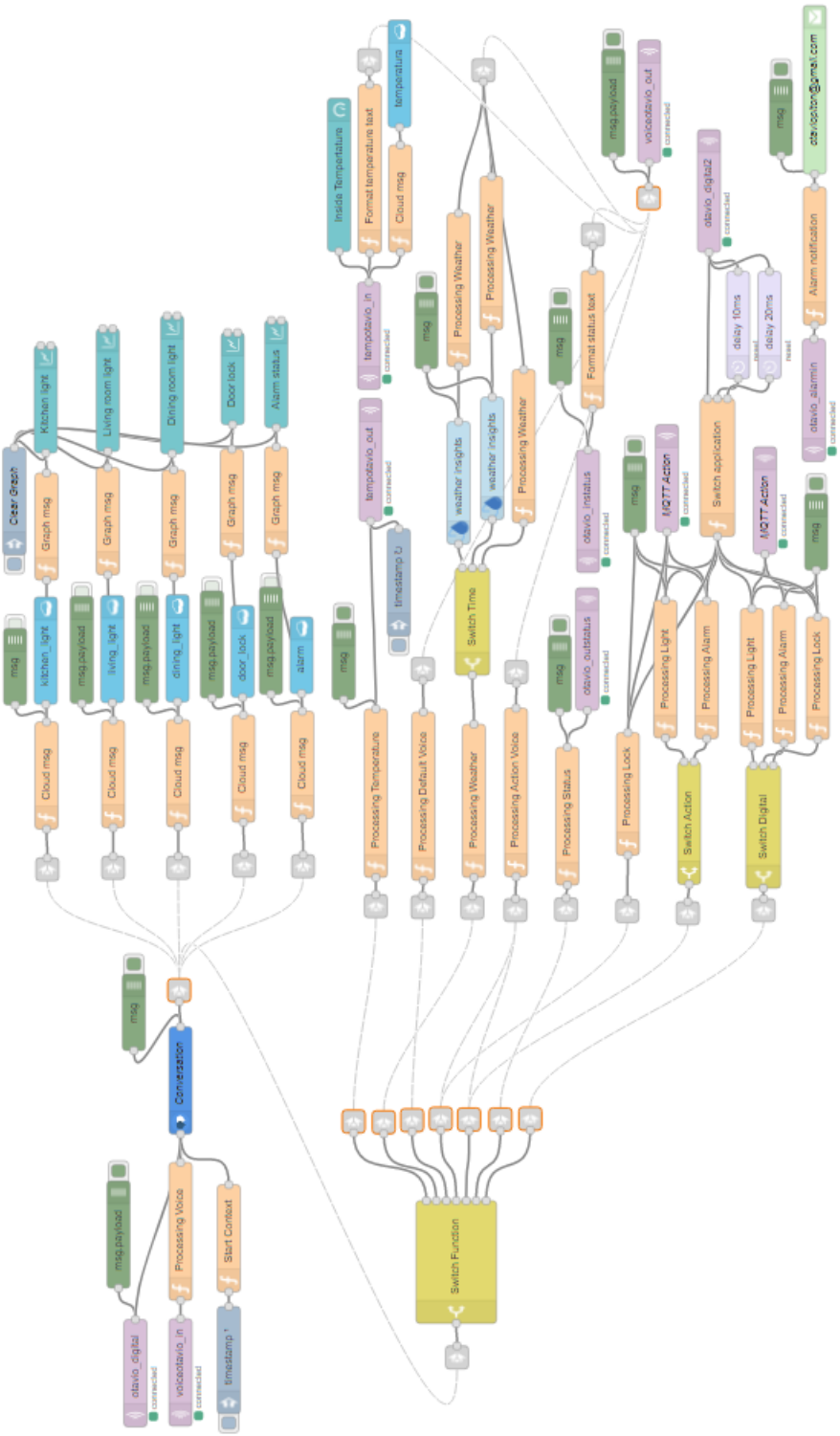
REYES, A. T. **O que é IBM Bluemix?** 2014. Disponível em: <https://www.ibm.com/developerworks/br/cloud/library/cl-bluemixfoundry/index.html>. Acesso em: 29.06.2017.

ROUSE, M. **IBM Watson supercomputer**. 2016. Disponível em: <http://whatis.techtarget.com/definition/IBM-Watson-supercomputer>. Acesso em: 18.09.2017.

YUAN, M. **Getting to know MQTT**. 2017. Disponível em: <https://www.ibm.com/developerworks/library/iot-mqtt-why-good-for-iot/index.html>. Acesso em: 19.06.2017.

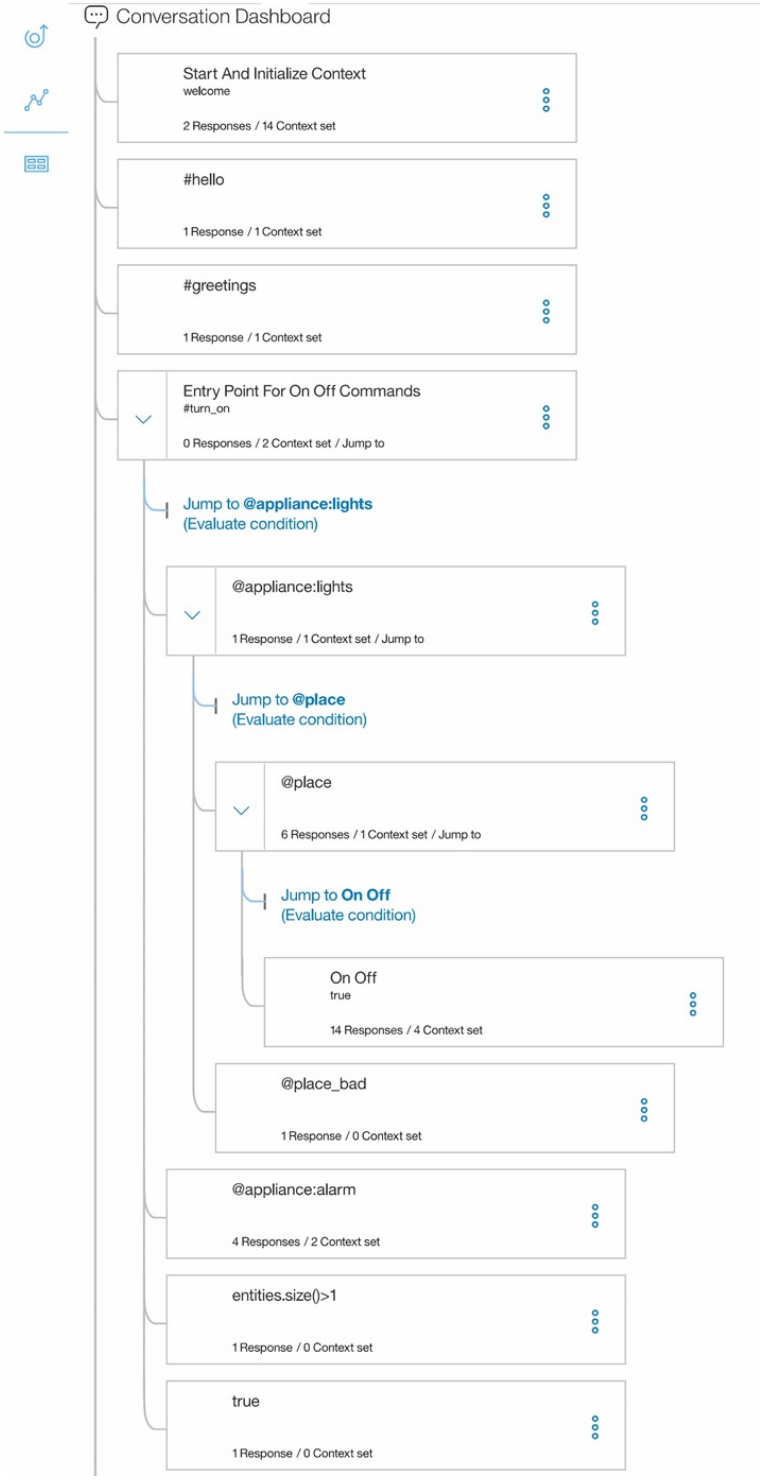
APÊNDICE A – *FLOW* COMPLETO DA PLATAFORMA NODE-RED

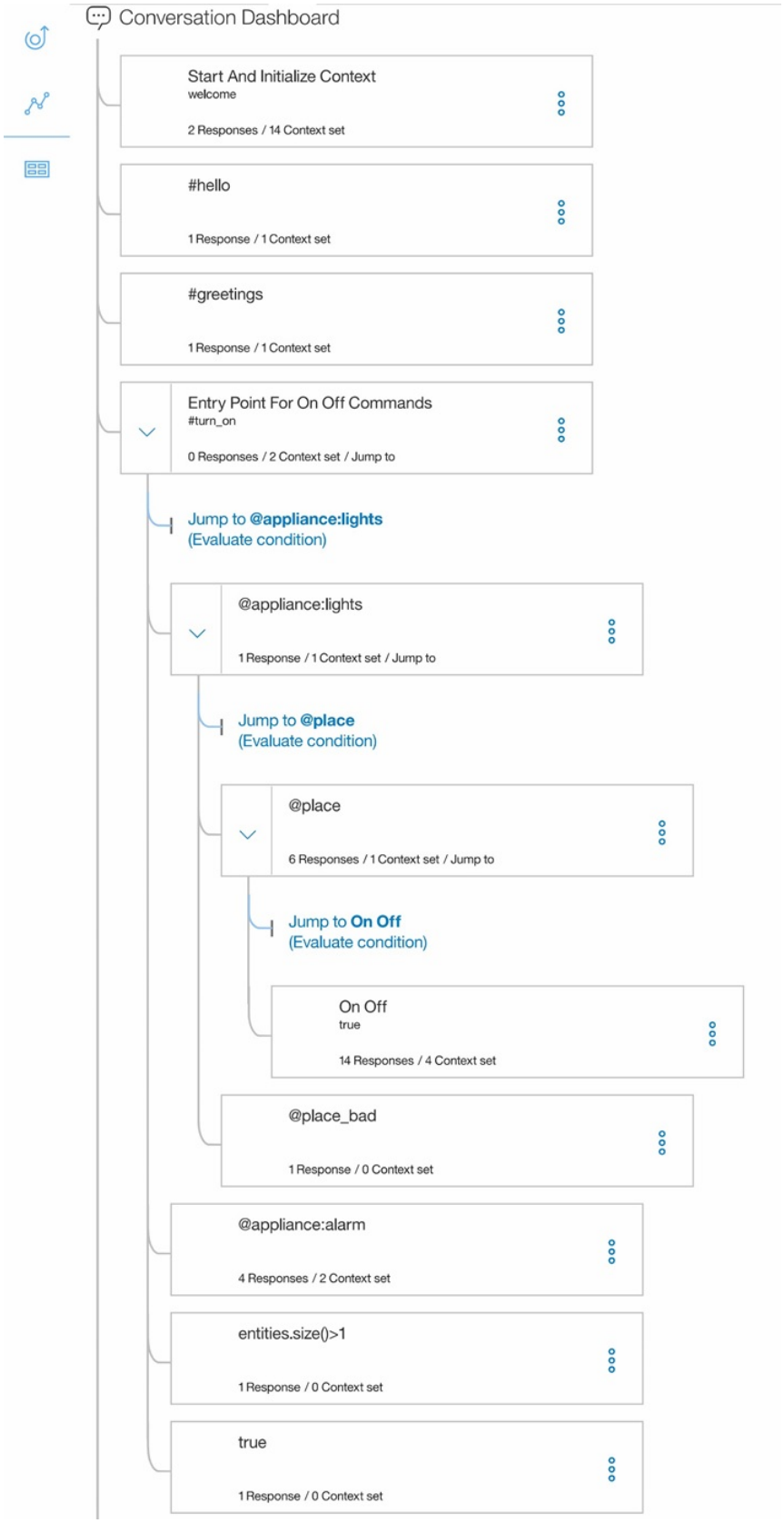
O flow pode ser encontrado e importado do repositório: <https://github.com/otaviopiton/TCC/blob/master/Node-Red%20Flow>.

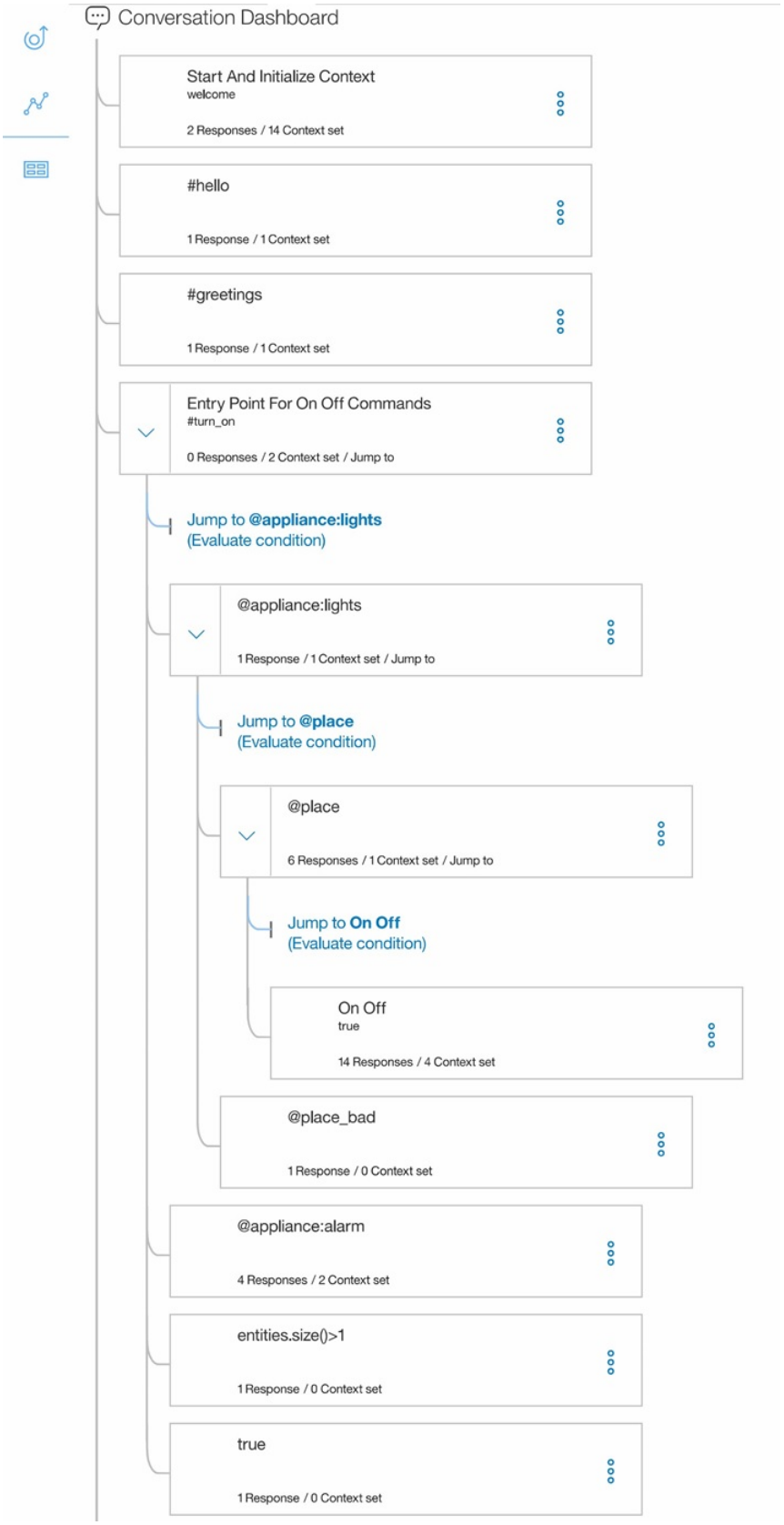


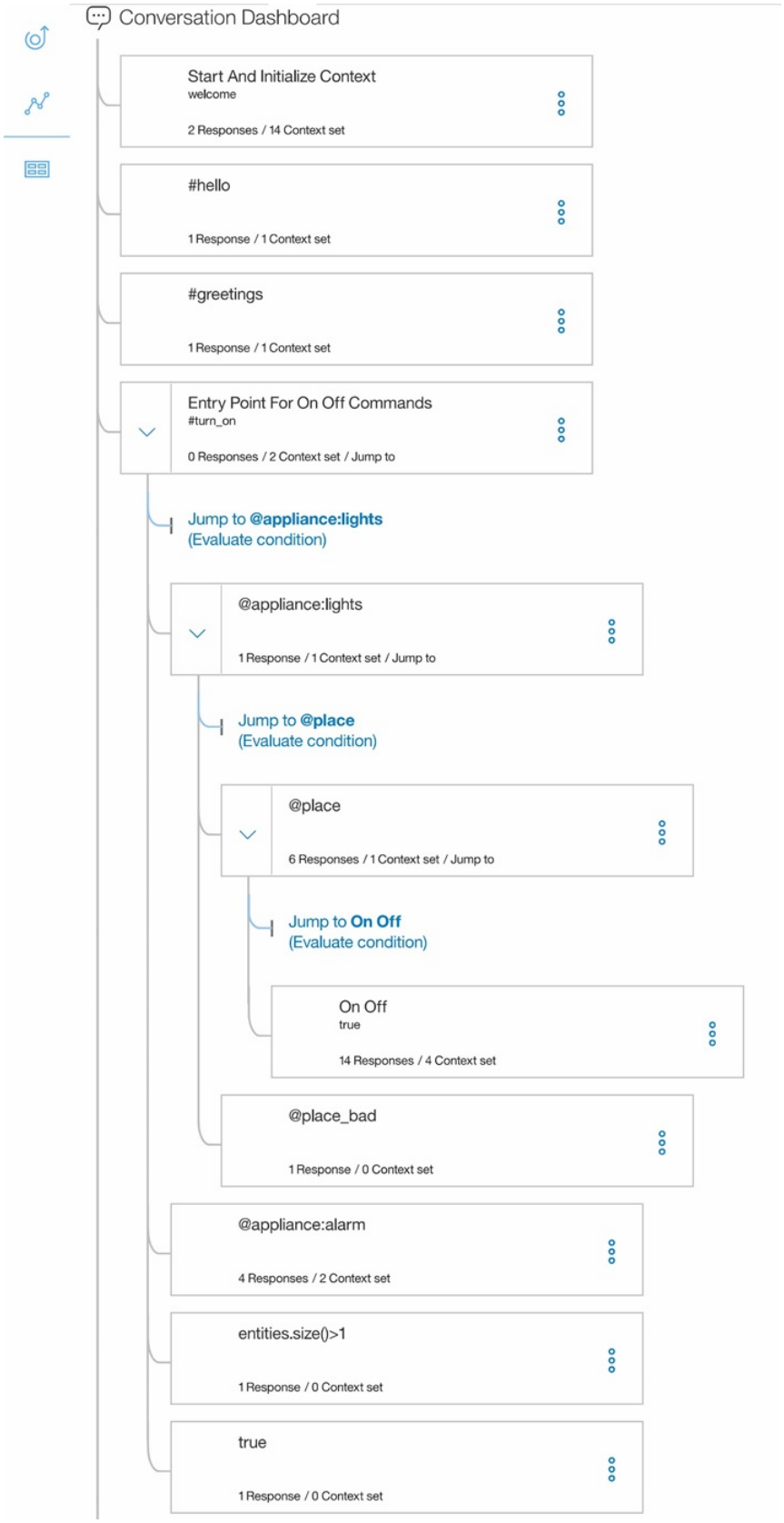
APÊNDICE B – *WORKSPACE* COMPLETO DO SERVIÇO WATSON CONVERSATION

Para ter acesso ao arquivo em formato json e assim importá-lo para o serviço Watson Conversation baixe todos os arquivos de <https://github.com/otaviopiton/TCC> pelo botão “Clone or download”.



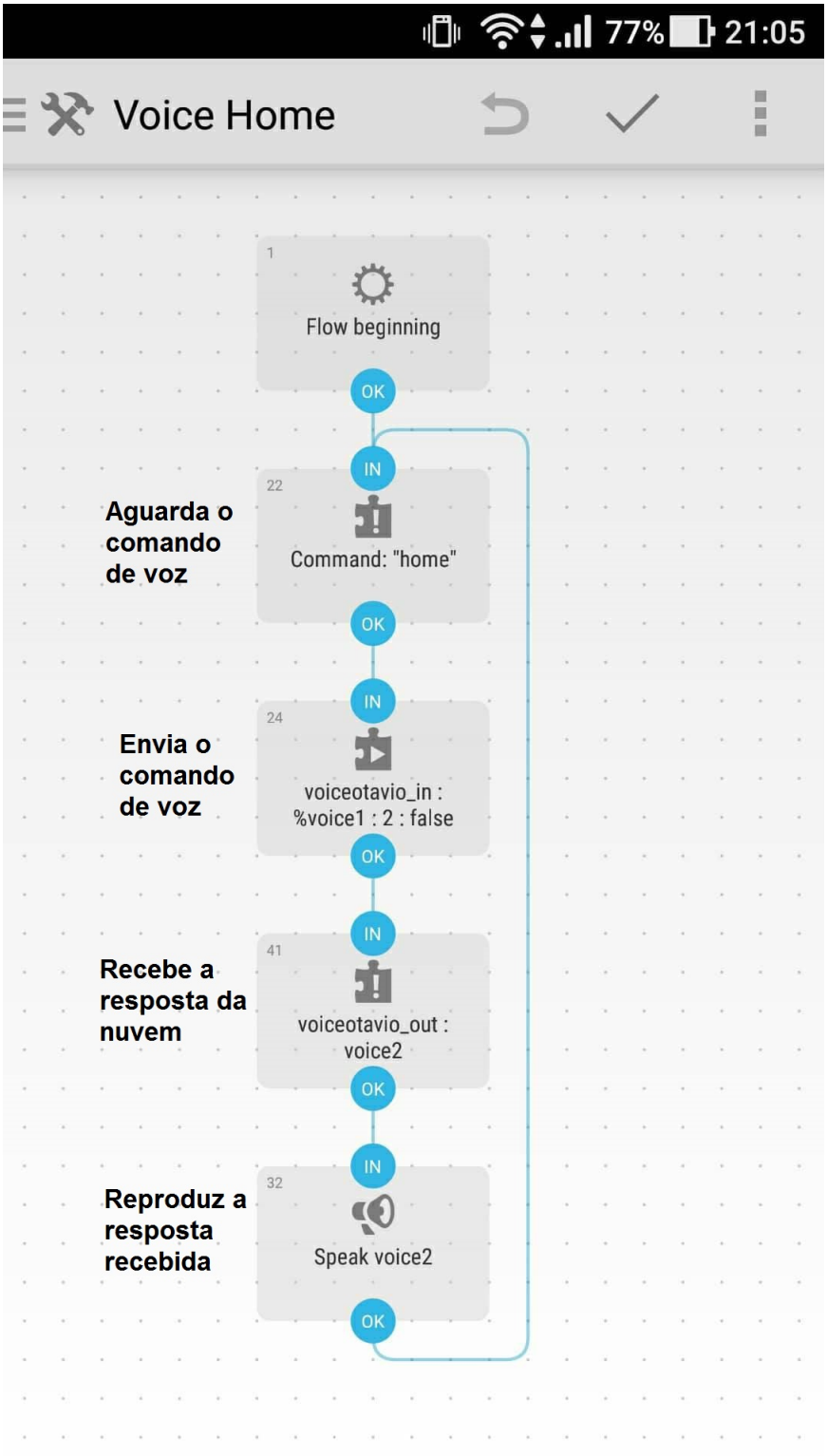

Conversation Dashboard
Start And Initialize Context
welcome
2 Responses / 14 Context set
#hello
1 Response / 1 Context set
#greetings
1 Response / 1 Context set
Entry Point For On Off Commands
#turn_on
0 Responses / 2 Context set / Jump to
Jump to @appliance:lights
(Evaluate condition)
@appliance:lights
1 Response / 1 Context set / Jump to
Jump to @place
(Evaluate condition)
@place
6 Responses / 1 Context set / Jump to
Jump to On Off
(Evaluate condition)
On Off
true
14 Responses / 4 Context set
@place_bad
1 Response / 0 Context set
@appliance:alarm
4 Responses / 2 Context set
entities.size()>1
1 Response / 0 Context set
true
1 Response / 0 Context set


Conversation Dashboard
Start And Initialize Context
welcome
2 Responses / 14 Context set
#hello
1 Response / 1 Context set
#greetings
1 Response / 1 Context set
Entry Point For On Off Commands
#turn_on
0 Responses / 2 Context set / Jump to
Jump to @appliance:lights
(Evaluate condition)
@appliance:lights
1 Response / 1 Context set / Jump to
Jump to @place
(Evaluate condition)
@place
6 Responses / 1 Context set / Jump to
Jump to On Off
(Evaluate condition)
On Off
true
14 Responses / 4 Context set
@place_bad
1 Response / 0 Context set
@appliance:alarm
4 Responses / 2 Context set
entities.size()>1
1 Response / 0 Context set
true
1 Response / 0 Context set


Conversation Dashboard
Start And Initialize Context
welcome
2 Responses / 14 Context set
#hello
1 Response / 1 Context set
#greetings
1 Response / 1 Context set
Entry Point For On Off Commands
#turn_on
0 Responses / 2 Context set / Jump to
Jump to @appliance:lights
(Evaluate condition)
@appliance:lights
1 Response / 1 Context set / Jump to
Jump to @place
(Evaluate condition)
@place
6 Responses / 1 Context set / Jump to
Jump to On Off
(Evaluate condition)
On Off
true
14 Responses / 4 Context set
@place_bad
1 Response / 0 Context set
@appliance:alarm
4 Responses / 2 Context set
entities.size()>1
1 Response / 0 Context set
true
1 Response / 0 Context set

APÊNDICE C – *FLOW* UTILIZADO NO *SMARTPHONE* PARA REALIZAR CONTROLE DE VOZ

O flow pode ser encontrado e importado do repositório: <https://github.com/otaviopiton/TCC/blob/master/Voice%20Home.flo>.



APÊNDICE D – CÓDIGO UTILIZADO NA PLACA WEMOS D1 R2

O código também pode ser encontrado e importado do repositório: <https://github.com/otaviopiton/TCC/blob/master/Wemos_Code>.

```
// BIBLIOTECAS ------------------------------------------------
#include <ESP8266WiFi.h>
#include<PubSubClient.h>
#include<Servo.h>

// VARIAVEIS --------------------------------------------------

char receivedChar;
char msg[5];

// Temperature
byte NTCPin = A0;
int i, n=0;
char temp;
float steinhart;

// Led
int dining_light = D7;
char dining_status = 'h';
int living_light = D6;
char living_status = 'g';
int kitchen_light = D3;
char kitchen_status = 'f';
char all_status = 'z';

// Servo
int pinservo = D2;
Servo s;
int pos;
char lock_status = 'o';

// Alarme
int buzzer = D1;
int pinopir = D0;
char alarm_status = 'r';
int valorpir = 0;


const char* mqtt_server = "iot.eclipse.org";
const char* ssid = "TP-LINK_C274";
```

```
41 const char* password = "12111994";
42 WiFiClient espclient;
43
44 // SETUP GERAL ---------------------------------------------
45
46 void setup() {
47
48   pinMode(dining_light, OUTPUT);
49   pinMode(living_light, OUTPUT);
50   pinMode(kitchen_light, OUTPUT);
51   pinMode(BUILTIN_LED, OUTPUT);
52   pinMode(buzzer, OUTPUT);
53
54   pinMode(pinopir, INPUT);
55
56   Serial.begin(115200);
57   Serial.print("WiFi connecting.");
58   WiFi.begin(ssid, password);        //SSID,PASSWORD
59   while (WiFi.status() != WL_CONNECTED) {
60     delay(500);
61     Serial.print(".");
62   }
63   Serial.print("Connected");
64   Serial.println();
65
66   reconnect();
67
68   s.attach(pinservo);
69   s.write(0);
70
71 }
72
73 PubSubClient client(mqtt_server, 1883, callback, espclient);
74
75 // Temperature sensor NTC ----------------------------------
76
77 void ntc() {
78
79   float A0value;
80   float Resistance;
81
82   for (i = 0; i < 5; i++) {
83     A0value = A0value + analogRead(NTCPin);
84   }
85   A0value = A0value / 5;
86
87   Resistance = (1023 / A0value) - 1;
88   Resistance = 10000 / Resistance;
```

```
89
90    float steinhart;
91
92    steinhart = Resistance / 10000; // (R/Ro)
93    steinhart = log(steinhart); // ln(R/Ro)
94    steinhart = steinhart / 3950; // 1/B * ln(R/Ro)
95    steinhart = steinhart + (1.0 / (25 + 273.15)); // + (1/To)
96    steinhart = 1.0 / steinhart; // Invert
97    steinhart = steinhart - 273.15; // convert to C
98    steinhart = round(steinhart);
99
100   int temp = (int) steinhart;
101
102   Serial.print("Temperature:");
103   Serial.print(steinhart);
104   Serial.println("C");
105
106   snprintf (msg, 5, " %d ", temp);
107
108   client.publish("tempotavio_in", msg);
109 }
110
111 // FUNCAO CALLBACK ------------------------------------------------
112
113 void callback(char* topic, byte* payload, unsigned int length) {
114   Serial.print("Message arrived [");
115   Serial.print(topic);
116   Serial.print("] ");
117
118
119   if (strcmp(topic, "testlight_otavio") == 0) {
120     for (int i = 0; i < length; i++) {
121       char receivedChar = (char)payload[i];
122       Serial.print(receivedChar);
123       if (receivedChar == '1')
124         digitalWrite(BUILTIN_LED, HIGH);
125       if (receivedChar == '0')
126         digitalWrite(BUILTIN_LED, LOW);
127     }
128   }
129
130   if (strcmp(topic, "otavio_alarm") == 0) {
131     n = 0;
132     for (int i = 0; i < length; i++) {
133       char receivedChar = (char)payload[i];
134       Serial.print(receivedChar);
135       if (receivedChar == '1'){
136         alarm_status = 'm';
```

```
137         snprintf (msg, 3, "%c", alarm_status);
138         client.publish("otavio_allstatus", msg);}
139       if (receivedChar == '0'){
140         alarm_status = 'r';
141         snprintf (msg, 3, "%c", alarm_status);
142         client.publish("otavio_allstatus", msg);}
143     }
144   }
145
146   if (strcmp(topic, "otavio_lock") == 0) {
147     for (int i = 0; i < length; i++) {
148       char receivedChar = (char)payload[i];
149       Serial.print(receivedChar);
150       if (receivedChar == '0'){
151         lock_status = 'o';
152         snprintf (msg, 3, "%c", lock_status);
153         client.publish("otavio_allstatus", msg);
154         s.write(90);}
155       if (receivedChar == '1'){
156         lock_status = 'c';
157         snprintf (msg, 3, "%c", lock_status);
158         client.publish("otavio_allstatus", msg);
159         s.write(0);}
160     }
161   }
162
163   if (strcmp(topic, "otavio_light") == 0) {
164     for (int i = 0; i < length; i++) {
165       char receivedChar = (char)payload[i];
166       Serial.print(receivedChar);
167       if (receivedChar == 'k'){
168         digitalWrite(kitchen_light, HIGH);
169         kitchen_status = 'k';
170         snprintf (msg, 3, "%c", kitchen_status);
171         client.publish("otavio_allstatus", msg);}
172       if (receivedChar == 'f'){
173         digitalWrite(kitchen_light, LOW);
174         kitchen_status = 'f';
175          snprintf (msg, 3, "%c", kitchen_status);
176         client.publish("otavio_allstatus", msg);}
177
178       if (receivedChar == 'd'){
179         digitalWrite(dining_light, HIGH);
180         dining_status = 'd';
181          snprintf (msg, 3, "%c", dining_status);
182         client.publish("otavio_allstatus", msg);}
183       if (receivedChar == 'h'){
184         digitalWrite(dining_light, LOW);
```

```
185         dining_status = 'h';
186         snprintf (msg, 3, "%c", dining_status);
187         client.publish("otavio_allstatus", msg);}
188 
189       if (receivedChar == 'l'){
190         digitalWrite(living_light, HIGH);
191         living_status = 'l';
192         snprintf (msg, 3, "%c", living_status);
193         client.publish("otavio_allstatus", msg);}
194       if (receivedChar == 'g'){
195         digitalWrite(living_light, LOW);
196         living_status = 'g';
197         snprintf (msg, 3, "%c", living_status);
198         client.publish("otavio_allstatus", msg);}
199 
200       if (receivedChar == 'a'){
201         digitalWrite(kitchen_light, HIGH);
202         kitchen_status = 'k';
203         digitalWrite(dining_light, HIGH);
204         dining_status = 'd';
205         digitalWrite(living_light, HIGH);
206         living_status = 'l';
207         all_status = 'a';
208         snprintf (msg, 3, "%c", all_status);
209         client.publish("otavio_allstatus", msg);}
210 
211       if (receivedChar == 'z'){
212         digitalWrite(kitchen_light, LOW);
213         kitchen_status = 'f';
214         digitalWrite(dining_light, LOW);
215         dining_status = 'h';
216         digitalWrite(living_light, LOW);
217         living_status = 'g';
218         all_status = 'z';
219         snprintf (msg, 3, "%c", all_status);
220         client.publish("otavio_allstatus", msg);}
221     }
222   }
223 
224   if (strcmp(topic, "otavio_outstatus") == 0) {
225     for (int i = 0; i < length; i++) {
226       char receivedChar = (char)payload[i];
227       Serial.print(receivedChar);
228     if (receivedChar == 'k'){
229     snprintf (msg, 3, "%c", kitchen_status);
230     client.publish("otavio_instatus", msg);}
231 
232     if (receivedChar == 'd'){
```

```
233     snprintf (msg, 3, "%c", dining_status);
234     client.publish("otavio_instatus", msg);}
235 
236     if (receivedChar == 'l'){
237     snprintf (msg, 3, "%c", living_status);
238     client.publish("otavio_instatus", msg);}
239 
240     if (receivedChar == 'c'){
241     snprintf (msg, 3, "%c", lock_status);
242     client.publish("otavio_instatus", msg);}
243 
244     if (receivedChar == 'm'){
245     snprintf (msg, 3, "%c", alarm_status);
246     client.publish("otavio_instatus", msg);}
247     }
248   }
249 
250 
251   if (strcmp(topic, "tempotavio_out") == 0) {
252     ntc();
253   }
254 
255   Serial.println();
256 }
257 
258 // FUNCAO RECONNECT ------------------------------------------
259 void reconnect() {
260   // Loop until we're reconnected
261   while (!client.connected()) {
262     Serial.print("Attempting MQTT connection...");
263     // Attempt to connect
264     if (client.connect("espclient")) {
265       Serial.println("Connected");
266       // ... and subscribe to topic
267 
268       client.subscribe("otavio_light");
269       client.subscribe("otavio_lock");
270       client.subscribe("otavio_alarm");
271       client.subscribe("testlight_otavio");
272       client.subscribe("tempotavio_out");
273       client.subscribe("otavio_outstatus");
274 
275     } else {
276       Serial.print("failed, rc=");
277       Serial.print(client.state());
278       Serial.println(" try again in 5 seconds");
279       // Wait 5 seconds before retrying
280       delay(5000);
```

```
281     }
282   }
283 }
284
285 // MAIN LOOP ------------------------------------------------
286 void loop()
287 {
288   if (!client.connected()) {
289     reconnect();
290   }
291
292   client.loop();
293
294
295   // Alarm function
296   if (alarm_status == 'm'){
297
298     valorpir = digitalRead(pinopir);
299
300      if (valorpir == 1) {
301       Serial.print("Alarm");
302       Serial.println();
303       tone(buzzer, 3000);
304       delay(2000);
305       noTone(buzzer);
306     } else {
307       noTone(buzzer);
308     }
309   }
310
311 }
```