UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Utilizando o Home Assistant e o NodeMCU para um modelo genérico de automação moderna.

Trabalho de Conclusão de Curso

Leonardo José Nascimento Silva

Departamento de Computação/UFS

São Cristóvão – Sergipe

2019

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Leonardo José Nascimento Silva

Utilizando o Home Assistant e o NodeMCU para um modelo genérico de automação moderna.

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): Giovanny Fernando Lucero Palma

São Cristóvão – Sergipe

2019

I dedicate this thesis to all my family, friends and professors who gave me the necessary support to get here.

I know the streets are cruel
but i will enjoy the ride today
(Dream theater)

Lista de ilustrações

List of abbreviations and acronyms

IoT	Internet of Things
IO	Input/Output
CPU	Central Process Unit
SDK	Software Development Kit
ADC	Analogic Digital Conversor
PWM	Power Width Modulation
IDE	integrated development environment
M2M	Machine-to-Machine
MQTT	Message Queuing Telemetry Transport
SSL	Secure Sockets Layer
QoS	quality of service
SBC	Single Board Computer
GPIO	General Port Input-Output
HC	Host Control
DIY	Do It Yourself
ROM	Read-Only Memory
SSID	Service Set Identifier
OTA	Over-The-Air
SSH	Secure Shell
DNS	Domain Name System

Sumário

1
Introdução

Com a popularização dos circuitos integrados a automação residencial vem deixando de se tornar um mito e está se tornando uma realidade no nosso cotidiano. Automações residenciais simples como o ligamento de uma lâmpada através de uma interface computacional já é algo comum, porém, com frequência, é um tipo de automação não integrada e pouco ergonômica, contrária aos rumos da automação.

Quando falamos em automação deve-se ter em mente que soluções têm como obrigação:

- Melhorar a interação com o objeto automatizado;
- Poupar o tempo do utilizador desses objetos e/ou
- Possibilitar a execução de tarefas que não eram possíveis de serem realizadas antes da automação.

Facilmente encontramos automações que fazem apenas a troca da interface de controle do objeto desejado. Usemos como exemplo a utilização de uma interface *web* para controlar uma lâmpada ao invés do usual interruptor. Na prática, manter seu *smartphone*, *tablet* ou computador devidamente conectado na sua rede, acessar alguma página ou abrir um determinado aplicativo para controlar uma lâmpada pode ser mais custoso do que apenas acionar um interruptor na parede. Mesmo supondo que estamos trabalhando com situações ideais de conexão entre os dispositivos, para realizar algum tipo de ação, ao chegarmos em casa, por exemplo, teríamos que pegar o *smartphone*, esperar o estabelecimento da conexão com a rede, abrir o aplicativo e, somente então, realizar alguma ação. Essa situação fere todas as três regras citadas anteriormente, pois não nos poupa tempo, não melhora a interação com o objeto e nem possibilita realizar uma tarefa que não era possível antes da automação.(SCHOUTSEN, 2016)

Uma automação ideal seria aquela onde a comunicação entre pessoas e máquinas seria feita de forma amigável para ambas as partes. Entretanto, esse tipo de comunicação não existe.

A forma real mais próxima da ideal obtém-se eliminando o humano da interação, e tornando a automação o mais independente possível, cabendo ao humano apenas identificar e configurar processos a serem automatizados.(SCHOUTSEN, 2016)

1.1 Motivação

Em Schoutsen (2016), são descritos os conceitos para uma automação perfeita. Segundo o autor, as automações devem se combinar com os padrões de utilização que estão disponíveis hoje, e não substituí-lo, uma vez em que a tecnologia deve se adaptar a nós, e não o contrário. Na mesma publicação, é dado o exemplo de um cenário de automação da iluminação de um ambiente onde, quando substituído o modelo normal, com interruptores, pelo automatizado, através de APP's, acabamos tendo uma interação menos efetiva com o objeto automatizado.

O trabalho pioneiro que estabeleceu estas ideias sobre automação e interação com os computadores é o de Weiser (1991). Nele é proposto o termo *embodied virtuality* que, em tradução livre, significa virtualização incorporada e traz a ideia de um ambiente onde a integração com dispositivos inteligentes é tão natural para as pessoas que acabam não sendo notados ou esquecidos. Ainda em Weiser (1991) é feita uma analogia entre a automação e motores elétricos a fim de facilitar o entendimento da ideia de como a tecnologia pode se envolver com o nosso contexto de tal forma que a sua presença acabe sendo despercebida. Na virada do século, fábricas e indústrias possuíam apenas um motor elétrico que movimentava diversas engrenagens e polias dentro de um sistema. Com barateamento e aumento da eficiência dos motores elétricos, foi possível que cada máquina possuísse o seu próprio motor e que, em seguida, conseguíssemos colocar diversos motores em uma só máquina. De forma semelhante acontece com os dispositivos de automação. Atualmente estamos em um processo de barateamento e aumento da disponibilidade desses dispositivos. Para que automações se tornem naturais ao ambiente com o qual convivemos, alguns fatores críticos precisam ser abordados, dentre eles:

- Preços de dispositivos;
- Dispositivos com menor consumo de energia e maior qualidade e
- Aprimoramento dos softwares, protocolos e infraestrutura de rede que conectam esses dispositivos.

Há uma diversidade muito grande de dispositivos IoT (Internet of Things) no mercado que procuram satisfazer esses fatores críticos da maneira mais eficiente possível. Dentre eles, podemos destacar o NodeMCU e o ESP32, placas de desenvolvimento desenvolvidas pela Espressif. Um dos grandes benefícios dessas placas de desenvolvimento é que elas facilitam o processo de desenvolvimento de soluções e prototipagem (ESPERSSIF, 2019).

Ao observar o cenário da automação residencial atual, é possível perceber facilmente que dispositivos inteligentes de código fechado, além de altos preços, possuem uma limitação muito grande quanto aos dispositivos com os quais se é possível integrar. A exemplo da Nest que desde o dia primeiro agosto de 2019 suspendeu o seu funcionamento para novos usuários do Home Assistant e passará a trabalhar apenas com outros dispositivos da Nest e da Google (ASSISTANT, 2019k). Por outro lado, enquanto ocorre esse fechamento há também a polarização de dispositivos genéricos utilizando *chips* e placas de desenvolvimento genéricas que muitas vezes apresentam uma alta capacidade de se integrar a diversos sistemas. Entretanto, podem também exibir um funcionamento muitas vezes abaixo do aceitável, servindo apenas para fins didáticos ou de prototipação. A resiliência a falhas de infraestrutura também consta como um fator essencial para que se obtenha um dispositivo com mais disponibilidade, que esteja pronto para uso sempre que precisarmos dele.

1.2 Objetivos

Neste trabalho, a placa de desenvolvimento NodeMCU será utilizada. Este dispositivo, além de ser um genérico, possui um preço e um consumo de energia considerado razoável pelo autor. Neste trabalho desenvolvemos um software que torna o nosso dispositivo resiliente a falhas e facilmente integrável com diversos sistemas de automação. O objetivo é satisfazer o último fator crítico para dispositivos de automação, mencionado anteriormente.

O software construído será utilizado para gerenciar a iluminação de um ambiente simples como, por exemplo, um quarto. Como o intuito não é simplesmente substituir os interruptores do quarto por um novo sistema de acionamento das lâmpadas, mas sim integrar ao sistema já existente novas funcionalidades, precisamos garantir que os interruptores do ambiente também continuem funcionando como antes, ou seja, manualmente. Apesar de não ser um problema rotineiro na maioria das casas, quedas e picos de energia, falhas de conexão e o religamento inesperado de dispositivos de rede acontecem e os dispositivos inteligentes de automação devem estar preparados para lidar com essas situações, reestabelecendo seu funcionamento normal de forma transparente ao usuário. Assim, usaremos técnicas e tecnologias para tornar o dispositivo resiliente a falhas.

Dispositivos terminais de uma automação, sensores e atuadores, possuem uma capacidade de processamento limitado, tendo suas automações implementadas, normalmente, por um sistema de automação executado em um servidor. No nosso caso, o servidor é um SBC (*Single Board Computer*) Raspberry Pi enquanto que o sistema de automação é implantado mediante regras dentro do Home Assistant. O NodeMCU pode ser usado como atuador, sensor digital e sensor analógico devido a sua variedade de pinos. Para conectar o NodeMCU com a maior variedade de sistemas de automação possível, utilizamos como protocolo de comunicação o MQTT (Message Queuing Telemetry Transport), que além sem muito popular entre dispositivos IoT também é

muito leve e rápido.

1.3 Resumo

A seguir, no Capítulo 2, apresentamos as tecnologias que dão suporte ao nosso trabalho, expondo as suas características e limitações. Logo, no Capítulo 3, é mostrado como cada uma das tecnologias foi utilizada para atingirmos o objetivo proposto. É apresentado de forma incremental como o código do dispositivo atuador foi construído até a sua integração com o Home Assistant. Depois da integração, configurações de automações são mostradas, juntamente com o resultado do teste do sistema. Para concluir o trabalho, são feitas observações quanto as limitações do dispositivo desenvolvido, pontos que podem ser aprimorados em trabalhos futuros e menções a trabalhos relacionados.

2

Tecnologias de IoT

Neste capítulo serão apresentada as tecnologias de hardware e software para IoT (Internet of Things) que dão suporte ao nosso trabalho.

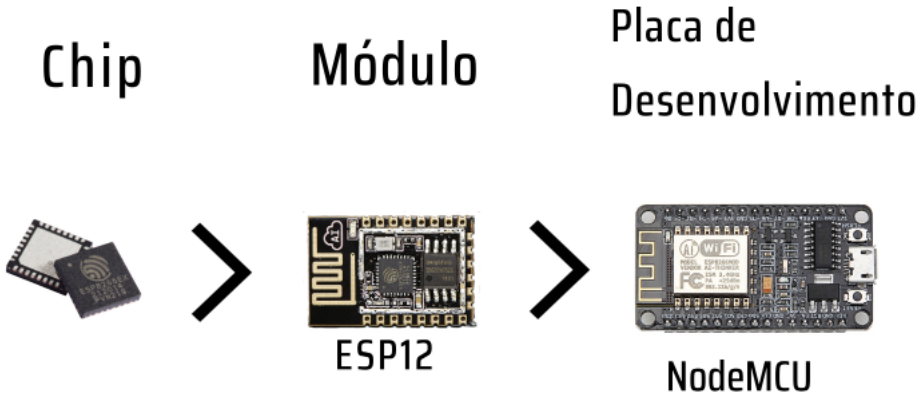
2.1 NodeMCU

Localizada em Shanghai, na China, e fundada em 2008, a Espressif é uma empresa que desenvolve soluções de hardware e software para Internet das Coisas (Internet of Things - IoT). A família mais popular de módulos da marca é, sem dúvida, o ESP que apareceu no mercado em 2014 com expectativas de revolucionar, apresentando dimensões e um poder de processamento que, até então, estavam indisponíveis.

Dentre os módulos ESP podemos destacar o ESP-01, pois é o mais popular da família e um dos mais utilizado até hoje. Atualmente são comercializados 12 módulos ESP da Espressif e todos eles utilizam o chip ESP8266, também da Espressif. O ESP8266 é um chip com wi-fi integrado que foi desenvolvido para trabalhar com um consumo muito baixo de energia e em situações adversas, chegando a suportar temperaturas de -40°C a 125°C. Com essas características, o ESP8266 é ideal para ser utilizado tanto em ambientes industriais como caseiros (ESPERSSIF, 2019).

Por se tratar de peças de hardware muito pequenas, esses módulos apresentam uma certa dificuldade no seu manuseio. Para facilitar o uso, placas de desenvolvimento são utilizadas para fazer todas as regulagens de tensões e configurações necessárias para que os desenvolvedores, técnicos e engenheiros se preocupem apenas em desenvolver. Como é possível observar na figura 1, placas de desenvolvimento utilizam o chip e o módulo junto com outros componentes eletrônicos com o intuito de aumentar as funcionalidades do dispositivo e facilitar o seu uso.

Figura 1 – Chip ESP8266, módulo ESP12 e placa de desenvolvimento NodeMCU.



Fonte: https://www.filipeflop.com/blog/guia-do-usuario-do-esp8266/

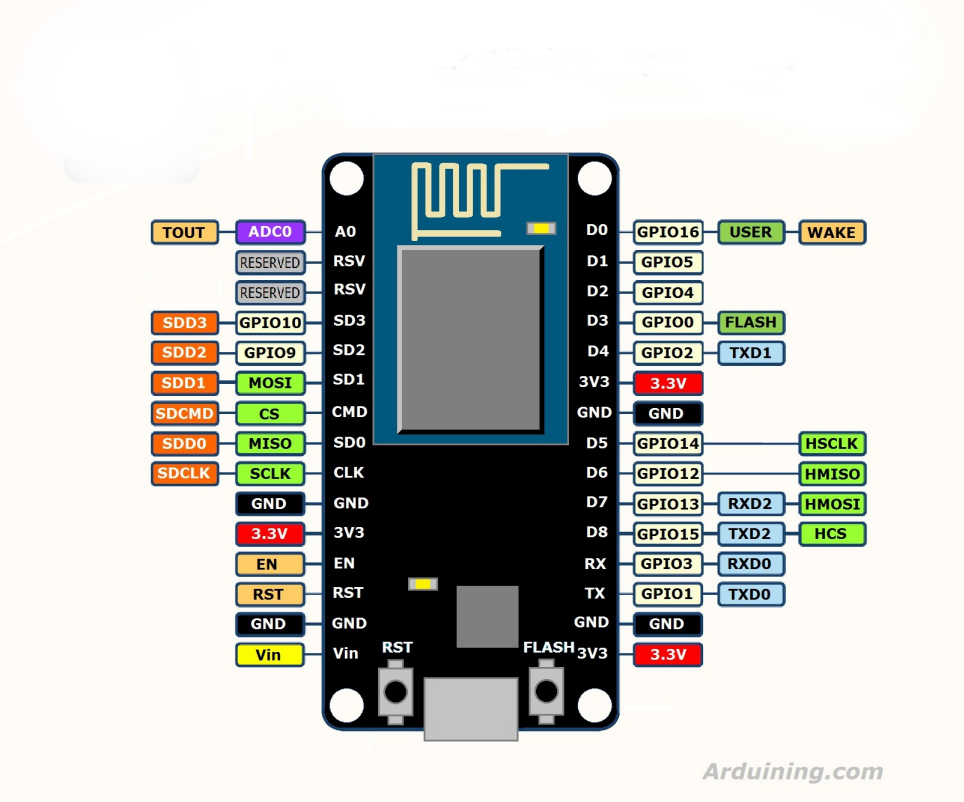
Dentre as placas de desenvolvimento Espressif podemos destacar o NodeMCU e o ESP32, quem além de todas as funcionalidades do NodeMCU ainda possui bluetooth integrado. Atualmente conseguimos encontrar o NodeMCU, que iremos utilizar na implantação da solução proposta nesse trabalho, com preços entre R$ 25,00 e R$ 40,00 reais no mercado nacional e entre R$ 9,00 e R$ 15,00 para importação, fora taxas aduaneiras.

O NodeMCU é um firmware e SDK (Software Development Kit) que permite a realização de protótipos IoT rapidamente. A placa hoje está na sua terceira versão e utiliza o modulo ESP12 para fazer processamentos e o gerenciamento de conexões. Nele já encontramos de forma nativa pinos de IO (Input/Output) digitais e analógico, ADC (Analogic Digital Conversor), PWM (Power Width Modulation), 1-Wire, conversor USB/Serial e uma porta micro USB, tudo em uma placa só, dessa forma fica muito mais fácil de alimentar a placa e conectá-la a um computador para fazer a programação.(NODEMCU, 2019)

Inicialmente, para programar o NodeMCU era utilizado a linguagem de script Lua, porém desde 2016, com o aumento da popularidade do Arduino, a própria Espressif juntamente com a comunidade vêm trabalhando em adaptar a IDE do Arduino para trabalhar com o NodeMCU. Hoje a IDE do Arduino já está madura o suficiente para para desenvolver softwares mais elaborados, robustos, com total controle do hardware e em C/C++, linguagem utilizada para o desenvolvimento das bibliotecas fornecidas pela Espressif.

Com o NodeMCU podemos dar vida aos nossos atuadores e sensores e conectá-los ao nosso sistema de forma integral. A placa possui 12 pinos de GPIO (General Port Input-Output) sendo 11 pinos digitais e 1 pino analógico, como se pode ver na figura 02. Sendo assim, podemos considerar a placa como genérica pois pode atuar com sensores e atuadores analógicos e digitais e ao mesmo tempo.

Figura 2 – Pinos do NodeMCU.



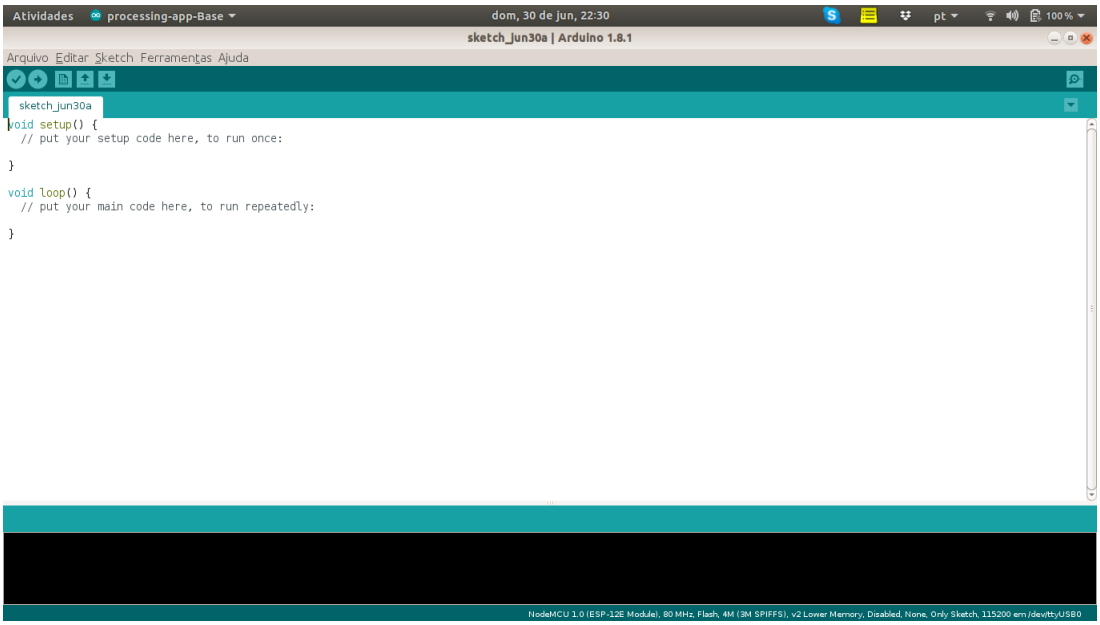
Fonte: https://iotmaker.vn/NodeMCU.html

Por se tratar de um firmware com licença MIT, o NodeMCU permite que seus produtos derivados possam ser copiados, modificados, mesclados, publicados, distribuídos e vendidos de forma livre desde que mantenham um aviso de copyright e uma copia da licença em todas as cópias do software. (MIT, 2019)

2.2 Arduino IDE

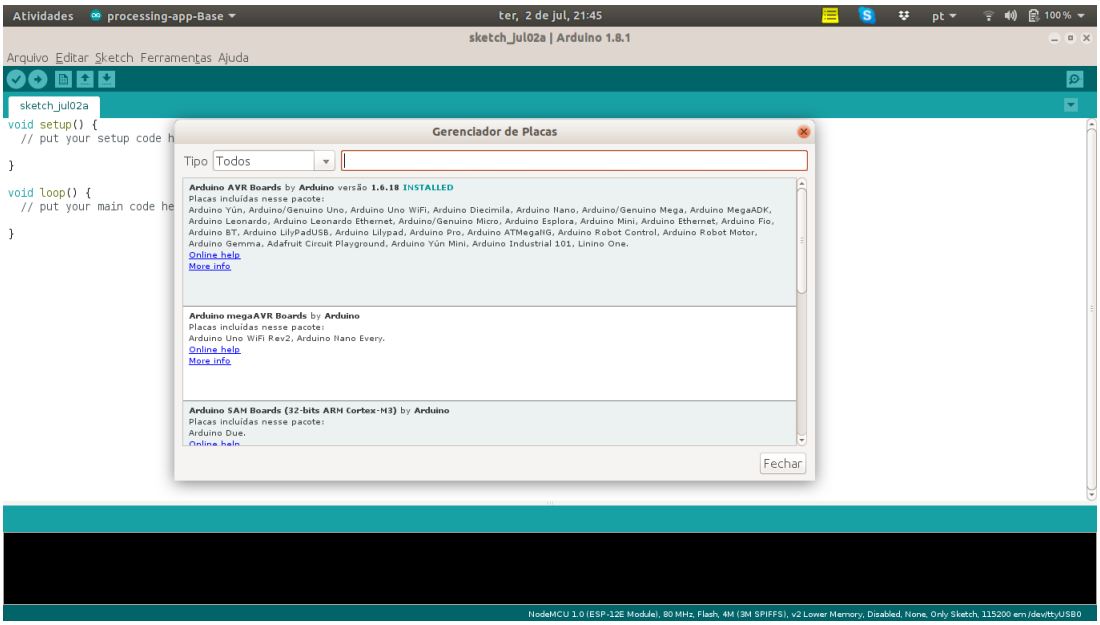
O Arduino IDE é um ambiente de desenvolvimento voltado para a programação de placas de desenvolvimento. Atualmente o Arduino IDE possui compatibilidade com aproximadamente 20 placas de desenvolvimento. Para programar placas de desenvolvimento utilizando o Arduino IDE devemos seguir uma estrutura padrão, ilustrada na Figura 3, que requer duas funções: `setup` e `loop`. No `setup` implementamos todas as configurações iniciais da placa, que são executadas única e exclusivamente após o `boot` da placa, enquanto que no `loop` colocamos o código que é executado repetidamente após a inicialização. Tanto o `setup` quanto o `loop` são implementados em C/C++.

Figura 3 – Arduino IDE com novo projeto.



Após a criação do projeto, precisamos especificar para qual dispositivo ou placa de desenvolvimento iremos programar. Dentro do Arduino IDE existe um gerenciador de placas de desenvolvimento com as quais podemos incrementar a plataforma. Fazendo o download do firmware da placa que será utilizada, é necessário especificar alguns detalhes de funcionamento e de hardware para uma compilação apropriada. A Figura 4 mostra como o Arduino IDE nos permite gerenciar as placas com as quais iremos trabalhar.

Figura 4 – Gerenciador de placas do Arduino IDE.



Após escolhermos, no Arduino IDE, a placa de desenvolvimento que iremos utilizar, temos que ir no menu de ferramentas e especificar os parâmetros da placa que estamos utilizando. Dentre os parâmetros, temos que especificar a velocidade de upload da placa, a frequência da

CPU (Central Process Unit) utilizada, o tamanho e o tipo da memória flash e quais áreas que irão ser sobrescrita durante o upload.

2.3 MQTT

Mesmo o NodeMCU possuindo um chip com desempenho melhor que o da maioria dos chips de mesma finalidade no mercado e até hoje ser um dos mais eficientes, ele ainda é um processador bastante limitado e precisa trabalhar com protocolos leves, como o MQTT, para que consiga fornecer um desempenho satisfatório.

Por outro lado, a arquitetura *publish-subscribe* vem ganhando muito espaço no desenvolvimento de dispositivos IoT, uma vez em que com essa arquitetura os sensores e atuadores não precisam ter nenhum conhecimento dos demais para funcionar de forma integral no ecossistema. O modelo *publish-subscribe* cria uma entidade central, responsável por fazer os processamentos mais pesados do sistema, comumente chamado de *broker*. Devidamente conectado ao *broker*, os atuadores e sensores irão publicar e se subscrever a tópicos que são pertinentes ao seu contexto, similar a uma assinatura de um RSS em uma página de notícias. (OGLIARI, 2017)

O *publish-subscribe* define um modelo de comunicação um-para-muitos entre objetos de tal forma que quando um objeto muda de estado, todos os seu dependentes são atualizados automaticamente. E necessário que todos os objetos conheçam o broker em tempo de compilação e, sendo assim, todos esses objetos, que não se conhecem em tempo de compilação, conseguem se acoplar e desacoplar a qualquer momento em tempo de execução. Dessa maneira conseguimos uma estrutura abstrata com um baixo acoplamento.

Seguindo a arquitetura *publish-subscribe* e tendo em vista a limitação de processamento de sensores e atuadores de uma automação, é necessário o uso de um protocolo eficiente de comunicação. MQTT (Message Queuing Telemetry Transport) é um protocolo *publish-subscribe* M2M(Machine-to-Machine) muito leve, simples e otimizado para redes TCP/IP não confiáveis ou de alta latência, ideal para se trabalhar com dispositivos de IoT. Apesar da sua simplicidade, o protocolo é muito versátil chegando a ser usado em comunicações via satélite para locais com pouca largura de banda e até em pequenas *fogs*, arquitetura descentralizada que concentra seu poder de processamento mais próximo dos limites da rede. (ORG, 2019)

O MQTT foi criado em 1999 pelos engenheiros Dr Andy Stanford-Clark e Arlen Nipper. Atualmente o MQTT está na sua quinta versão e utiliza por padrão as portas 1883 e 8883 para trabalhar sem e com SSL (Secure Sockets Layer), respectivamente. Para trazer mais segurança para o protocolo, desde a versão 3.1 é possível adicionar um nome e senha para as mensagens transmitidas com o protocolo. Utilizar o MQTT sob o SSL é possível, porém não é recomendado, uma vez em que o MQTT se propõe em fazer uma transmissão leve de mensagens e o SSL é um protocolo relativamente pesado e que gera uma sobrecarga significativa na rede. (MQTT, 2019)

Para configurar o *broker* da arquitetura *publish-subscribe* para trabalhar com o MQTT precisamos apenas instalar um servidor MQTT. Com um *broker* devidamente instalado, os tópicos aos quais os sensores e atuadores irão publicar e subscrever irão ser criador e apagados dinamicamente de acordo com o seu uso e configuração. Atualmente temos a disposição alguns servidores MQTT de código aberto no mercado como: CloudMQTT e o Mosquitto, da Eclipse Foundation. (CLOUDMQTT, 2019)

Para gerenciar tanto os objetos conectados ao *broker* quanto conexões, pacotes de controle são transmitidos pela rede. Através dos pacotes de controle fica garantido que todas as mensagens enviadas irão chegar ao *broker* e torna possível estabelecer níveis de QoS (Quality of Service). Dentre os pacotes trafegados entre os objetos podemos destacar:(IBM, 2019)

- CONNECT;
  - Pacote enviado pelo cliente para se conectar ao broker.
- CONNACK;
  - Reconhecimento da solicitação de conexão.
- PUBLISH;
  - Pacote enviado do cliente para o broker publicando uma mensagem.
- PUBACK;
  - Reconhecimento da solicitação de publicação.
- PUBREC
  - Publicação recebida. QoS 0, a mensagem não é armazenada.A mensagem é entregue no máximo uma vez ou não é entregue. A mensagem poderá ser perdida se o cliente for desconectado ou se o servidor falhar.
- PUBREL
  - Publicação publicada. QoS 1, modo de transferência padrão. A mensagem é sempre entregue pelo menos uma vez. Se o emissor não receber uma confirmação, a mensagem será enviada novamente.
- PUBCOMP
  - Publicação completada. QoS 2, a mensagem é sempre entregue exatamente uma vez e é armazenada localmente no emissor e no receptor até ser processada.
- SUBSCRIBE

  - Pacote enviado para subscrição em determinado tópico.

- SUBACK
  - Pacote de reconhecimento de subscrição.
- UNSUBSCRIBE
  - Pacote enviado para cancelar subscrição em determinado tópico.
- UNSUBACK
  - Pacote de reconhecimento de cancelamento de subscrição.
- DISCONNECT
  - Pacote enviado para se desconectar do broker.

Para facilitar a utilização do protocolo MQTT com o NodeMCU, existe a biblioteca de código aberto PubSubClient. Com essa biblioteca é possível trocar mensaagens MQTT com um *broker* de forma simplificada.pubsubclient (2019) Assim como para o MQTT, existem outras bibliotecas de código aberto que ajudam a tratar e implementar protocolos necessários para conseguirmos conectar e aumentar a precisão do sensoriamento do NodeMCU. Dentre outras bibliotecas de código aberto que são amplamente utilizadas e estão em constante produção, podemos destacar: ESP8266WiFi, WiFiManager, ArduinoOTA, Bounce2.

2.4 Home Assistant

O Home Assistant é uma alternativa gratuita e open source utilizada para desenvolver sistemas de automação residencial descentralizados que controlam atuadores, interpretam dados coletados por sensores, implementam regras de automação e gerenciam a comunicação entre dispositivos. O Home Assistant pode rodar em computadores Linux, diretamente em SBCs (*Single Board Computers*) com o sistema operacional dedicado Hass.io ou em qualquer outro ambiente que tenha Python e suas dependências instaladas. Segundo Assistant (2019e), dentre as vantagens do Home Assistan pode-se destacar:

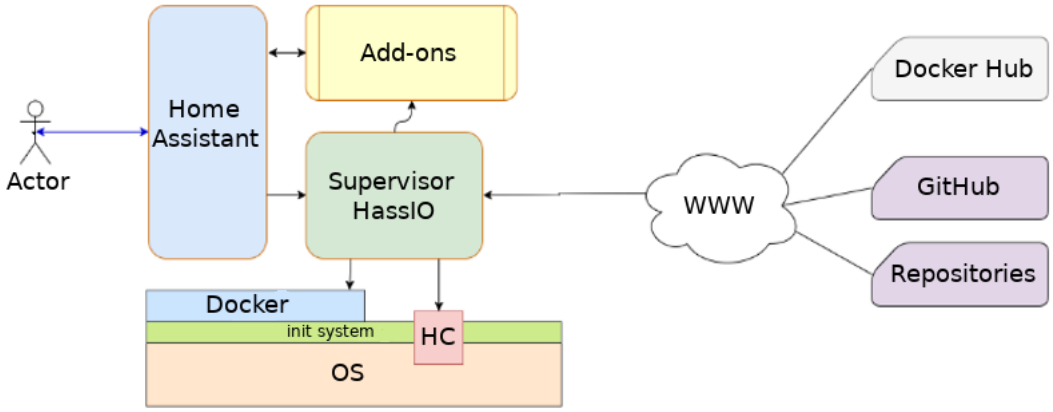
- É grátis e Open Source;
- É otimizado para rodar em SBCs;
- Permite que todas as automações sejam implantadas localmente;
- É de Fácil instalação;
- Possui interface web interativa para controle;

- Oferece *rollback* para versões antigas e
- É escalável, permitindo trabalhar com *add-ons*.

Para facilitar a instalação e utilização do Home Assistant foi desenvolvido o Hass.io, um sistema operacional customizado e já configurado para executar o Home Assistant. Como é possível observar na figura 5, o Home Assistant é uma aplicação gerenciada por um supervisor que roda dentro de um container Docker, que por sua vez, está sendo executado por um sistema operacional desenvolvido especificamente para SBCs. Com o supervisor, temos uma API que além de gerenciar os processos do Home Assistant também vai ser responsável por realizar atualizações do sistema. O HC (*Host Control*) é a interface responsável por gerenciar todas as configurações de rede do sistema, permitindo que façamos configurações de rede e hardware diretamente pela aplicação do Home Assistant.(ASSISTANT, 2019a)

Como o Home Assistant tem o intuito de ser uma aplicação escalável e que trabalha sob demanda, o módulo *add-on* é responsável por adicionar funcionalidades ao nosso sistema. A loja virtual de *add-ons* do Home Assistant é muito vasta e permite adicionar componentes de terceiros pelo github além dos addons oficiais que adicionam várias funcionalidade ao sistema, por exemplo: MQTT Home Broker, servidor DNS (Domain Names System), servidor de arquivos, servidor SSH (Secure Shell), etc..

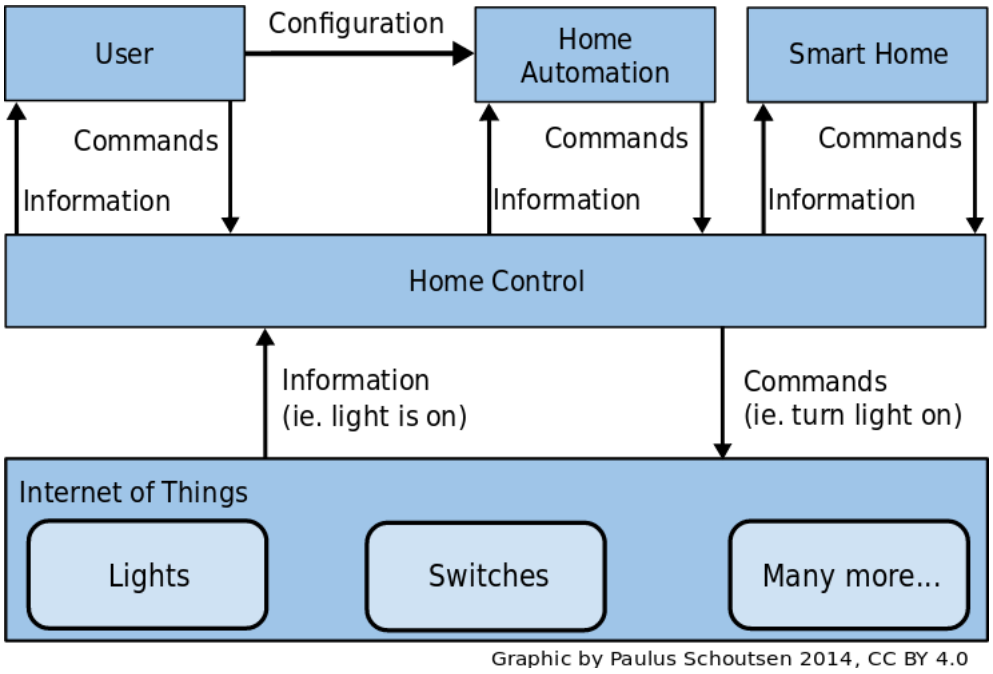
Figura 5 – Arquitetura do Hass.io



Fonte: https://developers.home-assistant.io/docs/en/architecture_hassio.html

Como é possível ver na figura 6, o Home Assistant possui uma arquitetura com 5 módulos, onde `Home Control` é responsável por coletar informações e controlar atuadores, `User` faz toda a gestão de usuários do sistema, `Home Automation` aciona comandos baseado em configurações fornecidas pelos usuários, `Smart Home` aciona comandos baseado nos estados de outros atuadores e sensores e `Internet of Things` são os atuadores e sensores em si.

Figura 6 – Módulos do Home Assistant.

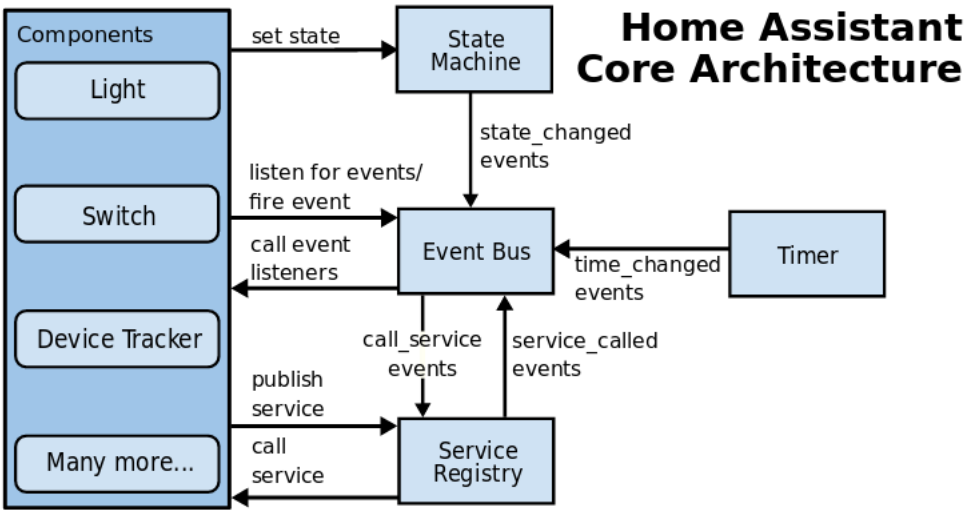


Fonte: hhttps://developers.home-assistant.io/docs/en/architecture_index.html

Para uma melhor compreensão do funcionamento Home Assistant precisamos definir os conceitos: evento, ação e serviço. Um evento é o acontecimento de algo observável pelo nosso sistema, alterando ou não o estado de algum atuador ou sensor; ação é um evento que é disparado uma vez que todas as suas condições necessárias para agir sejam satisfeitas e serviços são métodos executados para realizar ações.

Observando o `Home Control` mais detalhadamente, ainda podemos dividi-lo em partes menores para entender melhor como as ações são processadas pelo Home Assistant. Como é possível observar na figura 7, podemos dividir o `Home Control` em 4 módulos, onde: `Event Bus` é o módulo responsável por escutar e disparar eventos para os atuadores e manter a sincronia do sistema com o mundo real, uma vez que quando se deseja fazer a alteração do estado de algum atuador ou sensor, é o `Event Bus` que dispara as ações. `State Machine` é o modulo que mantém um registro de todos os estados dos atuadores e comunica ao `Event Bus` quando algum estado é alterado. `Service Registry` faz a chamada de serviços que por sua vez realizam ações nos atuadores. O `Timer` é o componente responsável por atualizar a hora no `Event Bus` emitindo um evento de mudança de tempo a cada um segundo para o `Home Control`. (ASSISTANT, 2019b)

Figura 7 – Home Control Core.



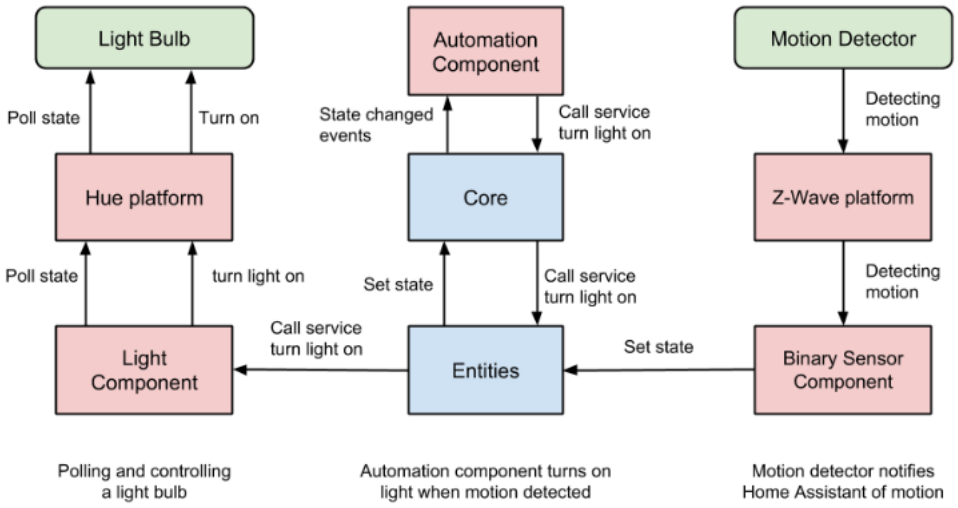
Fonte: https://developers.home-assistant.io/docs/en/architecture_index.html

Com essa arquitetura distribuída, fica implícito uma alternativa do Home Assistant para tratar falsos positivos/negativos. Uma vez em que haja a chamada de algum serviço pelo `Service Registry`, porém não haja nenhuma alteração no `State Machine` é porque houve alguma falha de comunicação ou algum outro erro durante a execução de alguma ação.

Como existem diversos fabricantes de dispositivos inteligentes para automação e cada um utiliza os seu próprios protocolos e padrões de comunicação, é necessário um tratamento pelo Home Assistant para que seja possível se comunicar com a maior variedade possível de dispositivos e fazer com que eles se comuniquem entre si.

Como é possível observar na figura 8, para se comunicar com o Home Assistant precisamos dos atuadores e sensores, na imagem representados pelo `Light Bulb` e o `Motion Detectos`, assim como também da camada de software responsável por dar inteligência aos atuadores e sensores, na imagem representados por `Hue Plataform` e `Z-Wave Plataform`, esses softwares são executados pelo próprio atuador ou sensor. Por parte do Home Assistant, precisamos de uma camada de software responsável por implementar os protocolos utilizados pelos atuadores e sensores e que trabalhe seguindo as mesmas regras destes, na imagem representado por `Light Component`. Uma vez em que temos os sensores e atuadores adaptados para o Home Assistant, precisamos representar cada instância destes para que possamos trabalhar com eles utilizando os conceitos de evento, ação e serviço, na imagem essa camada é representada por `Entities`.

Figura 8 – Componentes(atuadores/sensores) Home Assistant



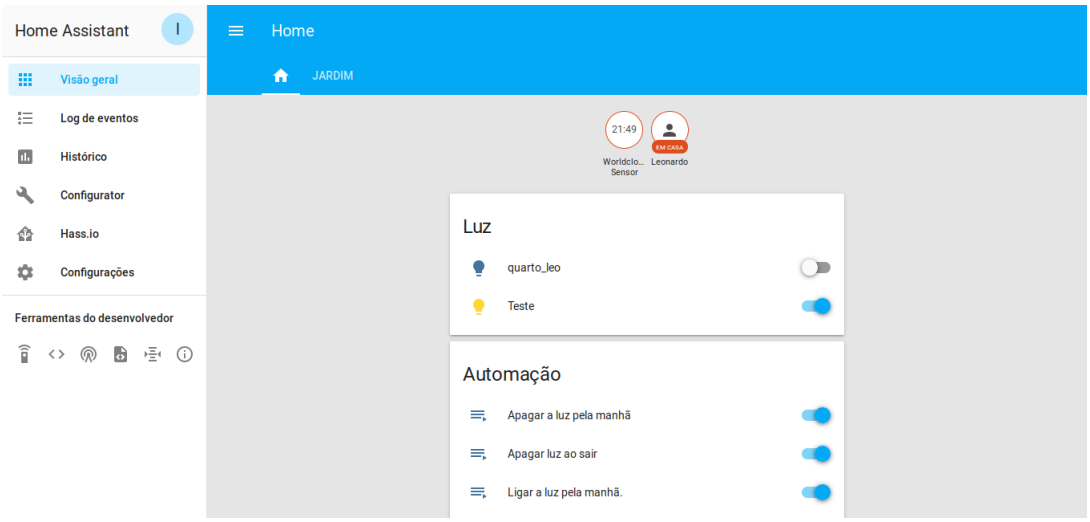
Fonte: https://developers.home-assistant.io/docs/en/architecture_components.html

No Home Assistant, como foi explicado anteriormente, precisamos de uma camada de software para se comunicar com protocolos e padrões dos atuadores. Tais softwares podem ser incrementados no Home Assistant através de *add-ons* baixados na loja virtual do sistema. Com todas as dependências sanadas, cabe a nó fazer as configurações de automação e a instanciação de todos os atuadores e sensores, quando não identificados automaticamente. Existem dois arquivos principais de configuração no Home Assistant: `settings.yaml` e `automations.yaml`. No arquivo de `settings.yaml` especificamos instâncias de todas as entidades do sistema e no arquivo `automations.yaml` definimos padrões de comportamento, também chamadas de automações, para as entidades do sistema.

Mesmo utilizando o formato yaml, desenvolvido para serializar dados de uma forma legível para humanos, ainda pode ser bastante complicado definir as configurações e automações do sistema. Com o intuito de tornar esse processo mais intuitivo e rápido, na versão v0.45 do Home Assistant foi disponibilizado um editor de automações através de uma interface web. Na interface web são expostas, de forma simples e clara, todas as entidades e sensores disponíveis no sistema. Na interface web precisamos definir quatro parâmetros para configurar uma automação, são eles: o nome, os gatilhos, as condições e as ações. Tais parâmetros serão explicados e exemplificados no próximo capítulo do trabalho. (ASSISTANT, 2019d)

Como pode ser visto na figura 9, além do editor de automações, a interface web do Home Assistant trás uma visão geral das entidades e automações do nosso sistema, o que possibilita ao usuário um acesso rápido e prático para o controle das entidades. A página web ainda facilita o acesso a quase todas as funcionalidades do sistema e à loja virtual de *add-ons*. A interface web pode ser acessada através da porta 8123 do dispositivo no qual foi instalado o sistema.

Figura 9 – Visão geral da interface web do Home Assistant



3

Integrando o NodeMCU com o Home Assistant

Neste capítulo será apresentado uma integração resiliente a falhas de redo do atuador genérico NodeMCU com o Home Assistant utilizando as tecnologias descritas no capítulo anterior

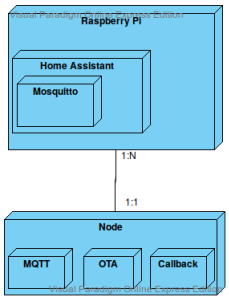
3.1 Arquitetura

Para uma melhor compreensão de como cada tecnologia apresentada anteriormente irá contribuir para o sistema proposto, nesta sessão iremos dar uma explicação da arquitetura da integração dessas tecnologias no sistema.

O sistema tem três tipos de componentes fundamentais: *broker*, sensores e atuadores. O *broker* é o coração do sistema, onde são armazenados os estados de cada atuador e as leituras dos sensores além de enviar comandos para os atuadores agirem. Os sensores interpretam dados do ambiente enquanto os atuadores realizam ações para interagir com o ambiente. Todas as três partes do sistema utilizam o protocolo MQTT para se comunicar.

Como é possível observar no diagrama de instalação da Figura 10, no centro da nossa arquitetura temos o Home Assistant e o Mosquitto, uma implementação do *broker* MQTT, ambos instalados em um Raspberry Pi. Como sensor e atuador do sistema utilizaremos o NodeMCU, que irá interpretar as posições dos interruptores e controlar os relés. A arquitetura do sistema é simplificada, uma vez em que utilizamos um dispositivo genérico que pode funcionar tanto como sensor quanto atuador, conhecendo os outros dispositivos do sistema única e exclusivamente através do Home Assistant.

Figura 10 – Diagrama de instalação do sistema.



3.2 Programação do NodeMCU

Nessa sessão será apresentada a implementação do software que faz do NodeMCU um dispositivo estável e resiliente a falhas. O código fonte compilado no NodeMCU será apresentado de maneira gradual de acordo com a relevância das funcionalidades de cada seção para o funcionamento do dispositivo integrado com o Home Assistant e os modelos atuais de controle, interruptores.

3.2.1 Efeito *Bouncing*

Não diferente de um botão regular, os interruptores também sofrem com os problemas do efeito *bouncing* quando lidos sem nenhum tratamento. O efeito *bouncing* é uma trepidação elétrica que acontece quando quando pressionamos um botão, *switch* ou interruptor, pois, diferente do que pode ser suposto, a alteração do estado de um botão não varia instantaneamente, mas se estabiliza após alguns pulsos elétricos para cima e para baixo, o que pode comprometer a leitura do sensor. Podemos tratar o efeito *bouncing* apenas estabelecendo um intervalo de tempo suficiente para a estabilização do sinal entre uma leitura e outra ou através de bibliotecas dedicadas para isso. Nesse trabalho iremos utilizar a biblioteca de código aberto Bounce2 para solucionar esse problema.

A biblioteca Bounce2 provê a classe `Bounce`. Para nosso NodeMCU, precisamos apenas criar uma instância dela, escolher um pino da nossa placa para atuar como o sensor que irá ser ligado ao interruptor e escolher um intervalo de tempo para cada leitura, em milissegundos. Todas essas configurações devem ser feitas dentro da função `setup`. Iremos utilizar o GPIO 5 para realizar a leitura do interruptor.

```
1 void setup(){
2 ...
3     int interruptor = 5;
4     int debounce_interval = 5
5
6     Bounce debouncer = Bounce();
7     debouncer.attach(interruptor);
8     debouncer.interval(debounce_interval);
9     lastState = debouncer.read();
```

```
...
}
```

Como a qualquer momento um interruptor pode mudar de estado, a verificação dessa alteração deve ser sempre observada. Colocando a leitura do interruptor dentro da função `loop` conseguimos que essa leitura seja feita de forma quase constante, variando apenas de acordo com o tempo que o `loop` leva para executar o seu corpo.

Apenas dois pinos são utilizados para fazer a automação de uma lâmpada controlada por um interruptor. No código anterior, o pino 5 do NodeMCU é responsável por fazer a leitura do estado do interruptor tratando o efeito *bouncing*. No código abaixo, o pino 16 é vinculado ao relé que controla fisicamente o ligamento da lampada. Toda vez que o estado do interruptor é alterado o estado do relé também será alterado e uma mensagem será enviada para o *broker* MQTT para atualizar o estado da lâmpada no sistema. O envio da mensagem MQTT é realizado pelo método `public` do objeto `client`, mais detalhes quanto a comunicação do atuador com o *broker* MQTT serão expostos nas próximas seções.

```
int SwitchedPin = 16;

void loop(){
...
  debouncer.update();
  if(lastState != debouncer.read()) {
    lastState = debouncer.read();
    if(digitalRead(SwitchedPin)){
      client.publish(COMMAND_TOPIC,"ON");
      digitalWrite(SwitchedPin, LOW);
    }
   else{
       client.publish(COMMAND_TOPIC,"OFF");
       digitalWrite(SwitchedPin, HEIGH);
    }
  }
...
}
```

3.2.2 Conexões Wi-Fi

Quanto a conexão Wi-Fi, o NodeMCU trabalha no padrão 802.11 g/n e pode assumir tanto o papel de cliente como o de ponto de acesso (*Acess Point, AP*). Para fornecer uma melhor interação com o usuário, iremos utilizar os dois modos disponíveis para conectar o NodeMCU na rede. A conexão com a rede Wi-Fi tradicional vai funcionar da seguinte maneira: primeiramente o NodeMCU atua no modo AP fora da rede caseira, quando não possui nenhum dado de conexão armazenado em sua memória *flash*. Nesta situação, o NodeMCU opera como um servidor *web* e, quando um segundo dispositivo se conecta a ele, uma interface *web* apresentará o resultado de

uma varredura das redes disponíveis, então o usuário pede escolher a qual rede o NodeMCU deve se conectar como cliente e, caso a conexão seja efetivada, o NodeMCU sai do modo AP e se conecta como cliente à rede doméstica determinada pelo usuário.

Para nos auxiliar no processo de conexão e alternância de modos de operação do NodeMCU, iremos utilizar a biblioteca WiFiManager. A WiFiManager é uma biblioteca de código aberto desenvolvida por entusiastas e profissionais da área de IoT. É esta biblioteca que fornece a funcionalidade do modo AP, assim ela implementa a interface *web* que varre as redes WiFi disponíveis e conecta o NodeMCU a rede selecionada. (TZAPU, 2019)

Para utilizarmos o WifiManager precisamos das seguintes configurações: importar as bibliotecas WiFiManager e ESP8266WiFi, mesma biblioteca utilizada pelo WiFiManager, porém precisaremos de alguns métodos dela que não estão disponíveis no WiFiManager, fazemos uma instância do WifiManager e definimos um nome da rede que será criada quando o NodeMCU estiver em modo AP. O atuador, NodeMCU, tem seu nome definido através o método `hostname` da biblioteca ESP8266WiFi no `setup` e parametrizado com a variável global do sistema, `HOSTNAME`, onde é definido o seu nome. A nomeclatura dos atuadores também será utilizada para definir padrões para a comunicação, como será mostrado nas sessões subsequentes.

```
#include <ESP8266WiFi.h>
#include <WiFiManager.h>

const String HOSTNAME = "TESTS";
WiFiManager wifiManager;

void setup(){
...
     WiFi.hostname(HOSTNAME);
...
}

void loop(){
...
    wifiManager.startConfigPortal(HOSTNAME.c_str());
...
}
```

Por padrão, um tempo máximo de 180 segundos é definido que o NodeMCU atue em modo AP Wi-Fi, e após esse tempo o AP é desabilitado e o código do `loop` será executado até chegar na próxima iteração. A biblioteca WifiManager tem como proposta fazer com que nenhum dado de conexão precise ser inserido no código e tratar falhas de conexão sempre com o auxílio do usuário, uma vez em que o dispositivo volta para o modo AP e o usuário precisa intervir para que haja uma nova conexão. Essas características da biblioteca são interessante, mas para uma automação de alto nível acaba sendo inviável, pois, caso haja problemas na infraestrutura da rede todos as conexões WiFi de todos os atuadores NodeMCU terão que ser reconfigurados de

forma manual e, como discutido anteriormente, a proposta de uma automação de alto nível é ser invisível para o usuário.

Supondo que haja uma queda de energia ou o roteador seja reiniciado por qualquer motivo, os atuadores, NodeMCU, irão todos retornar para o modo AP, mas para automatizar esse processo de reconexão com o Wi-Fi será utilizada a biblioteca ESP8266WiFi, responsável por realizar as conexões de rede do chip 8266. A biblioteca ESP8266WiFi é de código aberto e foi desenvolvida em C++ se baseando na biblioteca Arduino WiFi library, padrão para conexões de módulos Wi-Fi do arduino. Com o passar do tempo e devido ao uso distinto de cada biblioteca e de cada chip, a biblioteca ESP8266WiFi atualmente é muito mais rica e possui mais funcionalidades que a Arduino WiFi library (ARDUINO, 2019a)

Com a biblioteca ESP8266WiFi importada para o projeto, temos a disposição uma classe já instanciada chamada Wifi e por padrão basta executar a função `begin` dessa classe passando como parâmetro o *SSID (service set identifier)* e a senha da rede a qual se deseja conectar. Com todas as funcionalidades dessa biblioteca a disposição, foi traçada uma estratégia para o uso de cada uma de forma a usar uma interface de conexão apenas no primeiro acesso a uma rede doméstica.

Quando fazemos uma conexão com o NodeMCU a rede wifi em modo cliente, os dados de SSID e senha são armazenados na memória flash, não volátil e eletronicamente reprogramável, do chip ESP8266, porém foi observado que sempre que acionamos a página *web* para fazer uma conexão manual esses dados são apagados e sobrescritos com os dados da nova conexão caso ela seja feita com sucesso ou apagados caso não haja sucesso na conexão, ou seja, como mencionado anteriormente, a biblioteca WifiManager utiliza os dados salvos na mémoria flash do chip ESP8266 para decidir se irá realizar uma conexão automática com a rede domestica ou abrir uma interface *web* para efetuar uma conexão de forma manual.

Fica evidente que quando esses dados de conexão são apagados da memória flash fica impossível realizar uma reconexão automática com a rede doméstica apenas utilizando a biblioteca WifiManager. Para resolver esse problema iremos utilizar a interface *web* apenas para realizar a primeira conexão do NodeMCU à rede domestica e as demais conexões que sejam necessárias, devido a qualquer instabilidade na rede, será realizada através da da biblioteca ESP8266WiFi. Foi observado, a partir de testes práticos, que em intervalos de dois minutos temos um resultado satisfatório para tentar realizar uma reconexão com a rede doméstica sem afetar a leitura dos interruptores, dessa forma conseguimos realizar a reconexão com a rede doméstica e reestabelecer o funcionamento normal do atuador sem comprometer a funcionalidade dos interruptores enquanto o chip está tentando se conectar ou desconectado.

Para implementarmos essa funcionalidade no NodeMCU é definido uma variável global, `period`, com o período em que irá haver uma tentativa de reconexão caso o NodeMCU esteja desconectado e adicionamos o código abaixo dentro do *loop*. Para realizar a contagem de tempo entre uma tentativa e outra de reconexão é utilizado o método millis() que faz uma contagem de

quantos milissegundos se passaram desde que a aplicação foi inicializada. Um *overflow*, retorno para o valor zero, da quantidade de milissegundos decorridos acontece, aproximadamente, a cada cinquenta dias e não trás nenhum erro significativo para a contagem de tempo da aplicação proposta.(ARDUINO, 2019b)

```
unsigned long time_now = 0;
int period = 120000;

void loop(){
...
    if (WiFi.status() != WL_CONNECTED){
        if(( !WiFi.psk().equals("")) ){
            if(first_boot){
                first_boot = false;
                WiFi.begin( WiFi.SSID().c_str(), WiFi.psk().c_str());
                delay(3000);

            }
            else{
              if(millis() > time_now + period){
                time_now = millis();
                WiFi.begin( WiFi.SSID().c_str(), WiFi.psk().c_str());
                delay(3000);
              }
            }
        }
        else{
          wifiManager.setConfigPortalTimeout(180);
          wifiManager.startConfigPortal(HOSTNAME.c_str());
        }
...
}
```

3.2.3 Comunicação Através do Protocolo MQTT

Após devidamente conectado a uma rede Wi-Fi, os atuadores precisaram se comunicar entre si para dar vida ao sistema. Por se tratar de um protocolo leve e otimizado para dispositivos IoT, nesse projeto será utilizado o protocolo MQTT como padrão para a comunicação entre os dispositivos. Como mencionado anteriormente, a arquitetura *publish-sbscrive* possui um *broker* que faz o tratamento dos tópicos MQTT e das mensagens recebidas. Neste projeto o *broker* escolhido para ser utilizado foi o Mosquitto.

Como mencionado anteriormente, para utilizarmos o protocolo MQTT no NodeMCU será utilizada a biblioteca PubSubClient. A biblioteca PubSubClient é utilizada para fazer troca de mensagens de maneiras simples entre um cliente e um *broker* MQTT. (KNOLLEARY, 2019) Para suportar a troca de mensagens de forma segura utilizando o protocolo MQTT, precisamos

desenvolver funções que irão se comunicar com o servidor e tratar as mensagens recebidas de forma assíncrona.

Como é possível observar no código abaixo, primeiramente definimos se a conexão com o servidor é autenticada ou não. No broker MQTT podemos definir dois tipos de conexão: autenticada e anônima. Em conexões autenticadas o cliente é capaz de receber e publicar mensagens a tópicos do servidor enquanto que em conexões anônimas os clientes podem apenas subscrever a tópicos. Por uma questão de segurança, iremos trabalhar com uma configuração onde o broker aceita somente conexões autenticadas, mas mesmo trabalhando com conexões anônimas, também estaríamos suficientemente seguros pois elas não conseguem publicar mensagens em tópicos, logo não conseguem controlar atuadores.

Inicialmente precisamos definir um padrão para os tópicos da rede. Para essa solução foi elaborada uma arquitetura que organiza em três tópicos distintos aos quais o atuadores irão se subscrever, são eles: `system/log`, `hostname/system/set` e `hostname/set`. No `system/log`, como o nome sugere, será para o *log* do sistema, ou seja, onde é publicada todas as mensagens que os atuadores receberem dentro dos tópicos de comando e de sistema, dessa forma conseguimos ter de forma fácil e centralizada as informações de todos os atuadores da rede. Os outros dois tópicos são compostos pelo nome do atuador, que como vimos anteriormente é setado em uma variável global do código, mais o tipo de ação que se deseja realizar. Para darmos comandos de alteração de estado aos atuadores, publicamos mensagens no tópico `hostname/set` e para darmos comandos referentes as funcionalidades do atuador, como reiniciar ou reabrir a interface *web* para trocar de rede, publicamos no tópico `hostname/system/set`

Uma vez em que temos bem definidos os tipos de conexão que irão ser aceitas pelo *broker* uma arquitetura de estrutura dos tópicos, precisamos efetivar a conexão com o *broker* MQTT. Para se conectar com o *broker* desenvolvemos a função `checkMqttConnection`. Para generalizar a usabilidade desse método, como é possível ver no código abaixo, ele verifica se a conexão é autenticada ou não através da variável booleana MQTT_AUTH, e em casos verdadeiros ele tenta autenticar a conexão passando como parâmetro os valores de *username* e *password* definidos anteriormente, caso contrário tenta estabelecer uma conexão anônima. Pode-se observar que caso a conexão não consiga ser estabelecida a função retorna `False`, mas caso a conexão seja estabelecida com sucesso além de retornar `true` também já é feita as subscrições aos tópicos configurados anteriormente, e a correspondente publicação no tópico de *log*, registrando a nova conexão no *broker*.

Uma vez conectado ao *broker*, o nosso sistema ainda está suscetível a falhas, pois caso haja algum problema com o servidor, tentativas de reconexão sem cautela ou em excesso poderão comprometer a compatibilidade com os modelos atuais e fazer com que o desempenho do funcionamento dos interruptores não seja aceitável ou deixe de funcionar. Para tratar esse problema iremos utilizar uma abordagem semelhante a que foi utilizada para tratar as reconexões automáticas a rede Wi-Fi. Utilizando o método milis(), iremos controlar o tempo entre as

tentativas de reconexão com *broker*. Assim como na conexão Wi-Fi, definimos um tempo de 2 minutos entre as tentativas de se reconectar ao *broker*.

```
#define MQTT_AUTH true
#define MQTT_USERNAME "mqttuser"
#define MQTT_PASSWORD "4ccd5a"

WiFiClient wclient;
PubSubClient client(MQTT_SERVER, 1883, wclient);

const char *LOG_TOPIC = "system/log";
const char *SYSTEM_TOPIC = "TESTS/system/set";
const char *COMMAND_TOPIC = "TESTS/set";

bool first_boot_mqtt = true;

bool checkMqttConnection() {
  if (!client.connected()) {
      if (MQTT_AUTH ? client.connect(HOSTNAME.c_str(), MQTT_USERNAME,
    MQTT_PASSWORD) : client.connect(HOSTNAME.c_str())) {
        Serial.println("CONECTADO AO BROKER MQTT " + String(MQTT_SERVER)
    );
        client.publish(LOG_TOPIC, String("CONNECTED_" + HOSTNAME).c_str
    ());
        client.subscribe(SYSTEM_TOPIC);
        client.subscribe(COMMAND_TOPIC);
      }
 }
  return client.connected();
}

public void loop(){
...
    if(first_boot_mqtt){
      first_boot_mqtt = false;
      checkMqttConnection();
    }
    else if(client.connected()){
       client.loop();
    }
    else{
      if(millis() > time_now + period){
          time_now = millis();
          checkMqttConnection();
      }
    }
...
}
```

Uma vez em que temos o atuador devidamente conectado a rede Wi-Fi e ao *broker* precisamos definir uma função que será executada toda vez que uma nova mensagem chegue a algum tópico aos quais o atuador se subscreveu. Na solução apresentada esse método é chamado de *callback*. O *callback* é responsável por interpretar as mensagens recebidas nos tópicos e performar uma determinada ação com base no conteúdo dessa mensagem.

Como mostrado anteriormente, foi definido dois tópicos para cada atuador receber comandos, logo, além do tratamento do conteúdo da mensagem também temos que fazer o tratamento dos tópicos pelos quais as mensagens chegam. Como estamos tratando de um atuador simples que irá controlar um ponto de iluminação, precisamos tratar apenas as mensagens que mudam o estado da iluminação para ligado ou desligado no tópico de comandos, como é possível observar no código abaixo.

Além de ser compatível com os modelos de iluminação atual, a ideia é tornar os atuadores invisíveis ao usuário tanto na usabilidade quanto fisicamente, e isso implica em duas possibilidades de instalação, são elas: descentralizada, com maior dificuldade para alimentar os atuadores e manter todos eles conectados ao Wi-Fi e ao *broker*, porém com uma necessidade menor em passar cabos, e centralizada, onde é preciso passar mais cabos para conectar a iluminação e os interruptores aos atuadores mas com a vantagem de ser muito mais fácil de manter os dispositivos conectados, sendo a segunda maneira a mais recomendada. Ambas as possibilidade necessitam de uma alternativa de controle do atuador de maneira alternativa a física, uma vez em que o atuador pode estar instalado em locais pouco acessíveis, e parar realizar essas interações utilizamos o tópico `hostname/system/set`. Como é possível observar no código, no tópico `system/set` possuímos as funcionalidades de reiniciar o atuador, ativar a o modo AP com interface *web* para trocar de rede e ativar e desativar a funcionalidade OTA (Over-The-Air), funcionalidade que será explicada em uma sessão subsequente.

```
1  void setup() {
2  ...
3    client.setCallback(callback);
4  ...
5  }
6  void callback(char *topic, byte *payload, unsigned int length) {
7    String payloadStr = "";
8    for (int i = 0; i < length; i++) {
9      payloadStr += (char)payload[i];
10   }
11   String topicStr = String(topic);
12
13
14   if (topicStr.equals(SYSTEM_TOPIC)) {
15     if (payloadStr.equals("OTA_ON_" + String(HOSTNAME))) {
16       client.publish(LOG_TOPIC, "OTA ON");
17       OTA = true;
```

```
18        OTABegin = true;
19      } else if (payloadStr.equals("OTA_OFF_" + String(HOSTNAME))) {
20        client.publish(LOG_TOPIC, "OTA OFF");
21        OTA = false;
22        OTABegin = false;
23      } else if (payloadStr.equals("REBOOT_" + String(HOSTNAME))) {
24        client.publish(LOG_TOPIC, "REBOOT");
25        ESP.restart();
26      }
27      else if (payloadStr.equals("NETWORK_" + String(HOSTNAME))) {
28        client.publish(LOG_TOPIC, "NETWORK");
29        wifiManager.setConfigPortalTimeout(180);
30        wifiManager.startConfigPortal(HOSTNAME.c_str());
31      }
32
33    }
34
35
36    else if (topicStr.equals(COMMAND_TOPIC)) {
37       if(payloadStr.equals("ON")){
38           client.publish(LOG_TOPIC,"ON");
39           digitalWrite(SwitchedPin, LOW);
40        }
41       else if(payloadStr.equals("OFF")){
42           client.publish(LOG_TOPIC, "OFF");
43           digitalWrite(SwitchedPin, HIGH);
44        }
45    }
46  }
```

3.2.4 OTA

Uma vez em que temos o nosso sistema pronto para uso, a ideia é que todas as atualizações sejam feitas pelo Home Assistant, porém é inevitável que em algum momento teremos que fazer alguma alteração nos atuadores, NodeMCU. Como discutido anteriormente, possivelmente após a instalação dos atuadores eles não irão ficar em locais facilmente acessíveis. Para solucionar esse problema iremos utilizar a biblioteca ArduinoOTA, que permite que seja feita a compilação de um novo código fonte nos atuadores sem a necessidade de se conectar fisicamente aos atuadores.

Como mostrado na sessão anterior, no atuador existe um tópico que ativa e desativa as funcionalidades do sistema dele. Uma das funcionalidade dos atuadores é o OTA. Para configurar o OTA no atuador precisamos inserir o seguinte código nos atuadores.

```
1  #include <ArduinoOTA.h>
2
```

```
3  bool OTA = false;
4  bool OTABegin = false;
5
6  void loop(){
7      client.loop();
8      if (OTA) {
9        if (OTABegin) {
10           setupOTA();
11           OTABegin = false;
12       }
13       ArduinoOTA.handle();
14     }
15 }
16
17 void setupOTA() {
18   if (WiFi.status() == WL_CONNECTED && checkMqttConnection()) {
19   //  client.publish(LOG_TOPIC, "OTA SETUP");
20   //  ArduinoOTA.setHostname(HOSTNAME.c_str());
21   //  ArduinoOTA.setPassword(OTA_PASSWORD);
22
23     ArduinoOTA.onStart([]() {
24     Serial.println("Start OTA");
25     });
26
27     ArduinoOTA.onEnd([]() {
28       Serial.println("End OTA");
29     });
30
31     ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
32
33     });
34
35     ArduinoOTA.onError([](ota_error_t error) {
36
37       if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
38       else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
39       else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
40       else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
41       else if (error == OTA_END_ERROR) Serial.println("End Failed");
42     });
43
44     ArduinoOTA.begin();
45
46     Serial.println("Ready");
47     Serial.print("IP address:");
48     Serial.println(WiFi.localIP());
49   }
50 }
```

3.3 Home Assistant

Com o atuador NodeMCU, pronto para se integrar a rede da automação, precisamos configurar o Home Assistant e o *broker* MQTT que será executado nele. Como mencionado anteriormente, o Home Assistant pode ser instalado em qualquer máquina que possua python instalado. Aqui, por motivos de disponibilidade e praticidade, iremos utilizar um Raspberry Pi model B. Hass.io é uma imagem "bootável"do Home Assistant que permite que ele seja instalado de maneira simplificada dentro de um SBC. Fazendo o download da imagem do Hass.io no site oficial do Home Assistant, instalando-a em um cartão de memória de no mínimo 8gb e inserindo-o no Raspberry estamos prontos para configurar o Home Assistant.(ASSISTANT, 2019g)

Depois de instalado, o Home Assistant disponibiliza uma interface *web* de acesso através do endereço localhost:8123. No primeiro acesso ao Home Assistant é criado um usuário *owner* que possui permissão para alterar todos os parâmetros do sistema. Infelizmente, na versão atual do Home Assistant todos os usuários criados possuem as mesmas permissões do usuário *owner*, o que concede uma visão global do sistema para todos os usuários, diferente do que é ideal para o mundo real onde cada usuário tem acesso apenas a um determinado contexto do sistema.(ASSISTANT, 2019c)

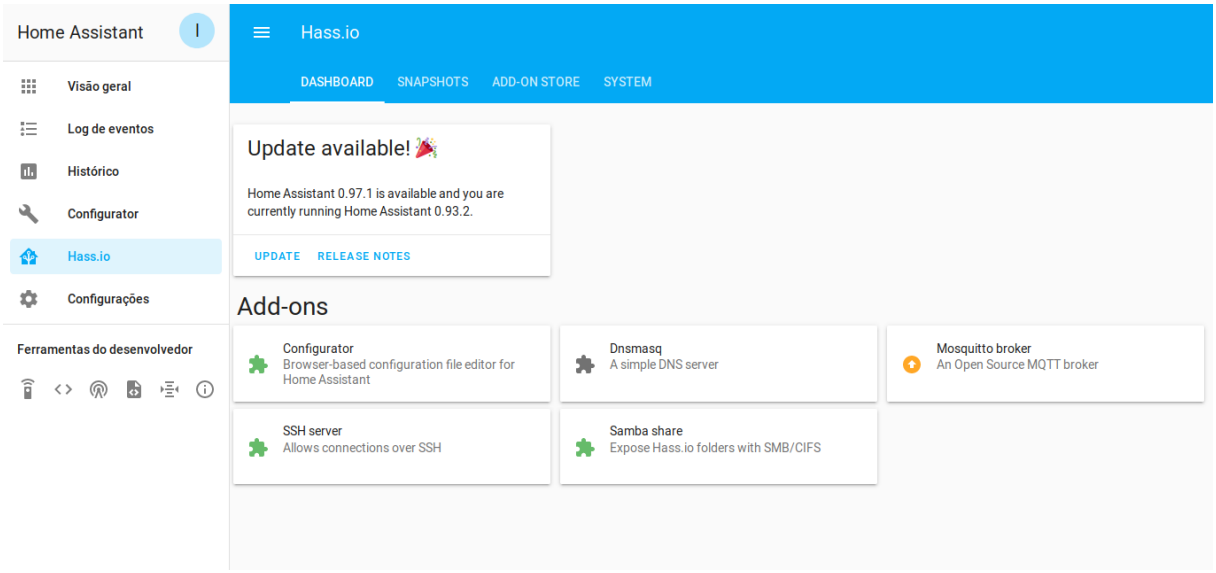
Depois do login autenticado, temos acesso aos controles do Home Assistant. Como o Home Assistant é um sistema modular, escalável de acordo com a nossa necessidade, após o sistema instalado possuímos poucas funcionalidades por padrão e precisamos ir até a loja virtual de aplicativos para incrementar o sistema com módulos do nosso interesse. Como é possível observar na Figura 11, para a solução proposta precisamos de três *add-ons* básicos, são eles: SSH Server, responsável por estabelecer uma conexão segura via SSH entre um cliente e o Home Assistant, Samba share, servidor de arquivos que, utilizando o protocolo SMB (Server Message Block), permite a manipulação de arquivos do Home Assistant de forma simplificada, e o Mosquitto *broker*, responsável pelo gerenciamento dos tópicos e o controle de atividades *publish-subscribe*, ou seja, o registro de subscrições e o encaminhamento de notificações aos subscritores.

Como os servidores SSH e SAMBA não possuem conexão direta com os atuadores, não entraremos em detalhes quanto as suas configurações que além de muito simples não acrescentam informações relevantes ao trabalho. O Mosquitto, por outro lado, possui uma alta conexão com os atuadores e precisa de uma configuração específica para funcionar de acordo com o esperado. Como é possível ver nos dados de configuração abaixo, criamos no Home Assistant um usuário com os mesmos parâmetros que foram utilizados no código fonte dos atuadores para que seja possível fazer uma conexão autenticada. É importante observar que o broker MQTT também

trabalha com conexões anônimas, ou seja, no caso em que o usuário não consiga se autenticar ou não possui credenciais de autenticação, ele é conectado ao servidor de forma anônima e não pode realizar nenhuma ação além de subscrever a tópicos já existentes. Por não conseguirmos mensurar o quão grave pode ser um dispositivo conectado a rede e que visualize as mensagens dos tópicos irrestritamente, iremos trabalhar apenas com conexões autenticadas.

```
{
  "logins": [
    {
      "username": "mqttuser",
      "password": "4ccd5a"
    }
  ]
...
}
```

Figura 11 – Dashboard do HomeAssistant



Uma vez devidamente autenticado e conectado ao *broker* Mosquitto, conseguimos conversar com os atuadores do sistema e passar comandos para serem interpretados por eles. Como discutido anteriormente, não é de grande valia o simples fato de conseguir controlar objetos pela interface do Home Assistant mas sim fazê-los interagir entre si e responder a estímulos de sensores externos. Além dos própios atuadores NodeMCU, que são sensores físicos, podemos adicionar sensores lógicos ao nosso sistema, por exemplo: sensor de data e hora e sensor de conexão ao Wi-Fi, que pode ser usado como sensor de presença.

Todas as configurações do Home Assistant são feitas através de arquivos de configuração escritos na linguagem YAML, feita para facilitar o entendimento da serialização dos dados pelos

humanos. Para acessar os arquivos de configuração do Home Assistant precisamos nos conectar a ele através do protocolo smb utilizando o gerenciador de arquivos. Após a conexão, como é possível obeservar na Figura 12, encontramos os arquivos de configuração subdivididos nas seguintes categorias:

- automation.yaml, onde é descrito todas as automações do sistema;
- configuration.yaml, onde é descrito todas as configurações de sensores e atuadores além de configurações gerais do sistema;
- customize.yaml, onde são descritas as customizações de entidades, por exemplo, um ícone customizado;
- groups.yaml, onde descrevemos como deverão ser agrupadas as entidades;
- homeassistant.log, onde é salvo o log do sistema;
- home-assistant_v2.db, banco de dados onde é salvo o histórico dos estados das entidades do sistema;
- known_devices.yaml, onde é armazenado as informações dos sensores e atuadores do sistema;
- scripts.yaml, é onde descrevemos uma sequência de ações que o home assistant deve executar ao ser ligado e
- secrets.yaml, onde podem ser armazenadas as senhas do sistema.

Como estamos criando um dispositivo genérico, ele não é descoberto automaticamente pelo Home Assistant. Para registrar o dispositivo no Home Assistant precisamos adicionar o código a seguir dentro do arquivo configurations.yaml. Já existe no Home Assistant a entidade *light* que nos possibilita apenas apenas parametrizá-la para criar uma nova entidade de iluminação. Será utilizado um QoS com valor 1 que, como mostrado anteriormente, garante que a mensagem seja sempre entregue ao atuador e caso não haja uma confirmação de recebimento outra mensagem é enviada. Os parâmetros de *payload* geraram serviços no Home Assistant que são chamados pelas automações para realizar ações no sistema. Com os parâmetros de *payload_on* e *payload_off* são criados os serviços de *light.turn_on* e *light.turn_off*, respectivamente.

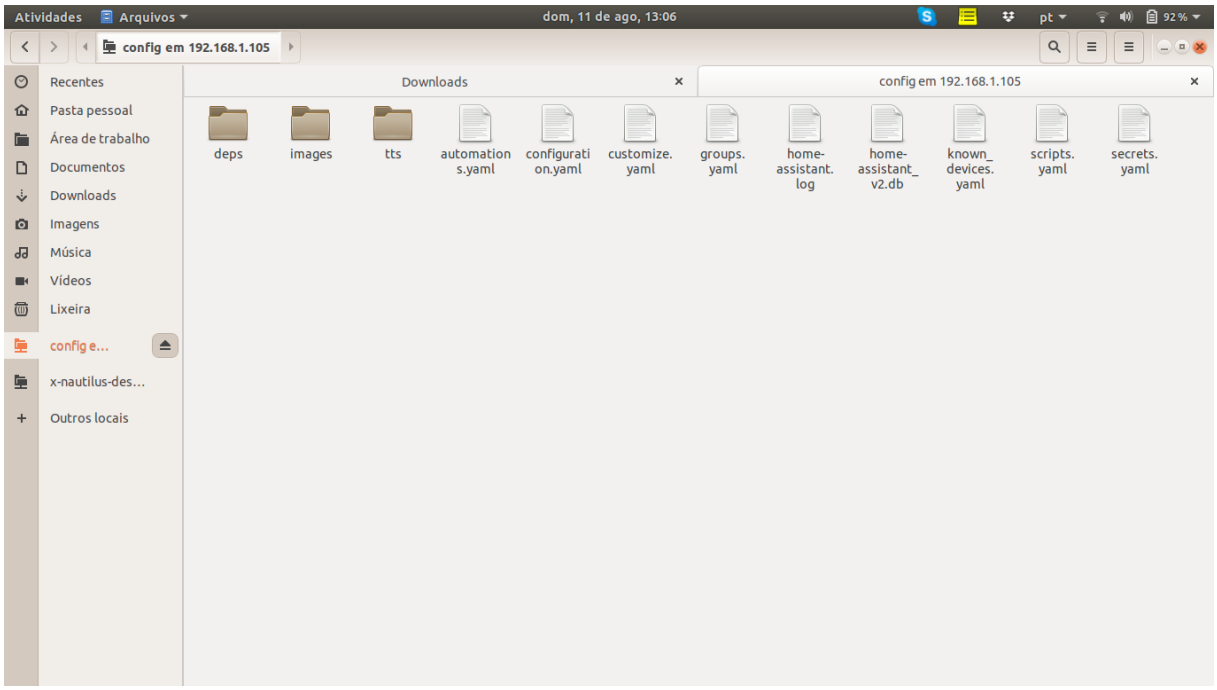
```
light:
  - platform: mqtt
    name: "Teste"
    command_topic: "TESTS/set"
    state_topic: "system/log"
    payload_on: "ON"
    payload_off: "OFF"
    qos: 1
    retain: true
```

Dois sensores lógicos foram utilizados para configurar as automações no sistema de exemplo, são eles: worldclock, responsável simplesmente por mostrar a hora na interface do Home Assistant(ASSISTANT, 2019l) e o TP-Link Smart Home, que permite a detecção de presença através da verificação de conexões com o roteador.(ASSISTANT, 2019j) Segundo a documentação oficial do Home Assistant e como é possível observar no código abaixo, basta adicionar essas informações ao nosso arquivo de configuração, configurations.yaml, e já conseguimos utilizar os sensores. Para o sensor de hora basta definirmos a zona em que nos encontramos para que o sistema registre a hora correta, enquanto que para o sensor do roteador precisamos informar e endereço e os dados de autenticação.

```
sensor:
  - platform: worldclock
    time_zone: America/Araguaina

device_tracker:
  - platform: tplink
    host: 192.168.1.1
    username: admin
    password: leonardo
```

Figura 12 – Gerenciador de arquivos conectado ao HomeAssistant



Com os sensores físicos e lógicos devidamente conectados, é possível gerar automações baseadas nesses sensores. Uma automação é composta por 4 componentes básicos, são eles:

- nome, utilizado para definir e identificar uma automação de forma simples;
- gatilho, o que inicia o processamento das regras de automação;
- condições, uma parte opcional de uma regra de automação e podem ser usadas para impedir que uma ação aconteça quando acionada em uma determinada condição e
- ações, tarefas que o Home Assistant deve executar uma vez em que a automação tenha sido acionada.

Para melhor entendimento da diferença entre gatilhos e condições, podemos dizer que os gatilhos verificam a alteração de um estado específico para outro de uma determinada entidade enquanto que as condições verificam o estado atual de uma entidade.

Como é possível observar nas figuras 13 e 14, o Home Assistant fornece um gerador automático de automações onde basta passarmos os parâmetros da automação e ele gera automaticamente o código referente a aquela automação no arquivo automations.yaml. Para exemplificar, iremos criar uma automação onde quando for detectado que o usuário especificado saiu de casa, o sistema irá automaticamente apagar a luz. Como gatilho da automação irmos utilizar a mudança de estado da entidade rastreada de "home"para "not_home". Como não iremos utilizar nenhuma condição para essa automação, caso o gatilho seja acionado o Home Assistant irá chamar o

serviço de "light.turn_off"definido anteriormente. Assim que a automação for salva, o seguinte código será gerado no arquivo automations.yaml:

```
- id: '1559694382069'
  alias: 'Apagar luz ao sair '
  trigger:
  - entity_id: device_tracker.f0d7aae8947c
    from: home
    platform: state
    to: not_home
  condition: []
  action:
  - alias: ''
    data:
      entity_id: light.quarto_leo
    service: light.turn_off
```

Figura 13 – Gerador de Automações do HomeAssistant

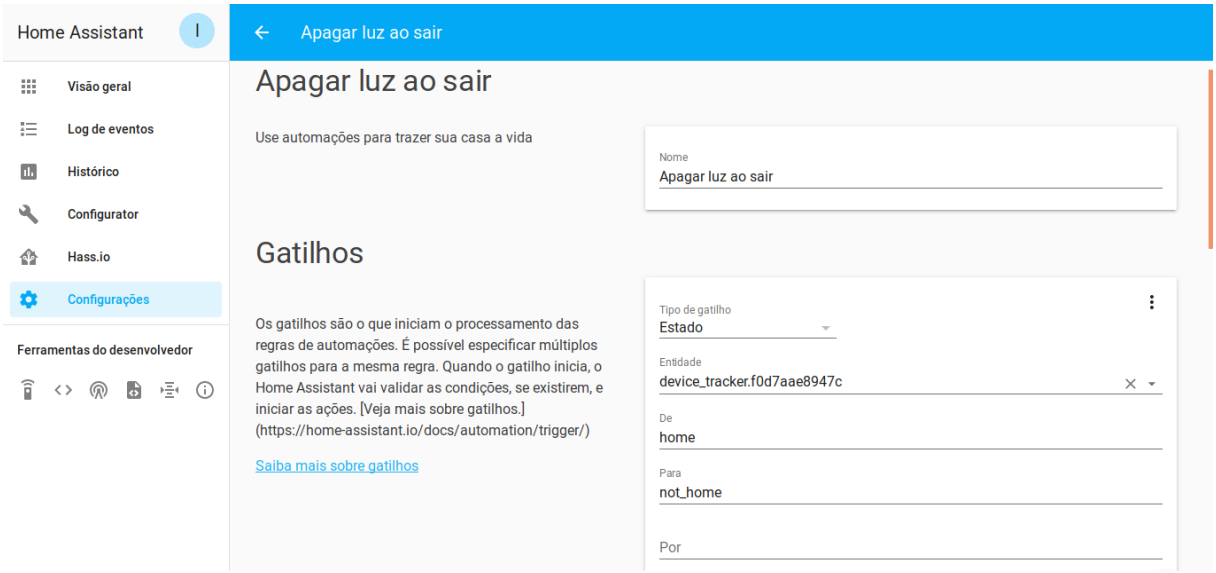
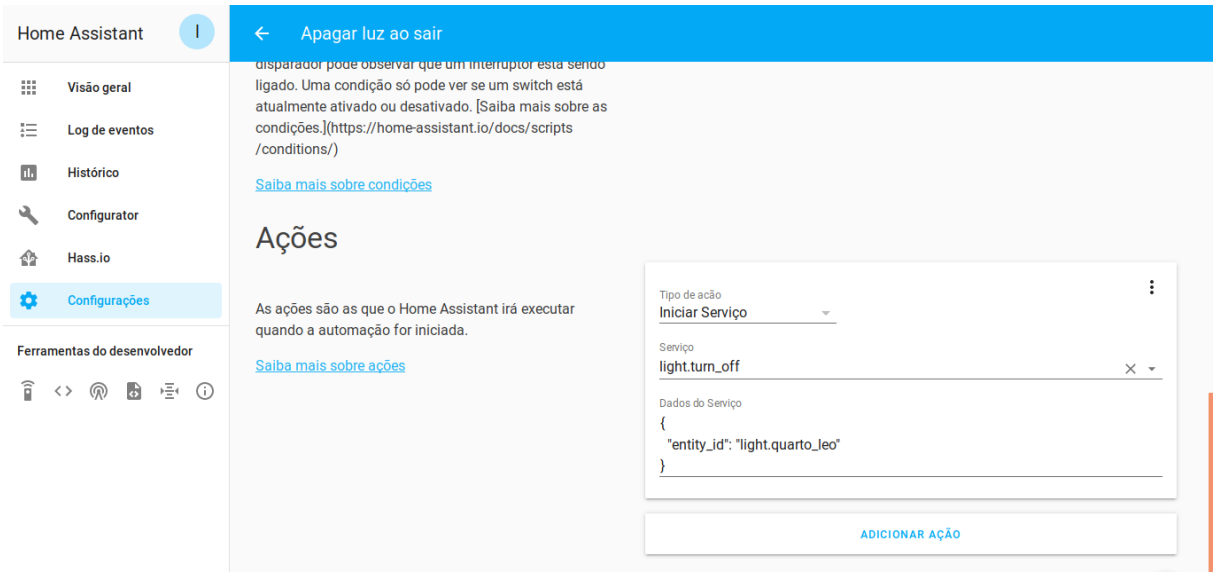


Figura 14 – Gerador de automações do HomeAssistant



Depois de registrar as entidades e configurar as automações no Home Assistant, o sistema está pronto para uso. Para definir como as entidades irão ser agrupadas e exibidas na interface precisamos alterar o arquivo groups.yaml. Assim como para as automações, o Home Assistant também possui um facilitador para gerar a interface final para o usuário. Lovelace UI é um editor de interface que permite alterar e agrupar as entidades do Home Assistant de forma bastante intuitiva.(ASSISTANT, 2019h) Atualmente podemos ver a tela inicial do sitema por duas rotas .../lovelace ou .../states. Na rota *states* é exposta a interface configurada no arquivo groups.yaml enquanto que na rota *lovelace* é exposta a interface configurada pelo Lovelace UI. Para configurar a rota *states* para ficar igual a interface montada com a Lovelace UI, mostrado na Figura 15, inserimos o seguinte código dentro do arquivo groups.yaml. Por fim, como mostrado na Figura 16, temos a interface organizada e apresentada de forma organizada e intuitiva.

```
1  default_view:
2    view: true
3    icon: mdi:home
4    entities:
5      - light.Teste
6      - light.quarto_leo
7      - automation.ligar_a_luz_pela_manha
8      - automation.apagar_a_luz_pela_manha
9      - automation.apagar_luz_ao_sair
10     - sensor.worldclock_sensor
11     - device_tracker.f0d7aae8947c
```

Figura 15 – Lovelace UI do HomeAssistant

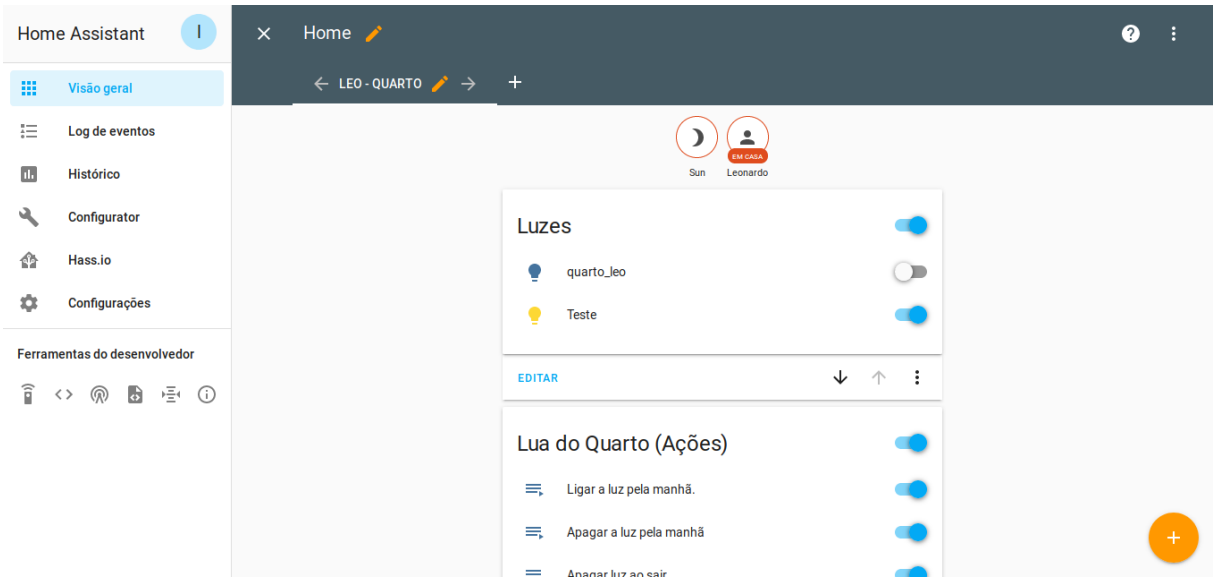
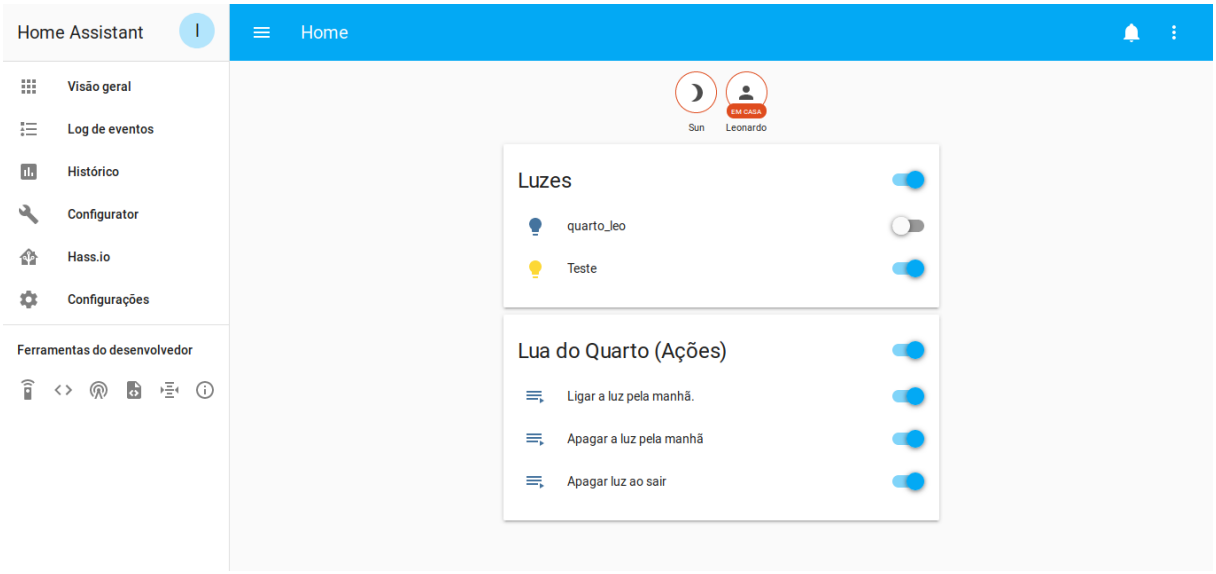


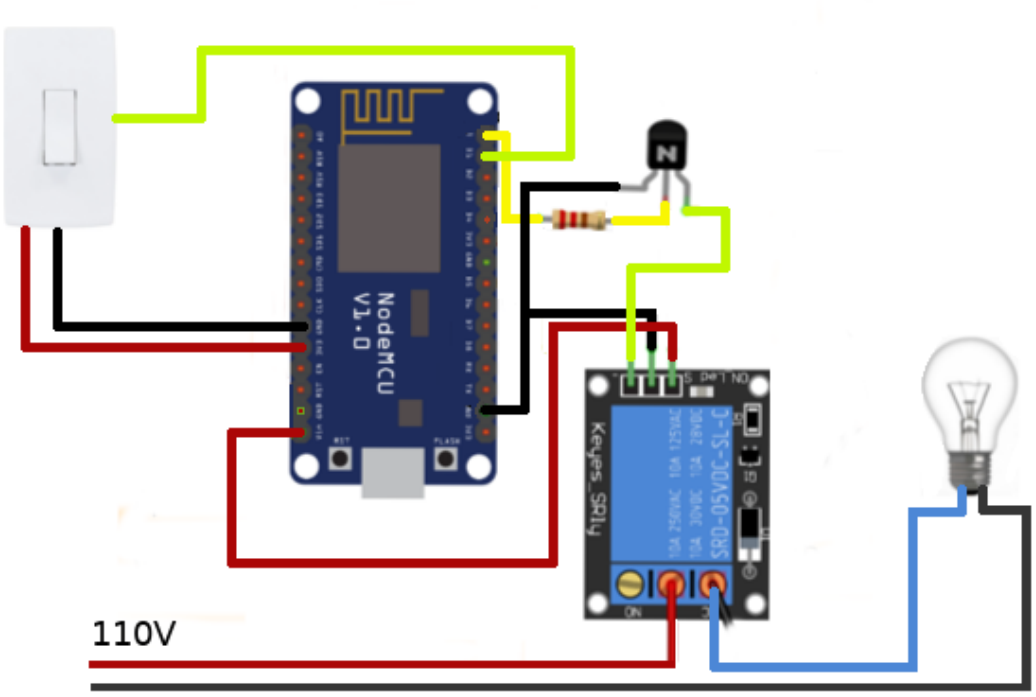
Figura 16 – Página Web HomeAssistant



3.4 Testes e Validações

Tentando ir além da prototipação e da teoria, o circuito mostrado na Figura 17 foi construído em uma placa perfurada de 7x5cm com o intuito facilitar as conexões entre o NodeMCU e os periféricos e posto em produção, controlando a iluminação de um ambiente residencial real. Testes funcionais foram desenvolvidos para simular um ambiente de infraestrutura de má qualidade e com muitas falhas.

Figura 17 – Tela de login do Home Assistant.



Antes de ser posto em produção, os dispositivos foram conectados a um roteador que se desligava sozinho de maneira aleatória, simulando quedas de energia. De forma semelhante, o Raspberry Pi, onde o Home Assistant foi instalado, também foi desconectado da rede diversas vezes de forma aleatória, simulando problemas de instabilidade no *broker* MQTT. Após testar cada versão do software em um ambiente caótico o dispositivo foi colocado em um cenário real controlando a iluminação de um quarto. A utilização do dispositivo em sua última versão teve a duração de aproximadamente dois meses e as seguintes automações foram definidas:

- Ligar a luz todos os dias às 7h da manhã;
- Desligar a luz todos os dias às 7:15h da manhã e
- Desligar a luz a caso o status de rastreamento mude de *home* para *not_home*.

Na situação prática real o dispositivo foi colocado a uma distância de aproximadamente 10m de distância do roteador e em um andar inferior. Durante todo o processo houveram poucas quedas e de energia mas a internet caiu e o roteador precisou ser reiniciado várias vezes. Considerando as condições mencionadas, nenhuma falha de funcionamento foi detectada durante a utilização da versão final do dispositivo, nos parecendo ser uma alternativa confiável para se construir automações robustas.

4
Conclusões

Por se tratar de um tema relativamente novo e muito abordado no mundo DIY (*Do It Yourself*), vários tutoriais de como utilizar o NodeMCU e MQTT podem ser facilmente encontrados na internet. Apesar da abundância de trabalhos utilizando essas tecnologias, de forma geral, eles propõem apenas prototipação e acabam não tendo preocupações com o uso em aplicações reais. Por esse motivo, poucos projetos acabam se preocupando com a integração com os modelos atuais ou com a resistência a falhas, por exemplo.

Apesar de poucos trabalhos se aprofundarem em soluções robustas para uma plataforma utilizando NodeMCU, MQTT e Home Assistant, devemos destacar o projeto ESP Home (HOME, 2019), o qual é uma das soluções mais completas utilizando as tecnologias mencionadas. Criado em Abril de 2018, o ESP Home é um sistema que se propõe a controlar dispositivos ESP8266/ESP32 através de um sistema de automação, dentre eles o Home Assistant, de maneira simples e robusta. Apesar de termos começado os estudos e pesquisas relacionadas a esse projeto antes do ESP Home ser lançado, pode-se observar o ESP Home com o intuito de vislumbrar, mesmo que parcialmente, o que temos como objetivo final nesse projeto.

Mesmo que o dispositivo desenvolvido tenha sido testado sob condições de falhas e mesmo que tenha apresentando resultados estáveis e se mostrando confiável para construir automações robustas, melhorias ainda podem ser feitas. Assim como no ESP Home, que já possui uma componente para o Home Assistant, uma extensão deste projeto poderia implementar um componente dedicado para o Home Assistant e incrementar as funcionalidades do NodeMCU, permitindo que ele controlasse outros dispositivos além de lâmpadas, tais como: televisores e aparelhos de ar-condicionado. Para construir uma plataforma nova no Home Assistant se utiliza a linguagem python seguindo as diretrizes de melhores práticas disponíveis nos guias da plataforma (ASSISTANT, 2019f).

Quando estamos trabalhando com apenas um NodeMCU a quantidade de energia consumida pela a placa não tem tanta relevância, mas a medida em que os projetos crescem, mais

placas são necessárias para conseguir realizar a automação e o consumo de energia das diversas placas e periféricos podem ter um impacto significativo no consumo de energia ao longo do tempo. Tendo em vista que o NodeMCU é um dispositivo genérico e seu uso pode se estender para projetos não residenciais que obriguem o dispositivo a ser alimentado por baterias, o chip do NodeMCU, ESP8266, possui alguns modos de operação com economia de energia onde cada um deles desativam certas funcionalidades da placa e as reativam depois de um período de tempo ou estimulação externa. Esses modos de operação são chamados, genericamente, de modo *sleep*. Uma melhoria significativa nos dispositivos seria colocá-los para funcionar em modo *sleep* que além da trazer economia de energia possibilitaria o uso de baterias ao invés de uma alimentação cabeada.

Apesar de ser uma ferramenta muito completa, algumas funcionalidades consideradas fundamentais para um sistema de automação ideal ainda estão ausentes na versão oficial do Home Assistant. Tendo em vista que ambientes automatizados devem apresentar visões diferentes para cada usuário, falta ao Home Assistant um sistema de permissões a entidades fazendo com que seja possível limitar o acesso de um determinado usuário a determinadas entidades.

Mesmo não sendo recomendada pela documentação oficial do Home Assistant, existe a funcionalidade experimental de criar grupos e atrelar usuários criados pelo *owner* a esses grupos. No entanto, como não é possível criar grupos através da interface gráfica do Home Assistant, esta funcionalidade requer a alteração dos arquivos fontes do Home Assistant (ASSISTANT, 2019i). Como um trabalho futuro, grupos de permissões poderiam ser criados e a página web do Home Assistant apresentaria a cada usuário tão somente os dispositivos para os quais ele tem permissão.

Referências

ARDUINO. *ESP8266 Arduino Core*. [S.l.], 2019. Disponível em: <https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html>. Citado na página 27.

ARDUINO. *Millis()*. [S.l.], 2019. Disponível em: <https://www.arduino.cc/reference/pt/language/functions/time/millis/>. Citado na página 28.

ASSISTANT, H. *Arquitetura do Hass.io*. [S.l.], 2019. Disponível em: <https://developers.home-assistant.io/docs/en/architecture_hassio.html>. Citado na página 18.

ASSISTANT, H. *Arquitetura do Home Assistant*. [S.l.], 2019. Disponível em: <https://developers.home-assistant.io/docs/en/architecture_index.html>. Citado na página 19.

ASSISTANT, H. *Authentication Hass.io*. [S.l.], 2019. Disponível em: <https://www.home-assistant.io/docs/authentication/>. Citado na página 34.

ASSISTANT, H. *Automation Editor*. [S.l.], 2019. Disponível em: <https://www.home-assistant.io/docs/automation/editor/>. Citado na página 21.

ASSISTANT, H. *Hass.io*. [S.l.], 2019. Disponível em: <https://www.home-assistant.io/hassio/>. Citado na página 17.

ASSISTANT, H. *Home Assistant Plataform Integration*. [S.l.], 2019. Disponível em: <https://developers.home-assistant.io/docs/en/creating_platform_index.html>. Citado na página 43.

ASSISTANT, H. *Installing Hass.io*. [S.l.], 2019. Disponível em: <https://www.home-assistant.io/hassio/installation/>. Citado na página 34.

ASSISTANT, H. *Lovelace UI*. [S.l.], 2019. Disponível em: <https://www.home-assistant.io/lovelace/>. Citado na página 40.

ASSISTANT, H. *Permission Hass.io*. [S.l.], 2019. Disponível em: <https://developers.home-assistant.io/docs/en/next/auth_permissions.html>. Citado na página 44.

ASSISTANT, H. *TP-Link Smart Home Devices*. [S.l.], 2019. Disponível em: <https://www.home-assistant.io/components/tplink/#presence-detection>. Citado na página 37.

ASSISTANT, H. *[Update: new users only] Nest to turn off their API*. [S.l.], 2019. Disponível em: <https://www.home-assistant.io/blog/2019/05/08/nest-data-bye-bye/>. Citado na página 9.

ASSISTANT, H. *Worldclock Home Assistant*. [S.l.], 2019. Disponível em: <https://www.home-assistant.io/components/worldclock/>. Citado na página 37.

CLOUDMQTT. *Hosted message broker for the Internet of Things*. [S.l.], 2019. Disponível em: <https://www.cloudmqtt.com/>. Citado na página 16.

ESPERSSIF. *ESP8266EX*. [S.l.], 2019. Disponível em: <https://www.espressif.com/en/products/hardware/esp8266ex/overview>. Citado 2 vezes nas páginas 8 e 11.

HOME, E. *ESP Homes*. [S.l.], 2019. Disponível em: <https://esphome.io/index.html>. Citado na página 43.

IBM. *Qualidades de serviço fornecidas por um cliente MQTT*. [S.l.], 2019. Disponível em: <https://www.ibm.com/support/knowledgecenter/pt-br/SSFKSJ_8.0.0/com.ibm.mq.dev.doc/q029090_.htm>. Citado na página 16.

KNOLLEARY. *Arduino Client for MQTT*. [S.l.], 2019. Disponível em: <https://github.com/knolleary/pubsubclient>. Citado na página 28.

MIT. *Licença Mit*. [S.l.], 2019. Disponível em: <https://www.mit.edu/~amini/LICENSE.md>. Citado na página 13.

MQTT. *FAQ - Frequently Asked Questions*. [S.l.], 2019. Disponível em: <http://mqtt.org/faq>. Citado na página 15.

NODEMCU. *NodeMCU connect things easy.* [S.l.], 2019. Disponível em: <https://www.nodemcu.com/index_en.html>. Citado na página 12.

OGLIARI, R. *Popularização do padrão Publisher/Subscriber no mobile e na IoT*. [S.l.], 2017. Disponível em: <https://imasters.com.br/desenvolvimento/popularizacao-do-padrao-publishersubscriber-no-mobile-e-na-iot>. Citado na página 15.

ORG mqtt. *Mqtt*. [S.l.], 2019. Disponível em: <http://mqtt.org/>. Citado na página 15.

PUBSUBCLIENT. *Arduino Client for MQTT*. [S.l.], 2019. Disponível em: <https://github.com/knolleary/pubsubclient>. Citado na página 17.

SCHOUTSEN, P. *Perfect Home Automation*. [S.l.], 2016. Disponível em: <https://www.home-assistant.io/blog/2016/01/19/perfect-home-automation/>. Citado 2 vezes nas páginas 7 e 8.

TZAPU. *Wi-FiManager*. [S.l.], 2019. Disponível em: <https://github.com/tzapu/WiFiManager>. Citado na página 26.

WEISER, M. *The computer for the 21st century*. [S.l.], 1991. Disponível em: <https://pt.wikipedia.org/wiki/Computaç~ao_Ubíqua>. Citado na página 8.