UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

ANDERSON IMPERATORI

PROPOSTA DE SOLUÇÃO E IMPLEMENTAÇÃO PARA MONITORAMENTO DE AMEAÇAS EM DOMICÍLIOS

BENTO GONÇALVES

2022

ANDERSON IMPERATORI

PROPOSTA DE SOLUÇÃO E IMPLEMENTAÇÃO PARA MONITORAMENTO DE AMEAÇAS EM DOMICÍLIOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Dr. Ricardo Vargas Dorneles

BENTO GONÇALVES

2022

ANDERSON IMPERATORI

PROPOSTA DE SOLUÇÃO E IMPLEMENTAÇÃO PARA MONITORAMENTO DE AMEAÇAS EM DOMICÍLIOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado em 28/06/2022

BANCA EXAMINADORA

Prof. Dr. Ricardo Vargas Dorneles
Universidade de Caxias do Sul - UCS

Prof. Dr. André Luis Martinotto
Universidade de Caxias do Sul - UCS

Prof. Me. Alexandre Erasmo Krohn Nascimento
Universidade de Caxias do Sul - UCS

AGRADECIMENTOS

Agradeço à minha esposa Tamires Ribeiro Piccoli, por ter me ajudado e guiado todos os dias nesta etapa tão importante da nossa vida. Agradeço também aos meus pais Valmir Imperatori e Maricelsi Nicaretta Imperatori que não mediram esforços para me ajudar a concluir essa etapa da vida.

Aos meus amigos e colegas, que de alguma forma me ajudaram e incentivaram nesta etapa. Especialmente ao amigo e colega Leonardo Bertele Tosin por todos os conhecimentos compartilhados.

Agradeço ao Professor Ricardo Vargas Dorneles, responsável pela orientação deste trabalho. Sou grato aos Professores André Luis Martinotto e Alexandre Erasmo Krohn Nascimento, que apoiaram em cada etapa da pesquisa e contribuíram com as revisões do conteúdo.

Também sou grato a todos os professores que compartilharam seus conhecimentos ao longo de toda a minha vida acadêmica.

“Nem tão longe, impossível
Nem tampouco lá... já”
Humberto Gessinger

RESUMO

A residência desempenha papel fundamental para o bem-estar dos moradores, além de representar um elemento essencial para a segurança dos residentes, assim como seus bens e objetos pessoais. Dessa forma, é imprescindível que o ambiente residencial esteja protegido contra possíveis ameaças, como por exemplo uma tentativa de furto ou até mesmo uma fuga de gás. Em contrapartida mecanismos disponíveis no mercado para prover segurança aos lares possuem um custo elevado. Considerando os avanços da tecnologia, sobretudo da *internet* das coisas, é possível ter acesso a diversos dispositivos e conectá-los entre si, tais como câmeras e sensores. Dessa forma, o presente trabalho propõe o desenvolvimento de um sistema de monitoramento de ameaças em residências diversas, sendo elas casas, apartamentos, entre outros modelos de lares, utilizando sensores e câmeras para identificar e monitorar possíveis ameaças. Através da coleta de dados em tempo real, os moradores serão notificados onde quer que estejam por meio da *internet*, para efetuar as contramedidas necessárias. Para isso, foi utilizada a solução *Home Assistant*, que é uma plataforma *open source* usada para automação residencial. A partir da instalação de sensores e câmeras na residência, os dados são coletados e enviados ao *Home Assistant*, que por sua vez processa e disponibiliza-os para os moradores. Ao processar os dados recebidos a plataforma envia notificações aos *smartphones* dos usuários de acordo com regras preestabelecidas. Além disso, o presente trabalho também conta com um modo de contingência que possui a função de garantir que os dados sejam acessados pelos moradores, mesmo que a residência não tenha à sua disposição energia elétrica ou acesso à *internet*.

Palavras-chave: *Home Assistant*. Segurança. Residência. MQTT.

ABSTRACT

The residence has a fundamental role for the well-being of residents, further that it represents an essencial element for residents, as well as goods and personal objects. It is essential that the residential environment is protected against threats, such as attemped theft or even a gas leak. On the other hand, market devices that provide security in homes have a high cost. Considering the technological advances, especially the internet of things, it is possible to have access to different devices and connect them to each other, such as cameras and sensors. So the present paper propose to develop a threat monitoring system using sensors and cameras to identify and monitor possible threats. Through real-time data collection, residents will be notified wherever they are via internet, to execute countermeasures. For this, the Home Assistant solution was used, which is an open source platform used for home automation. From the installation of sensors and cameras in the residence, data is collected and sent to Home Assistant, that processes it and make it available to the residents. When processing received data, the platform send notifications to smartphones users according to pre-established rules. In addition, this paper also has a contingency mode that has the function of ensuring that data can be accessed by residents, even if the residence does not have electricity or internet access.

Keywords: Home Assistant. Security. Residence. MQTT.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

ABESE Associação Brasileira das Empresas de Sistemas Eletrônicos de Segurança

API *Application Programming Interface*

ARM *Advanced RISC Machine*

AWS Amazon Web Services

CPU *Central Processing Unit*

DDNS *Dynamic* DNS

FPS *Frames Per Second*

GB *Giga Byte*

GHz *Gigahertz*

GLP Gás Liquefeito de Petróleo

GNV Gás Natural Veicular

GPIO *General Purpose Input/Output*

HAOS *Home Assistant Operating System*

HTTP *Hypertext Transfer Protocol*

IaaS *Infrastructure as a Service*

IDE *Integrated Development Environment*

IDLH *Immediately Dangerous to Life or Health*

IoT *Internet of Things*

IP *Internet Protocol*

JPEG *Joint Photographics Experts Group*

LAN *Local Area Network*

Mbps Megabits por Segundo

MHz *Megahertz*

MJPEG *Motion* JPEG

MQTT *Message Queuing Telemetry Transport*

MVVM *Model View ViewModel*

ORM *Object Relational Mapper*

PaaS *Platform as a Service*

PIR *Passive Infrared Sensor*

PPM Partes por Milhão

QoS *Quality of Service*

RAM *Random-access Memory*

SaaS *Software as a Service*

SDK *Software Development Kit*

SSH *Secure Shell*

SSL *Secure Sockets Layer*

TCP *Transmission Control Protocol*

TLS *Transport Layer Security*

URL *Uniform Resource Locator*

USB *Universal Serial Bus*

LISTA DE SÍMBOLOS

Ω	Ohm

SUMÁRIO

1 INTRODUÇÃO

A segurança domiciliar é essencial para manter o bem-estar dos moradores e a integridade de seus pertences. Segundo Conway (2005, p. 15) a residência é um elemento crítico para a segurança e proteção de seus residentes por diversas razões, sendo alguns dos exemplos citados pelo autor a permanência dos residentes em mais de 10 horas por dia em casa; a sensação de segurança proporcionada pelo ambiente residencial faz com que as pessoas baixem suas guardas; a casa normalmente representa o maior investimento financeiro da pessoa; os indivíduos utilizam suas residências para guardar suas posses; e o período de sono, que em média são 8 horas por dia, acaba gerando uma situação de vulnerabilidade.

Dessa forma, conforme Sinopoli (2016, p. 11) as residências devem prover proteção física para seus ocupantes e seus pertences. Os sistemas de construção para fornecer segurança e proteção à vida devem ter alarmes de fogo, vídeo monitoramento, controle de acesso e detecção contra intrusos. Além disso, Conway (2009, p. 1) ressalta que muitas pessoas já foram vítimas de crimes ou conhecem pessoas que já passaram por essa situação, o que reforça a importância da segurança domiciliar.

De acordo com Conway (2005, p. 15) apesar de existirem diferenças entre uma mansão e um apartamento do 15º andar, ambos possuem elementos em comum. Todas as moradias contam com entrada, vizinhos, portas, janelas, além de serem passíveis de falhas, indiferente da edificação. Exemplo disso é apresentado no texto de Ceccon (2019) que fala sobre o monóxido de carbono, produzido na queima de combustíveis tais como o gás de cozinha, que representa um risco para os moradores. "O monóxido de carbono é um gás que pode matar em minutos e as vítimas não conseguem perceber o que as está asfixiando" (CECCON, 2019). Além disso, a fumaça de incêndios é descrita por Pinheiro (2021) como um perigo que pode levar a casos graves em que “cerca de 80% dos óbitos são por inalação de vapores e produtos químicos, principalmente monóxido de carbônico (sic) e cianeto”.

Além disso, a segurança domiciliar possui um custo elevado, principalmente para as pessoas que possuem orçamento limitado. Dessa forma, segundo a Associação Brasileira das Empresas de Sistemas Eletrônicos de Segurança (ABESE) "com o orçamento mais curto, a tecnologia se tornou a resposta para dois grandes problemas da vida condominial: segurança e orçamento" (ASSOCIAÇÃO BRASILEIRA DAS EMPRESAS DE SISTEMAS ELETRÔNICOS DE SEGURANÇA, 2021). Com o crescente avanço e popularização da *Internet of Things* (IoT) é possível conectar dispositivos tais como câmeras e sensores além de ter acesso aos dados coletados com acesso remoto. Segundo Sinopoli (2016, p. 20), mensagens e notificações podem ser usadas para situações de emergência. A capacidade de notificação de um sistema provê comunicação em tempo real sobre uma situação de emergência, portanto melhora significativamente a segurança e o bem-estar das pessoas.

1.1 QUESTÕES DE PESQUISA

Esse trabalho partiu das seguintes questões de pesquisa:

- Como a IoT pode ajudar a prevenir acidentes e incidentes domésticos?
- É possível criar um sistema de segurança domiciliar utilizando tecnologias *open source*?

1.2 OBJETIVOS

Este trabalho tem por objetivo planejar e desenvolver um sistema de segurança voltado ao monitoramento e envio de notificações para residentes de determinada moradia. Como uma alternativa para fugir das grandes corporações, serão utilizados *softwares open source*, tanto para baratear o sistema, quanto para não depender de uma corporação. O *hardware* será moldado para ser o menos invasivo possível na residência.

Para atender o objetivo geral, os objetivos específicos são:

1. Planejar a arquitetura geral do sistema;
2. Apresentar e desenvolver uma proposta de sistema de segurança domiciliar utilizando IoT;

1.3 ESTRUTURA DO TRABALHO

O Capítulo 2 é destinado ao referencial teórico em que são esclarecidos todos os conceitos necessários para o entendimento deste trabalho. No Capítulo 3 é apresentada a arquitetura do sistema. Já no Capítulo 4 é exposto o desenvolvimento da proposta de solução. Por fim, no Capítulo 5 são apresentadas as conclusões finais do trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo serão abordados os conceitos necessários para o entendimento deste trabalho dentre eles segurança domiciliar, IoT, domótica, protocolos de comunicação, MQTT, *streaming*, servidores e trabalhos relacionados.

2.1 SEGURANÇA DOMICILIAR

A segurança nas residências é afetada por diversos fatores tais como a localização e idade da construção, composição dos residentes, bem como suas condições econômicas e níveis de escolaridade (SINOPOLI, 2016, p. 11). Todavia, existem conceitos básicos comuns para todos os planos de segurança, seja para uma família em uma casa, uma fazenda ou um escritório (COBB, 2012, p. 27). Conway (2005, p. 15) reforça que existem diferenças entre uma mansão e um apartamento, porém todos os domicílios possuem características em comum como entradas, vizinhos, portas e janelas.

Um dos aspectos primordiais no planejamento da segurança é ter em mente que nenhum plano é absolutamente perfeito. Com tempo suficiente e motivação, o agressor sempre será capaz de derrotar qualquer plano de segurança (COBB, 2012, p. 13).

Para compreender melhor este cenário que engloba as nuances do ambiente residencial, Conway (2005, p. 6-9) define seis termos básicos:

a) **segurança (*security*)** é a aplicação dos métodos e procedimentos que são usados para fazer com que os estilos de vida estejam seguros contra quaisquer vulnerabilidades, ameaças ou riscos;

b) **vulnerabilidade** é o meio que a ameaça usa para alcançar e prejudicar uma pessoa;

c) **ameaça** é qualquer ocorrência que pode causar prejuízo, perda ou sofrimento. Ameaças podem ser impostas por criminosos por meio de furto, roubo e arrombamento;

d) **risco** é a dimensão da exposição às ameaças às quais se está vulnerável. Pode ser medido de duas formas: 1) o impacto da ameaça, que é medido pelo dano que a mesma pode causar; ou 2) a probabilidade de ser vítima da ameaça, que é medida pela frequência de exposição ao risco;

e) **contramedidas** são ações que podem ser realizadas para aumentar a segurança. Segundo Conway (2009, p. 6-7), ao identificar um problema, existem basicamente quatro ações que podem ser tomadas para reduzir o risco, sendo elas:

   – Ignorar e torcer para que o risco vá embora;

   – Tomar uma atitude para reduzir o risco;

– Tomar uma atitude para evitar o risco;

– Tomar uma atitude para remover o risco.

Entre essas opções, tendo em mente o estilo de vida e a circunstância de cada pessoa, algumas podem ser mais aceitáveis e sensatas e outras podem parecer irrealísticas e temerárias. A opção "remover o risco" pode ser a solução mais efetiva a longo prazo, porém ela requer tempo, dinheiro e esforço para ser alcançada (CONWAY, 2009, p. 8);

f) **segurança (*safety*)** pode ser definida também[1] como a circunstância em que todas as vulnerabilidades foram extintas ou foram reduzidas a níveis insignificantes;

g) **problemas de não segurança (*non-security problems*)** são problemas que não estão relacionados com segurança, porém são potenciais ameaças. Por exemplo, ao checar o jardim sentiu-se cheiro de gás, então ali pode haver um vazamento ou quando é visto um fio desencapado.

Existem diversas vulnerabilidades em cada residência, dessa forma este trabalho se restringe às vulnerabilidades relacionadas a detecção de gás, fumaça e invasores.

2.2 *INTERNET* DAS COISAS

O termo *Internet of Things*, ou em português *internet* das coisas, foi introduzido por Kevin Ashton, em uma apresentação em 1999 (ASHTON, 2009). Segundo Stallings (2017, p. 25), "A Internet das Coisas [...] é um termo que refere à interconexão expansiva dos dispositivos inteligentes, indo de aplicações a minúsculos sensores". Já Santos (2016, p. 8) conceitua IoT como a capacidade de comunicação de forma inteligente entre dispositivos e objetos equipados com sensores que fazem parte do nosso dia a dia.

Atualmente a IoT está presente na rotina das pessoas em vários aspectos, fazendo da tecnologia uma aliada do cotidiano. Conforme Magrani (2018, p. 24) residências automatizadas permitem que uma série de atividades sejam realizadas de forma remota, tais como abrir os portões, desligar alarmes, preparar o banho quente, colocar música ambiente e alterar a temperatura da casa.

Importante ressaltar que apesar das facilidades que a IoT proporciona, os diversos dispositivos que estão conectados irão coletar e transmitir grandes quantidades de dados do cotidiano, sendo que muitos destes dados são privados e sensíveis (MAGRANI, 2018, p. 24). Dessa forma, a utilização da IoT implica em responsabilidade sobre os dados para garantir a privacidade da informação.

[1] A palavra segurança é a tradução para ambas as expressões em inglês, *security* e *safety*. Dessa forma, ela possuirá dois significados distintos.

2.3 DOMÓTICA

O termo domótica resulta da junção da palavra *domus*, que significa casa, com a palavra robótica, que refere-se ao controle automatizado do ambiente. (STEVAN; FARINELLI, 2018, p. 2). A ideia de automatização residencial data dos anos 1980, com foco no controle remoto e automatização de pequenas tarefas, tais como controlar remotamente portões e garagens, controle de iluminação e alarmes de segurança (STEVAN; FARINELLI, 2018, p. 4).

> Com caráter multidisciplinar, ela agrega vários conceitos de outras ciências como Arquitetura, Engenharia, Ciência da Computação, Medicina, Sociologia e Psicologia, a fim de estudar todas as necessidades do usuário frente às possibilidades oferecidas pelo mundo digital e suas interações com a residência automatizada (BOLZANI, 2007, p. 18).

Ao longo do processo de automatização digital, manifesta-se um tema mais abrangente, denominado IoT. O termo "casas conectadas" surge no século XXI, mostrando que a *internet* está no cotidiano das pessoas. Surge também, a expressão "casa inteligente", tendo foco na comunicação e interação entre os dispositivos. A domótica está inserida dentro desse conjunto de conceitos, sendo que ela não se limita às automatizações dos anos 1980, trazendo inteligência e conectividade entre dispositivos (STEVAN; FARINELLI, 2018, p. 4).

2.4 PROTOCOLOS DE COMUNICAÇÃO

Existem diversas tecnologias competindo para se tornarem padrão na comunicação *wireless* nas casas inteligentes. Algumas das mais populares são: Wi-Fi[2], Wi-Fi HaLow[3], Bluetooth 5[4], Insteon[5], Thread[6], ZigBee[7] e Z-Wave[8]. Todas possuem vantagens, desvantagens e limitações (DAWOUD; DAWOUD, 2020, p. 513).

Nos últimos anos, a *Wi-Fi* (IEEE[9] 802.11) tornou-se o padrão banda larga *wireless* para *Local Area Network* (LAN) em casas e escritórios. Atualmente, é muito comum casas possuírem *Wi-Fi*, por isso faz sentido que as automações residenciais tomem vantagem dessa disponibilidade (DAWOUD; DAWOUD, 2020, p. 537).

As comunicações *Wi-Fi* viajam por ondas de rádio no espectro de 2.4 *Gigahertz* (GHz) ou 5 GHz. Dispositivos compatíveis possuem uma antena receptora e transmissora. Em teoria o alcance é longo, porém na prática, nas residências o alcance máximo é de 60 pés (cerca de 18

[2] https://wi-fi.org/discover-wi-fi
[3] https://wi-fi.org/discover-wi-fi/wi-fi-halow
[4] https://bluetooth.com/
[5] https://insteon.com/technology#ourtechnology
[6] https://threadgroup.org/BUILT-FOR-IOT/Home
[7] https://zigbeealliance.org/solution/zigbee/
[8] https://z-wavealliance.org/about_z-wave_technology/
[9] https://ieee.org/

metros). O alcance pode ser afetado por diversos fatores tais como barreiras físicas, interferência, potência do transmissor e qualidade da antena (DAWOUD; DAWOUD, 2020, p. 538).

A *Wi-Fi* possui taxa de transferência alta e velocidade entre 10 Mbps a 100 Mbps, dessa forma, é possível fazer *streaming* de áudio e vídeo em alta definição. Em teoria o limite de conexões é de 256 dispositivos porém, a rede pode congestionar antes mesmo de alcançar o número máximo de dispositivos conectados, especialmente se eles demandarem grandes quantidades de banda (DAWOUD; DAWOUD, 2020, p. 538).

Os principais benefícios da *Wi-Fi* são velocidade de transmissão, alcance e sobretudo a disponibilidade da tecnologia. Muitas pessoas possuem roteador *Wi-Fi* em suas casas e instalar um roteador despende menos recursos do que instalar cabos *Ethernet* pela casa (DAWOUD; DAWOUD, 2020, p. 538).

Do ponto de vista da automação residencial, a *Wi-Fi* têm algumas desvantagens, tais como ser suscetível a interferência quando houverem muitos dispositivos competindo por largura de banda. Outro problema é o consumo de energia pelos dispositivos *Wi-Fi*, além disso o longo alcance e a alta velocidade requerem muita energia para operar (DAWOUD; DAWOUD, 2020, p. 538).

2.5 MQTT

Message Queuing Telemetry Transport (MQTT) é um protocolo usado para a comunicação entre máquinas e conectividade entre IoT. É um protocolo leve que trabalha com o mecanismo de *publish-subscribe* (publicação-assinatura) e executa sobre TCP/IP[10]. Dessa forma, qualquer dispositivo que possua TCP/IP e seja capaz de usar uma biblioteca MQTT pode ser um cliente MQTT (HILLAR, 2021, p. 31-32).

Segundo Yuan (2017a) o protocolo foi criado em 1999 pela IBM[11], sua aplicação original era vincular sensores em *pipelines* de petróleo a satélites. Atualmente o protocolo MQTT é o mais popular e com melhor recebimento no ramo da IoT em todo o mundo (HIVEMQ, 2021, p. 39).

Importante ressaltar que o MQTT não é mais considerado um acrônimo, mas sim, o nome do protocolo. Ele não é uma solução tradicional de enfileiramento de mensagem, apesar de ser possível enfileirar mensagens em certo casos (HIVEMQ, 2021, p. 5). No final de 2014 tornou-se oficialmente aberto com padrão OASIS[12], com suporte a diversas linguagens de programação (YUAN, 2017a).

[10] "TCP/IP é um acrônimo para o termo *Transmission Control Protocol/Internet Protocol Suite*, dois dos mais importantes protocolos que conformam a pilha de protocolos usados na Internet."(DUARTE, 2003)

[11] https://ibm.com/

[12] https://www.oasis-open.org/standards/

2.5.1 Padrão *publish/subscribe*

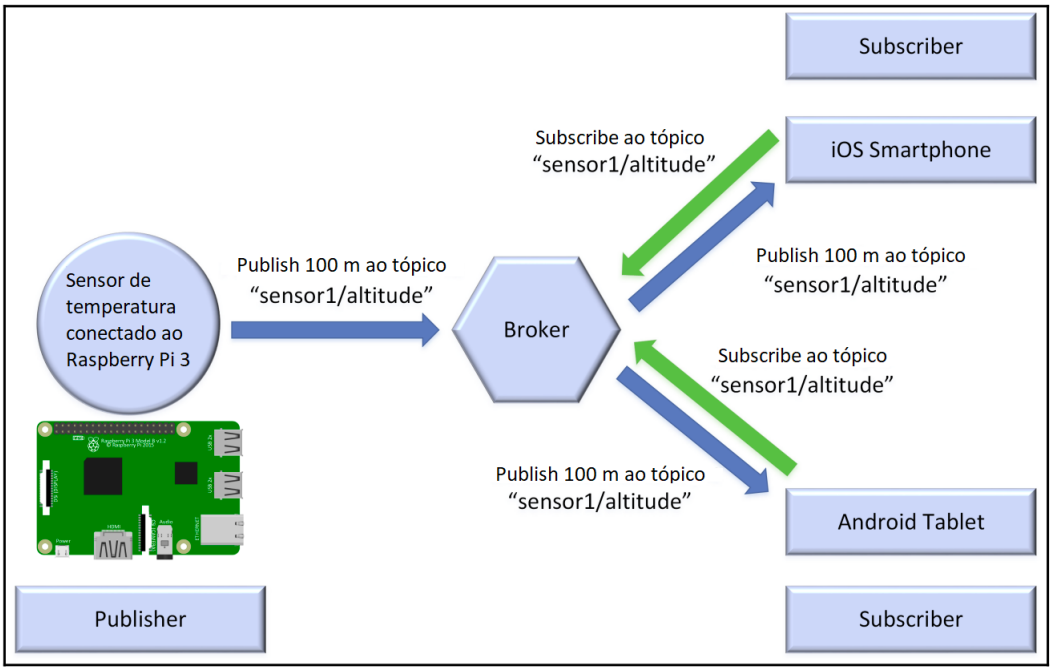
O padrão *publish/subscribe*, ou em português publicação/assinatura, usa um modelo que desacopla o cliente enviador da mensagem (*publishers*) do cliente receptor da mensagem (*subscribers*). Ambos, *publishers* e *subscribers* nunca têm contato direto, a conexão entre eles é feita por um terceiro componente chamado *broker*, cujo trabalho é filtrar as mensagens e distribuí-las para os assinantes (*subscribers*) (HIVEMQ, 2021, p. 7).

Cada mensagem deve conter um tópico que o *broker* utiliza para enviar a mensagem aos clientes que o estão assinando (*subscribers*). O tópico é uma *string* estruturada em hierarquia delimitada por barras (/) (HIVEMQ, 2021, p. 15). Por exemplo:

```
casa/quarto/temperatura
```

Na Figura 1, Hillar (2021, p. 27) mostra como o padrão funciona. No exemplo, um *Raspberry Pi* coleta a altitude proveniente de um sensor e a publica (*publish*) no *broker* pelo tópico "sensor1/altitude". Do outro lado, um *smartphone* e um *tablet* assinam (*subscribe*) o tópico "sensor1/altitude". O *broker* ao receber a mensagem no determinado tópico, irá buscar quais dispositivos o estão assinando e enviará a mensagem apenas para esses.

Figura 1 – Padrão *publish/subscribe*



Fonte: Adaptado de Hillar (2021).

Em um cliente *subscriber* é possível assinar um tópico em específico, como exemplificado na Figura 1, ou pode-se usar um curinga. Existem duas formas de usar curingas: nível único ou multinível (HIVEMQ, 2021, p. 19).

No nível único o curinga substitui o tópico de um nível. É utilizado o símbolo de soma (+) para definir o curinga de nível único (HIVEMQ, 2021, p. 19). No exemplo abaixo, serão coletadas as temperaturas de todos os cômodos da casa.

```
casa/+/temperatura
```

Já o multinível cobre uma série de níveis de tópicos. É utilizado o símbolo de cerquilha (#) no final da *string* do tópico para ser definido o filtro de todas as mensagens que serão enviadas para os tópicos que têm o padrão antes da cerquilha (#) (HIVEMQ, 2021, p. 20). No exemplo abaixo serão recebidas mensagens de todos os sensores que estiverem dentro do quarto.

```
casa/quarto/#
```

2.5.2 Níveis de qualidade do serviço

A *Quality of Service* (QoS), ou qualidade do serviço em português, é o acordo entre o enviador e o recebedor da mensagem que definem a garantia de entrega de uma determinada mensagem. Existem três níveis de qualidade em MQTT:

a) **QoS 0** é o menor nível, com ele a mensagem é enviada apenas uma vez, não havendo garantia de entrega (HIVEMQ, 2021, p. 22);

b) **QoS 1** garante a entrega da mensagem pelo menos uma vez ao receptor. O enviador irá receber um pacote de resposta que informa que a mensagem foi recebida. Se o enviador não receber a resposta em um determinado período de tempo, a mensagem será reenviada. Dessa forma, é possível que uma mensagem seja recebida duas vezes pelo receptor (HIVEMQ, 2021, p. 23);

c) **QoS 2** é o nível mais alto e nele é garantido que cada mensagem será recebida uma única vez pelo receptor. É o nível mais seguro porém, o mais lento. A garantia é provida por pelo menos dois fluxos de requisição/resposta entre o enviador e o receptor. Ambos usam pacotes identificados da mensagem original para coordenar a entrega. Quando o fluxo for completo, ambas as partes terão certeza que a mensagem foi entregue (HIVEMQ, 2021, p. 23-24).

Segundo HiveMQ (2021, p. 25), ao utilizar níveis de QoS, o *publisher* deve definir o QoS da mensagem que será enviada ao *broker*. Porém o *subscriber* também deve definir o nível de QoS ao assinar um tópico. Dessa forma, pode haver dois níveis de qualidade diferentes para a mesma mensagem. Na prática, o nível mais alto será definido pelo *publisher*, pois se o *publisher* utilizar QoS 1 e o *subscriber* utilizar QoS 2, a mensagem poderá ser enviada mais de uma vez para o *subscriber*, pois o QoS 1 garante a entrega da mensagem pelo menos uma vez porém, não previne múltiplas entregas.

2.5.3 Segurança

Na maioria dos serviços de MQTT *broker*, a configuração padrão é que os dados sejam transmitidos sem criptografia. Também, se as portas do *firewall* estiverem abertas e o roteador estiver com redirecionamento de porta, qualquer cliente MQTT com o *Internet Protocol* (IP) poderá publicar ou assinar qualquer tópico (HILLAR, 2021, p. 75), violando a integridade dos dados.

Para proteger a integridade dos dados, deve ser utilizado *Transport Layer Security* (TLS)[13] com o MQTT. Dessa forma, os dados serão privados pois estarão criptografados e íntegros já que não serão alterados no meio do caminho (HILLAR, 2021, p. 75).

Segundo Forouzan (2010, p. 990) o protocolo TLS tem por intuito fornecer a segurança das transações na *internet*, oferecendo segurança ponta a ponta para aplicações que usam o protocolo *Transmission Control Protocol* (TCP). O autor exemplifica os seguintes serviços de segurança:

a) Cliente precisa estar seguro que o servidor pertence ao verdadeiro fornecedor;

b) O conteúdo da transação entre cliente e servidor não pode ser modificado;

c) As informações confidenciais não podem ser interceptadas por um impostor.

2.5.4 Limitações

Apesar de todos os benefícios, existem algumas limitações que são importantes ressaltar, tais como o *publisher* e o *subscriber* devem saber quais tópicos eles devem usar; e o *publisher* não sabe se alguém está escutando suas mensagens.

É possível efetuar *streaming* de vídeo utilizando MQTT, porém a sua essência é trabalhar com dados leves e de forma assíncrona (HILLAR, 2021, p. 32), por isso não é a melhor escolha.

2.6 *STREAMING*

Streaming consiste na transmissão contínua de arquivos de áudio ou vídeo de um servidor para um cliente. O arquivo é quebrado em pacotes com um determinado número de segundos cada e é adicionada uma identificação para manter a ordem de reprodução, sendo que, o arquivo de mídia executado no dispositivo do cliente é armazenado remotamente (CLOUDFLARE, 2021c).

Outra forma de reproduzir áudio ou vídeo de um servidor é fazendo *download* do arquivo para o dispositivo, porém isso implica em ter que esperar a conclusão do *download* para

[13] TLS é um protocolo que tem por objetivo prover privacidade e integridade dos dados entre a comunicação de duas aplicações (RESCORLA; DIERKS, 2008, p. 3).

ser possível reproduzir o arquivo. Além disso, o arquivo é armazenado no dispositivo, diferentemente do *streaming* que é em tempo-real, sendo executadas pequenas partes da mídia uma após a outra (CLOUDFLARE, 2021c).

Para que o áudio ou vídeo sejam reproduzidos de forma suave e constante, é utilizada a prática de *buffering*, que consiste em pré-carregar um segmento de dados, para ser possível continuar reproduzindo quando há uma breve interrupção na conexão (CLOUDFLARE, 2021a).

2.6.1 *Live streaming*

Live streaming, ou transmissão ao vivo em português, acontece quando um vídeo é enviado pela *internet* em tempo-real, sem ser gravado e armazenado. O termo usualmente refere-se a conexões *one-to-many*[14] (CLOUDFLARE, 2021b).

Segundo a CloudFlare (2021b), para ser realizada uma transmissão ao vivo devem ser efetuados os seguintes passos:

1. **Captura do vídeo**: Tudo começa pela informação captada pela câmera em que o dispositivo computacional está anexado, sendo que a informação é representada de forma digital;

2. **Compressão**: O próximo passo é comprimir o dado, removendo as informações visuais redundantes. Por exemplo, se um *frame* (quadro) mostra uma pessoa falando com um fundo branco, o fundo branco não precisa ser renderizado para nenhum subsequente *frame* que tenha o mesmo fundo;

3. **Codificação**: A codificação refere-se ao processo de converter o dado em um novo formato. Alguns padrões de codificação para vídeo são: H.264, H.265, VP9 e AV1;

4. **Segmentação**: Pelo fato dos vídeos serem pesados, sua transmissão pela *internet* normalmente é demorada. Para resolver esse problema, o vídeo é dividido em diversos segmentos com alguns segundos de comprimento;

5. **Decodificação e reprodução de vídeo**: Cada dispositivo que está assistindo o *streaming* recebe, decodifica e descomprime o segmento do vídeo. Por fim, um reprodutor de mídia, no dispositivo do usuário interpreta os dados como uma informação visual, reproduzindo o vídeo.

2.7 SERVIDORES

Servidores são computadores dedicados a executarem uma tarefa em específico. Atualmente existem duas formas onde o *hardware* que irá executar a aplicação pode estar. Nas

[14] Um para muitos (1-n)

próximas seções será explicado o que é, quais as vantagens e desvantagens em utilizar *on premise* ou *cloud computiong*.

2.7.1 *On premise*

On premise refere-se a utilização de servidores que estão alocados na empresa ou domicílio. Estes servidores normalmente contam com controle de temperatura por meio de arcondicionados para evitar o superaquecimento, também requerem a utilização de *nobreak* para prevenir quedas de energia e rotinas de *backups* para resguardar possíveis falhas (IPSENSE, 2017).

Os servidores são adquiridos conforme a necessidade computacional (EVOLVIT, 2019). Segundo Mahalle *et al.* (2021, o. 46), pequenas aplicações de uso autônomo, tais como casas inteligentes, precisam de recursos de processamento e espaço em memória pequenos, com isso podem ser utilizados dispositivos próprios (*smartphones*, *raspberry*).

Como benefícios destacam-se o controle total sobre os dados e controle sobre a segurança. Ao faltar energia ou *internet*, as funções críticas podem rodar por meio de um *backup* (EVOLVIT, 2019).

Já como desvantagens destacam-se a necessidade de comprar *hardware* que tenha capacidade computacional suficiente, bem como ter a responsabilidade de manter e atualizar o *hardware* e *software*. Dependendo do requerimento de energia do equipamento, pode haver um aumento considerável na conta de luz (EVOLVIT, 2019).

2.7.2 *Cloud computing*

Cloud computing, ou computação em nuvem, consiste no fornecimento de banco de dados, *software*, rede, entre outros, de forma flexível como um serviço por meio da *internet* (MICROSOFT, 2021). Um espaço em um servidor deve ser alugado por meio de uma empresa provedora ao invés de dispor de um servidor próprio, sendo que o provedor é responsável por toda a infraestrutura e manutenção dos servidores (IPSENSE, 2017). A *Cloud computing* possui seus princípios fundamentados em compartilhamento e maximização da disponibilidade dos recursos (MAHALLE *et al.*, 2021, p. 46). Por ser independente da localização, a nuvem permite acesso de qualquer lugar por qualquer dispositivo através da *internet* (MAHALLE *et al.*, 2021, p. 46).

Atualmente existem três tipos de modelo de serviço, sendo eles:

a) ***Software as a Service* (SaaS)**: O modelo SaaS permite ao usuário usar qualquer aplicação *web* sem a necessidade de baixar ou instalar a aplicação/*software* localmente. Toda a aplicação é gerenciada pelo provedor do serviço (MAHALLE *et al.*, 2021, p. 43). Alguns exemplos são: Dropbox, Microsoft Office 365 e Google Drive;

b) ***Platform as a Service* (PaaS)**: O modelo PaaS permite ao cliente desenvolver e testar diversas aplicações conforme a sua necessidade, por meio de um ambiente virtual de execução. Com isso, são eliminadas as preocupações sobre as configurações do sistema operacional, *middleware*[15] e ambientes individuais para cada máquina. Junto com o ambiente de execução, outros recursos tais como armazenamento, servidores e comunicação são mantidos pelo provedor do serviço (MAHALLE *et al.*, 2021, p. 43). Alguns exemplos são: Amazon Web Services (AWS) e Google App Engine;

c) ***Infrastructure as a Service* (IaaS)**: O modelo IaaS fornece ao usuário toda a infraestrutura, tais como servidores, armazenamento e rede, requeridos para a operação do negócio, eliminando a preocupação do cliente sobre os custos da aquisição, configuração e manutenção da infraestrutura (MAHALLE *et al.*, 2021, p. 43). Alguns exemplos são: Microsoft Azure, Google Cloud e IBM Cloud.

Como benefícios destacam-se o relativo padrão nos valores, permitindo um melhor planejamento dos gastos. Preço flexível e opções para customização permitem escalar conforme a demanda de forma rápida. O provedor é quem se responsabiliza pela manutenção da infraestrutura (EVOLVIT, 2019).

Já como desvantagem destacam-se problemas na rede que podem inviabilizar o acesso aos dados. Servidores baseados na nuvem são mais visados por cibercriminosos. É preciso confiar na segurança contra vulnerabilidades do provedor do serviço (EVOLVIT, 2019)

2.8 TRABALHOS RELACIONADOS

Com a popularização da IoT, diversos trabalhos foram propostos na academia e diversos outros estão no mercado solucionando diversos problemas. Entre tantos, serão apresentados a seguir três trabalhos acadêmicos e uma solução de mercado, respectivamente.

Marchesan (2012) desenvolveu um sistema de segurança residencial de baixo custo, utilizando *Arduino*, em que é possível acessar pela *internet* os sensores dispostos pela residência que enviam as alterações ocorridas para um sistema *web* através da *internet*, sendo que esse sistema possui uma interface para acesso dos usuários.

Santos (2012) teve por objetivo desenvolver um sistema para detecção de gás GLP utilizando o sensor MQ-9 com avisos por SMS, sendo que dependendo do nível de gás presente no ambiente foram criadas regras para emitir um aviso sonoro, exibir uma mensagem em um *display* LCD ou enviar SMS para um determinado celular.

Já Bruxel (2015) criou um sistema para facilitar o monitoramento de um determinado ambiente, em que uma câmera é capaz de acompanhar o movimento de objetos por meio de

[15] Segundo Red Hat (2021) "*Middleware* é um *software* que fornece serviços e recursos comuns a aplicações. Gerenciamento de dados, serviços de aplicações, sistema de mensageria, autenticação e gerenciamento de APIs são recursos comumente operados por um software de *middleware*".

detecção de movimento por sensores de movimento.

Como solução de mercado a Positivo (2021) possui um *kit* para controlar movimentos de possíveis intrusos, uma câmera e sensores de abertura para detectar a abertura de uma janela, por exemplo. Todos os dispositivos se conectam a uma central, que por sua vez conecta-se à *internet*, sendo possível controlar os dispositivos por meio de um aplicativo móvel.

2.9 CONCLUSÕES DO CAPÍTULO

Diante dos estudos feitos neste capítulo, chegou-se a conclusão de que um sistema de envio de notificações aumenta a segurança dos residentes. Sensores conectados a dispositivos de IoT podem detectar possíveis ameaças e notificar os residentes ou até mesmo as autoridades. No Capítulo 3 será detalhado como o sistema de envio de notificações foi arquitetado.

3 DESCRIÇÃO DA ARQUITETURA DO SISTEMA

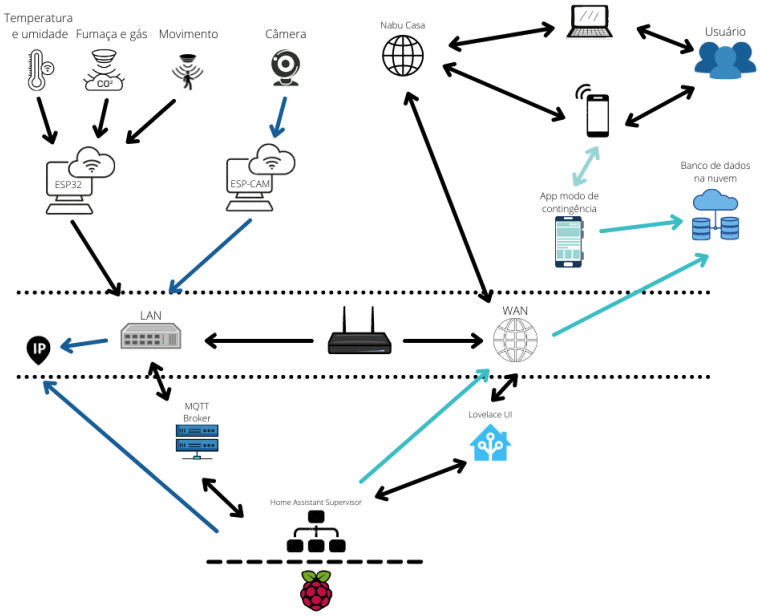
Para que um sistema de segurança baseado em notificações atue de forma efetiva é preciso que essas notificações sejam enviadas de forma precisa e o mais rápido possível para que a contramedida seja executada o quanto antes. Para isso, são necessários dispositivos que coletam dados do ambiente e enviam notificações para os residentes independente do local que estejam. Atualmente, como a maioria das pessoas possuem *smartphones* é vantajoso usá-los como receptores destas notificações. Dessa forma, o residente consegue executar uma contramedida de forma remota por meio da *internet*. Além disso, a aplicação também pode ser acessada por meio de um computador para que o usuário tenha acesso aos dados, devendo ser uma aplicação multiplataforma.

Na Seção 3.1 será apresentada de forma geral a arquitetura do sistema. Na Seção 3.2 serão apontados os casos de uso para limitar as funcionalidades do sistema. Na Seção 3.3 serão exibidos os diagramas de atividades para demonstrar os fluxos do programa. Nas Seções 3.4 e 3.5 serão expostas a arquitetura e as configurações do *Home Assistant*, respectivamente.

3.1 ARQUITETURA GERAL

A arquitetura proposta foi dividida em quatro partes, sendo elas: coleta e transmissão dos dados, processamento e armazenamento dos dados, acesso aos dados, e modo de contingência. Na Figura 2 é ilustrada a arquitetura geral do sistema e como os dispositivos se comunicam.

Figura 2 – Arquitetura geral do sistema



Fonte: Próprio autor (2022).

Para a coleta dos dados foram utilizados sensores e câmeras que são controlados por microcontroladores. Por tratar-se de um sistema de segurança domiciliar os sensores utilizados detectarão gás e fumaça, temperatura e umidade, movimento e também uma câmera para o monitoramento em tempo real.

Para enviar os dados coletados pelos sensores para o servidor, foi utilizado o protocolo MQTT, em que o microcontrolador publica em determinados tópicos e o servidor está escutando estes tópicos para processar e armazenar os dados. Já a câmera faz *streaming* do vídeo no formato *Motion* JPEG (MJPEG) em um determinado endereço de IP.

Como servidor foi utilizado um *Raspberry Pi* que executa o *Home Assistant* e um MQTT *broker*. O *Home Assistant* assina os tópicos definidos, grava os dados recebidos em seu banco de dados e, conforme as regras definidas, envia as notificações para os usuários estipulados. O *Home Assistant* também recebe o *streaming* de vídeo, que tem a função de detectar movimento na imagem, gravar as imagens da detecção e enviar a notificação.

Para ter acesso aos dados, o *Home Assistant* fornece uma interface gráfica personalizável e multiplataforma, baseada em *dashboards* que pode ser configurada conforme a preferência do usuário, sendo possível acessá-la por qualquer dispositivo que possua um navegador de *internet*, já que possui uma interface *web*. Todavia, com os *smartphones* é possível acessá-la por meio de um aplicativo, tanto para *Android* quanto para *iOS*, tendo a vantagem de ser possível receber as notificações pelo aplicativo.

Diante do fato que o servidor está hospedado localmente, caso falte energia ou *internet* os dados não estarão disponíveis. Diante disso, viu-se a necessidade de replicar os dados em um banco de dados na nuvem, para que estes dados possam ser acessados pelos residentes através de seus *smartphones* e que eles recebam notificações quando a instância local estiver *offline*. Para armazenar os dados na nuvem foi escolhido um banco de dados NoSQL para armazenar os dados dos sensores e um banco de dados para arquivos para armazenar as imagens captadas pela câmera nas detecções de movimento. Para disponibilizar os dados armazenadas na nuvem foi desenvolvido um aplicativo para dispositivos móveis que além de ter a função de exibir os dados, mostra e envia notificações caso a instância do *Home Assistant* esteja *offline*.

3.2 DIAGRAMA DE CASOS DE USO

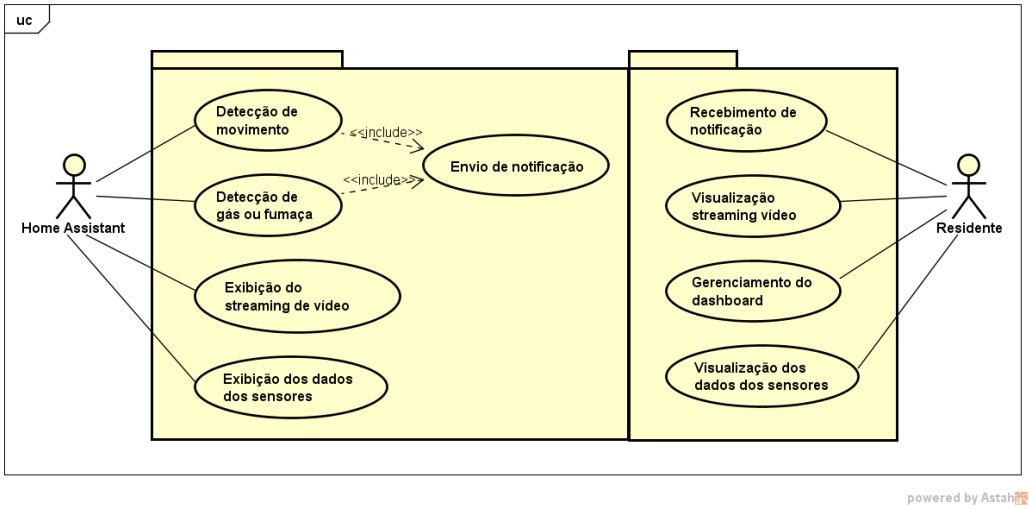
Segundo Fowler (2003, p. 102), casos de uso é uma técnica utilizada para capturar os requerimentos funcionais do sistema, eles descrevem as interações entre os usuários com o sistema. O diagrama de casos de uso mostra o ator, os casos de uso e como eles se relacionam. A Figura 3 demonstra o diagrama de casos de uso deste trabalho, sendo que os usuários foram divididos em dois, sendo eles: Home Assistant e Residente.

O Home Assistant é um computador que tem o papel de detectar movimento, fuga de gás e fumaça, sendo que quando forem detectadas, ele envia notificações para os residentes,

além de possuir a função de exibir para os residentes o *streaming* de vídeo e os dados coletados pelos sensores (temperatura, umidade, movimento, gás e fumaça).

Já o Residente, representa uma pessoa que mora na residência onde o sistema está implantado. Ele tem papel de visualizar e analisar os dados coletados pelos sensores, receber as notificações enviadas pelo *Home Assistant*, visualizar o *streaming* de vídeo e gerenciar os *dashboards* conforme sua preferência.

Figura 3 – Diagrama de casos de uso



Fonte: Próprio autor (2021).

3.3 DIAGRAMA DE ATIVIDADES

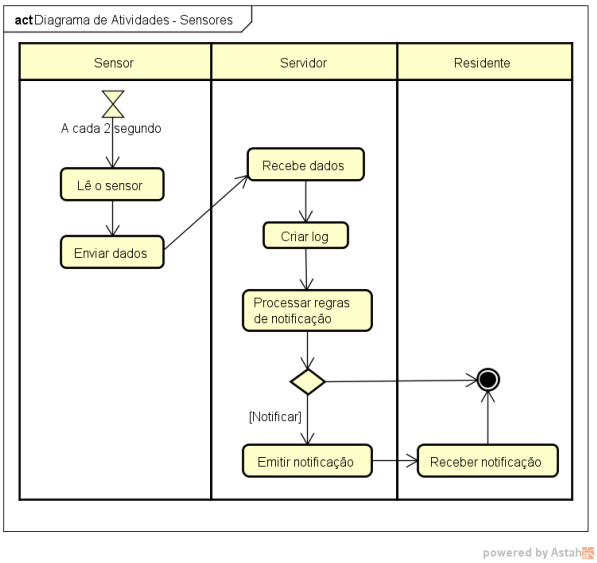
Segundo Fowler (2003, p. 117), diagrama de atividades é a técnica utilizada para descrever lógica procedural, regras de negócio e fluxos de trabalho. Para este trabalho foi necessária a criação de três diagramas de atividades: um para demonstrar o fluxo dos sensores, outro para o *streaming* de vídeo e um para demonstrar a replicação dos dados para a nuvem.

O fluxo de trabalho dos sensores é exibido no diagrama de atividades da Figura 4a, em que a cada dois segundos o dado será coletado do sensor e enviado para o servidor. É importante ressaltar que este diagrama é referente aos sensores de temperatura e umidade, fumaça e gás, já que todos eles possuem o mesmo fluxo.

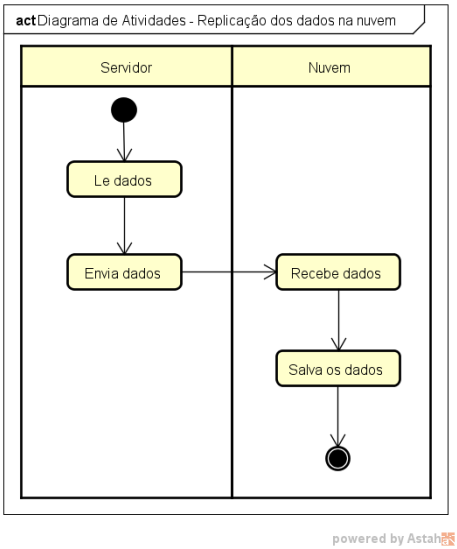
O segundo diagrama de atividades, exibido na Figura 4c, é referente ao *streaming* de vídeo, que além de detectar movimento, deve disponibilizar o *streaming* do vídeo para os residentes visualizarem.

O diagrama exibido na Figura 4b, mostra o fluxo do sistema para enviar os dados, oriundos dos sensores, do banco de dados do *Home Assistant* para um banco de dados hospedado na nuvem.

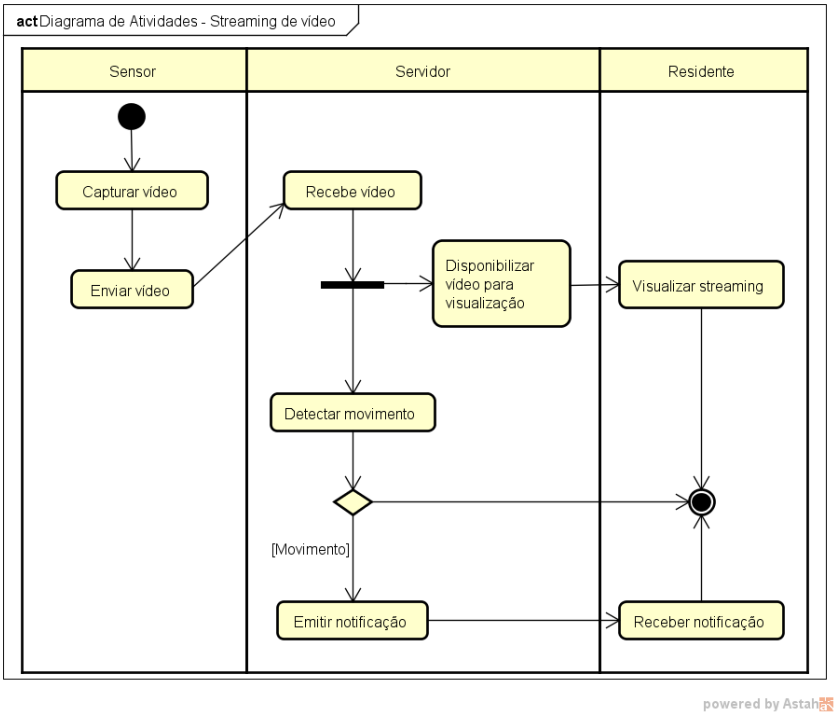
Figura 4 – Diagramas de atividades



(a) Sensores



(b) Replicação dos dados na nuvem



(c) *Streaming* de vídeo

Fonte: Próprio autor (2022).

3.4 ARQUITETURA DO *HOME ASSISTANT*

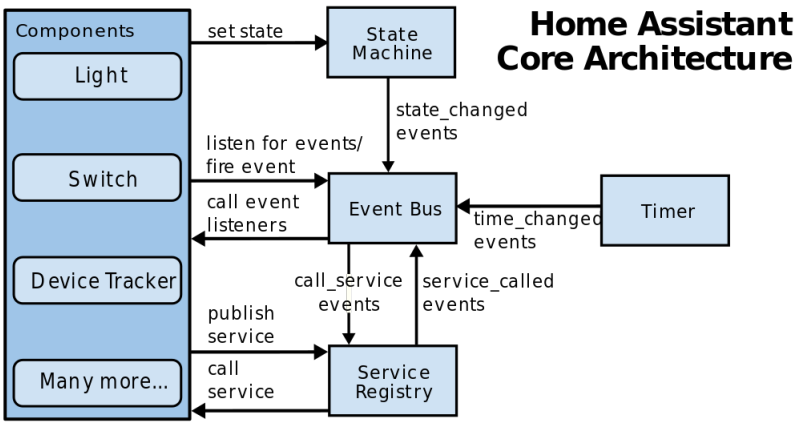
Segundo Home Assistant (2021a) a arquitetura de aplicação divide-se em três partes, sendo elas Sistema Operacional, Supervisor e *Core*, em que cada camada possui uma atribuição específica que corrobora para o funcionamento integral do sistema. A Figura 5 ilustra a composição e hierarquia das diferentes camadas.

Figura 5 – Arquitetura do *Home Assistant*

Fonte: Adaptado de Home Assistant (2021a).

- O **Sistema operacional** provê um ambiente Linux mínimo para executar o Supervisor e o *Core*. O *Home Assistant Operating System* (HAOS) foi construído com o propósito de executar o *Home Assistant* em computadores *single board* e sistemas x84-64, com foco em prover robustez, sem haver manutenções no sistema operacional (HOME ASSISTANT, 2021h).
- O **Supervisor** permite ao usuário gerenciar a instalação do *Home Assistant* pelo próprio *Home Assistant*. Suas responsabilidades são executar o *Home Assistant Core*; atualizar o *Home Assistant Core*; fazer e restaurar *backups*; *Add-ons*; e atualizar o HAOS, dependendo da instalação.
- O ***Core*** interage com o usuário, supervisor e os dispositivos e serviços de IoT. Inclui diversos *helpers* para tratar cenários comuns. Segundo Home Assistant (2021e), o *core* consiste em quatro partes principais, a Figura 6 demonstra a disposição delas.

Figura 6 – Arquitetura do *Home Assistant Core*



Fonte: Home Assistant (2021e).

Onde:

- *Event Bus*: É o meio utilizado para enviar e receber os eventos, ele facilita o disparo e escuta dos eventos;
- Estado de máquina: Acompanha os estados dos elementos e dispara um evento `state_changed` quando algum estado mudar;
- Registro de serviço: Escuta ao *Event Bus* por um evento de `call_service` e permite outro código registrar serviços;
- Temporizador: Envia um evento de `time_changed` a cada um segundo no *Event Bus*.

3.5 CONFIGURAÇÕES DO *HOME ASSISTANT*

O *Home Assistant* é um sistema dinâmico que pode ser configurado conforme cada necessidade, sendo que o que provê essa dinamicidade são os arquivos de configurações YAML[1] presentes no *Home Assistant*. Apesar de sua configuração poder ser feita através da interface do usuário, também é possível configurá-lo diretamente pelos arquivos de configuração (HOME ASSISTANT, 2021d). O arquivo `configuration.yaml` é o arquivo principal de configuração, ele contém as configurações das integrações que devem ser carregadas. A Figura 7 ilustra um exemplo de arquivo de configuração.

Figura 7 – Exemplo de arquivo de configuração do *Home Assistant*

```
1  # Configure a default setup of Home Assistant (frontend, api, etc)
2  default_config:
3
4  # Text to speech
5  tts:
6    - platform: google_translate
7
8  group: !include groups.yaml
9  automation: !include automations.yaml
10 script: !include scripts.yaml
11 scene: !include scenes.yaml
12
13 camera:
14   - platform: mjpeg
15     mjpeg_url: http://10.0.0.113
16
17 sensor:
18   - platform: mqtt
19     state_topic: "cozinha/temperatura"
20
```

Fonte: Próprio autor (2021).

[1] YAML não é uma linguagem de marcação, mas sim uma linguagem de serialização de dados amigável para todas as linguagens de programação (YAML, 2021).

Home Assistant (2021n) indica que não devem ser armazenados dados privados no arquivo `configuration.yaml`, já que ele é um arquivo de texto simples, dessa forma os dados privados devem ser armazenados em um arquivo separado ou em variáveis de ambiente. Por padrão o arquivo é chamado `secrets.yaml`, sendo que após serem adicionados os dados privados no arquivo, deve ser utilizada a diretiva `secret` no arquivo `configuration.yaml` para ser chamado o dado desejado.

Add-ons permitem aos usuários estender funcionalidades do seu *Home Assistant*, podendo instalar diversas aplicações que estão disponíveis na loja ou criar novos *add-ons*, eles são configurados e instalados pelo painel do supervisor. É possível executar aplicações que o *Home Assistant* possa integrar com o MQTT *Broker* ou para compartilhar as configurações via Samba[2] para uma edição facilitada por outros computadores. Importante ressaltar que os *add-ons* estão disponíveis apenas para instalações com *Home Assistant Operation System* ou *Home Assistant Supervised* (HOME ASSISTANT, 2021f).

Integrações podem ser estendidas pelo *Home Assistant Core*, sendo que cada uma delas é responsável por um domínio específico. As integrações podem escutar ou disparar eventos, oferecer serviços e manter estados. O *Home Assistant* oferece diversas integrações prontas para uso, todavia permite a criação de novas integrações, que devem ser escritas em *Python* (HOME ASSISTANT, 2021j). Exemplos de integrações prontas para uso é os clientes MQTT, IFTTT, entre diversas outras.

Automações no *Home Assistant* permitem responder de forma automática a determinados eventos, tais como acender as luzes ao pôr do sol ou enviar uma notificação caso seja detectado movimento. O *Home Assistant* possui informações sobre todos os dispositivos e serviços, sendo que elas estão disponíveis para os usuários por meio dos *dashboards* e podem ser usadas para disparar automações (HOME ASSISTANT, 2021c), sendo possível criar regras para determinar quando cada ação deve acontecer.

Scripts por sua ver, permitem aos usuários especificarem sequências de ações para serem executadas pelo *Home Assistant*, podendo ser executadas manualmente ou por meio de automações, por exemplo, como um serviço (HOME ASSISTANT, 2021m).

Para prover segurança e restringir o acesso ao sistema, o *Home Assistant* possui um sistema de autenticação embutido, permitindo que diferentes usuários interajam com o *Home Assistant* (HOME ASSISTANT, 2021b). Cada usuário tem um *login* e senha, tendo a possibilidade de utilizar autenticação por multi-fator e a possibilidade de autenticar com o mesmo usuário em diversos dispositivos. Os usuários são gerenciados por usuários administradores, sendo que qualquer usuário pode ser definido como administrador.

[2] Um servidor de arquivos Samba permite compartilhar arquivos entre diferentes sistemas operacionais pela rede. (UBUNTU, 2021)

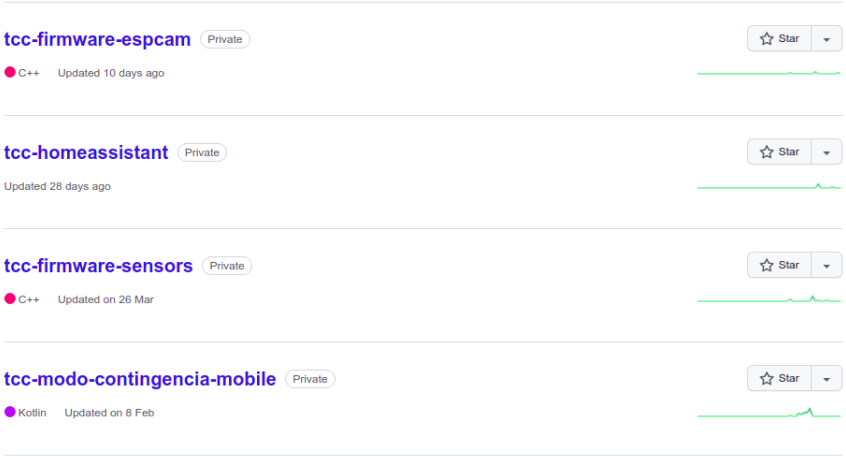
3.6 CONCLUSÕES DO CAPÍTULO

Após o desenvolvimento da descrição da arquitetura do sistema viu-se a necessidade do desenvolvimento de *firmwares* para controlar os sensores e a câmera, um aplicativo móvel para acesso aos dados e recebimento de notificações para o modo de contingência e a configuração do *Home Assistant*. Apesar do *Home Assistant* ser uma solução pronta para uso, por trabalhar com arquivos de configuração `.yaml`, é possível configurá-lo conforme a necessidade de cada aplicação, além de permitir desenvolver novas integrações e *add-os*. Diante da arquitetura exposta, o Capítulo 4 irá detalhar o desenvolvimento e o funcionamento do sistema proposto.

4 IMPLEMENTAÇÃO DA SOLUÇÃO

Neste capítulo será abordado como foram desenvolvidos os *softwares* deste trabalho, quais tecnologias foram utilizadas e como elas se comunicam. Este trabalho foi dividido em quatro *softwares*, sendo que todos eles foram versionados utilizando *git*[1] e seus repositórios foram armazenados no *GitHub*[2]. A Figura 8 mostra como os repositórios foram divididos.

Figura 8 – Repositórios das aplicações



Fonte: Próprio autor (2022).

- **tcc-firmware-espcam**[3]: Contém o código para fazer o *streaming* de vídeo através da ESP-CAM e disponibilizar o vídeo na rede local em um determinado endereço de IP;
- **tcc-homeassistant**[4]: Contém os arquivos de configuração do *Home Assistant* e a integração desenvolvida;
- **tcc-firmware-sensors**[5]: Contém o código para coletar os dados dos sensores e enviá-los para o MQTT *broker*;
- **tcc-modo-contingencia-mobile**[6]: Contém o código fonte do aplicativo móvel utilizado para exibir os dados e as notificações no modo de contingência.

Na Seção 4.1 será apresentado como os dados são coletados e transmitidos dos sensores para o servidor. Na Seção 4.2 será abordado como os dados oriundos dos sensores são

[1] *Git* é um sistema *open source* para controle de versão distribuída, desenhada para pequenos e grandes projetos (GIT, 2022);
[2] https://github.com/
[3] https://github.com/aimperatori/tcc-firmware-espcam
[4] https://github.com/aimperatori/tcc-homeassistant
[5] https://github.com/aimperatori/tcc-firmware-sensors
[6] https://github.com/aimperatori/tcc-modo-contingencia-mobile

processados, armazenados e como o servidor foi implementado. Na Seção 4.3 é exposto como os residentes acessam o sistema. Na Seção 4.4 será explicado o funcionamento do modo de contingência e como ele foi implementado. E por fim, na Seção 4.5 é exporto a estimativa de custo da aplicação proposta.

4.1 COLETA E TRANSMISSÃO DOS DADOS

Para ser possível detectar ameaças na residência são necessários sensores que coletem dados do ambiente onde estejam inseridos. Junto a eles estão os microcontroladores que têm a função de comandá-los estipulando regras para a coleta, processamento e transmissão dos dados coletados.

Para coletar os dados do ambiente a fim de prover segurança domiciliar, foram utilizados sensores de temperatura e umidade, fumaça e gás, movimento, e também uma câmera para vídeo monitoramento. Após coletar os dados dos sensores é preciso enviá-los ao servidor, sendo que é preciso de um dispositivo que tenha essa capacidade. Para isso foram utilizados microcontroladores, que enviam os dados através de uma conexão sem fio (*wireless*).

As próximas subseções irão explicar detalhadamente como a coleta e a transmissão dos dados foram implementadas.

4.1.1 Sensor de gás e fumaça

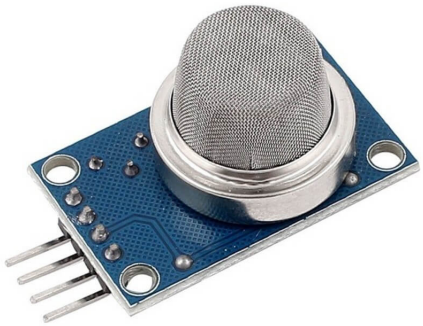
Sensores de gás são dispositivos que detectam e identificam diferentes tipos de gases, sendo que eles variam de tamanho, cobertura e sensibilidade (ROBU.IN, 2020). A série MQ possui diversos tipos de sensores para cada tipo de gás, ROBU.IN (2020) lista os sensores da série MQ e quais gases eles detectam:

- MQ-2: Gás metano, butano, Gás Liquefeito de Petróleo (GLP) e fumaça;
- MQ-3: Álcool, etanol e fumaça;
- MQ-4: Metano e Gás Natural Veicular (GNV);
- MQ-5: Gás natural e GLP;
- MQ-6: GLP e gás butano;
- MQ-7: Monóxido de carbono;
- MQ-8: Gás hidrogênio;
- MQ-9: Monóxido de carbono e gases inflamáveis;
- MQ131: Ozônio;

- MQ135: Qualidade do ar (CO, amônia, benzeno, álcool e fumaça);
- MQ136: Gás sulfeto de hidrogênio;
- MQ137: Amônia;
- MQ138: Benzeno, tolueno, álcool, acetona, propano, gás formaldeído e hidrogênio;
- MQ214: Metano e gás natural.

O MQ-2 é o sensor que mais se adéqua às residências pelo fato de detectar GLP, o popular "gás de cozinha", que é amplamente utilizado nos domicílios. Outra vantagem é que ele detecta fumaça, possibilitando a identificação de possíveis incêndios. Por esses motivos o sensor MQ-2 foi utilizado neste trabalho. Segundo ROBU.IN (2021) o sensor MQ-2 tem uma amplitude de detecção de 300 a 10.000 PPM[7], possui alta sensibilidade e rápido tempo de resposta, sendo que a sensibilidade pode ser ajustada por meio de um potenciômetro. A Figura 9 ilustra o sensor.

Figura 9 – Sensor de gás e fumaça MQ-2



Fonte: Instituto Digital (2021).

Segundo PROCTOR; HUGHES; FISCHMAN[8] (1988 apud CDC, 1994), GLP em concentrações extremas pode causar asfixia. Ainda segundo o autor, por causar asfixia em concentrações bem acima do limite explosivo inferior, o *Immediately Dangerous to Life or Health* (IDLH) é de 2.000 PPM. Por isso observou-se a necessidade de criar regras para envio de notificações, quando for detectado um PPM inferior a 2.000 PPM, para que, dessa forma, seja possível efetuar uma contramedida antes que haja perigo ou danos à vida.

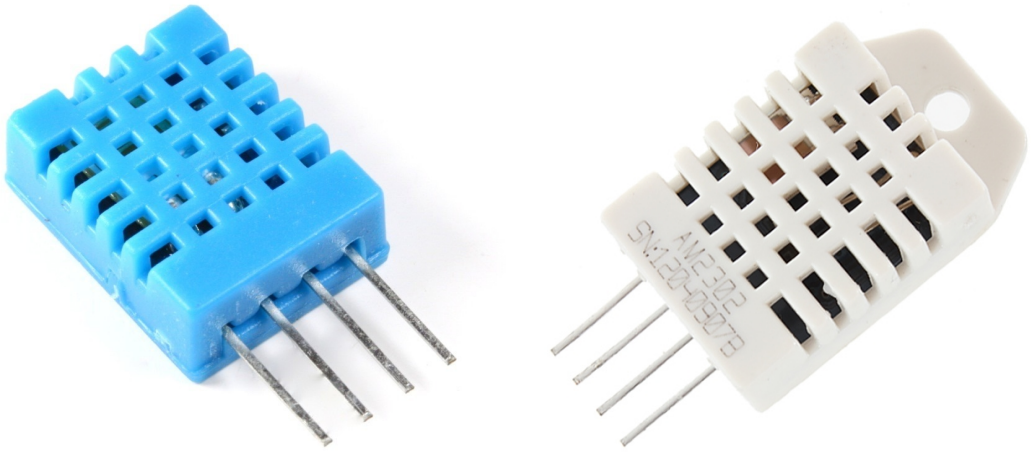
[7] Segundo ROBU.IN (2020), Partes por Milhão (PPM) é a medida utilizada para medir a quantidade de moléculas de gás em razão a outros gases. Por exemplo, 1.000 PPM de CO significa que o ambiente possui 1.000 moléculas de CO e 999.000 moléculas de outros gases.

[8] Proctor, N.H.; Hughes, J.P.; Fischman, M.L. **Liquified petroleum gas.** In: Chemical hazards of the workplace, 2. ed. Filadélfia, 1988. 420-421 p.

4.1.2 Sensor de temperatura e umidade

Sensores de temperatura e umidade têm a função de medir a temperatura e umidade do ambiente onde estão inseridos. Existem diversos sensores, cada um para determinada aplicação, variando de tamanho, limite de temperatura e umidade. Pode-se destacar os sensores DHT11 e DHT22, que são ilustrados na Figura 10.

Figura 10 – Sensores de temperatura e umidade



(a) DHT11 (b) DHT22

Fonte: FilipeFlop (2021c) e FilipeFlop (2021b).

Quadro 1 – Comparativo entre os sensores DHT11 e DHT22

	DHT11	DHT22
Alimentação	3-5VDC	3-5VDC
Umidade	10 a 90% UR	0 a 100% UR
Temperatura	0º a 50ºC	-40º a +80ºC
Precisão umidade	±5,0% UR	±2,0% UR
Precisão temperatura	±2,0ºC	±0,5ºC
Tempo de resposta	2s	2s
Dimensões (sem terminais)	15 x 12 x 5 mm	25 x 15 x 7 mm
Preço	R$ 16,90	R$ 59,90

Fonte: Adaptado de FilipeFlop (2021c) e FilipeFlop (2021b).

O Quadro 1 estabelece um comparativo entre os dois modelos de sensores. Pode-se observar que o sensor DHT22 tem uma maior abrangência e precisão tanto na temperatura quanto na umidade, porém há uma diferença considerável no preço dos sensores. Dessa forma, pelo fato de uma residência não necessitar de uma abrangência e precisão tão alta, foi utilizado o sensor DHT11 para baratear a solução.

4.1.3 Sensor de movimento

Sensores de movimento detectam movimento em uma determinada área onde estão direcionados. Existem tecnologias distintas para detectar movimento, sendo algumas delas *Passive Infrared Sensor* (PIR), sensor infravermelho ativo e sensor por micro-ondas que apesar de terem a mesma função, funcionam de formas diferentes.

O sensor *Passive Infrared Sensor* (PIR) utiliza luz infravermelha para identificar assinaturas de calor em corpos quentes. Desta forma, são detectadas grandes quantidades de calor como um corpo humano, um animal ou um veículo. Entretanto, esse sensor tem dificuldade em detectar pequenos movimentos, o que pode representar uma vantagem já que o torna menos passível a ruídos, tais como o vento ou o movimento de árvores (FILIPEFLOP, 2021d).

Já o sensor infravermelho ativo funciona como um transmissor e um receptor em que o transmissor emite um feixe de luz infravermelho até o receptor, criando uma espécie de cerca invisível. Quando algo ou alguém passa pelo feixe, o sensor é acionado, tendo seu uso principalmente para delimitação de perímetros (INTELBRAS, 2018).

O sensor por micro-ondas detecta o movimento através da reflexão das ondas emitidas pelos objetos presentes em seu campo de visão, sendo possível detectar movimento através de materiais não metálicos, tais como vidro, madeira e plástico (FILIPEFLOP, 2021a). Ele tem a capacidade de detectar pequenos movimentos, porém isso o torna mais passível a ruídos. Para um bom funcionamento, deve haver uma distância entre os sensores de, no mínimo, um metro, pois podem ocorrer interferências entre eles (FILIPEFLOP, 2021a).

Como o objetivo deste trabalho é detectar movimentações de possíveis invasores para enviar notificações para os residentes, o sensor que melhor se adéqua é o PIR, já que ele detecta assinaturas de calor e é menos propenso a ruídos. A Figura 11 ilustra o sensor utilizado.

Figura 11 – Sensor de movimento PIR

Fonte: Próprio autor (2021).

4.1.4 Câmera

Existem diversas aplicações para a utilização das câmeras, para este trabalho a câmera foi utilizada para vídeo monitoramento. Para isso, é preciso que os residentes tenham acesso ao vídeo por meio da *internet*, para ser possível monitorar a residência em qualquer lugar. Existem diversos modelos de câmeras no mercado, tanto para ambientes internos, quanto externo e com os mais variados preços.

Dessa forma, para essa solução foi utilizado o módulo ESP32-CAM, que será introduzido na Seção 4.1.5, por possuir um bom custo-benefício, dispor da tecnologia *Wi-Fi* e ser programável. Este módulo possui uma câmera que tem a capacidade de capturar vídeos com taxa de atualização de três *Frames Per Second* (FPS) na resolução máxima de 1.600 x 1.200, ou 30 FPS com a resolução mínima de 160 x 120 (CAMERON, 2021, p. 40).

4.1.5 Microcontroladores

Para controlar os periféricos expostos, viu-se a necessidade da utilização de dispositivos que tenham a capacidade de coletar os dados dos sensores e da câmera, processá-los e transmitir os dados para o servidor.

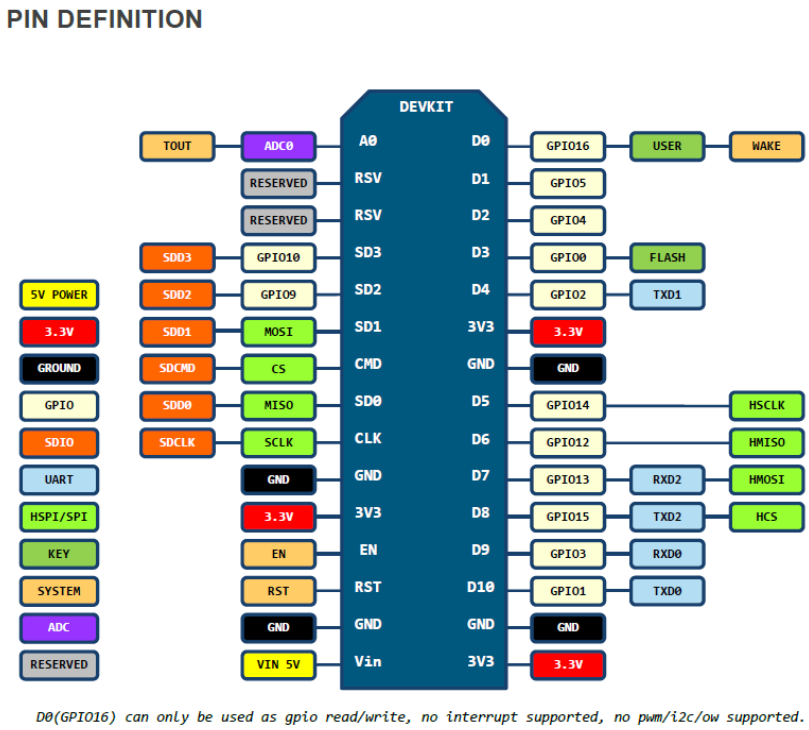
Com isso, chegou-se aos microcontroladores, que segundo Stallings (2017, p. 26) "[...] microcontrolador é um chip simples que contém processador, memória não volátil para o programa (ROM), memória volátil para entrada e saída (RAM), um *clock* e uma unidade de controle de E/S[9]". Usualmente, os microcontroladores menores e baratos são usados como processadores dedicados para efetuar uma tarefa específica em automações, por exemplo (STALLINGS, 2017, p. 27).

Com isso, para controlar os periféricos foi optado para uso neste trabalho o microcontrolador ESP32, desenvolvido pela empresa *Espressif Systems*[10]. Entre os motivos da escolha do microcontrolador estão o custo-benefício, pois têm módulos integrados de *Wi-Fi* e *Bluetooth*; seu processador possui dois *cores* que podem ser controlados individualmente; possibilidade de ajustar a frequência do *clock* da *Central Processing Unit* (CPU) entre 80 *Megahertz* (MHz) a 240 MHz; além de possuir um processador de baixo consumo de energia (ESPRESSIF SYSTEMS, 2021, p. 6).

Por ser bastante difundido, o ESP32 possui módulos de desenvolvimento e prototipagem, entre eles destaca-se o NodeMCU[11], que é um *hardware open source*, que foi construído primeiramente para o seu predecessor ESP8266 e posteriormente implementado para ESP32 (YUAN, 2017b). Na Figura 12 é exibida a pinagem do NodeMCU e a Figura 13 ilustra o NodeMCU ESP32.

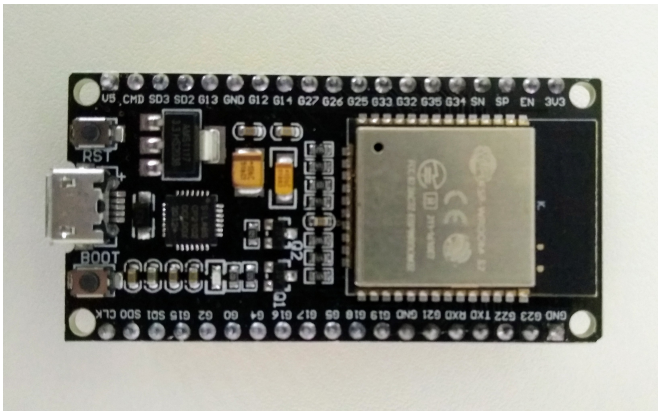
[9] Entrada/Saída
[10] https://espressif.com/
[11] https://www.nodemcu.com/

Figura 12 – Pinagem do NodeMCU



Fonte: NodeMCU (2015).

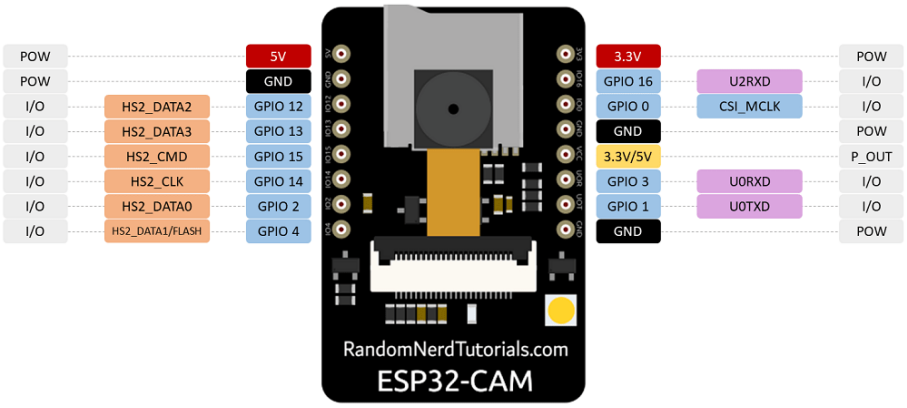
Figura 13 – NodeMCU ESP32



Fonte: Próprio autor (2021).

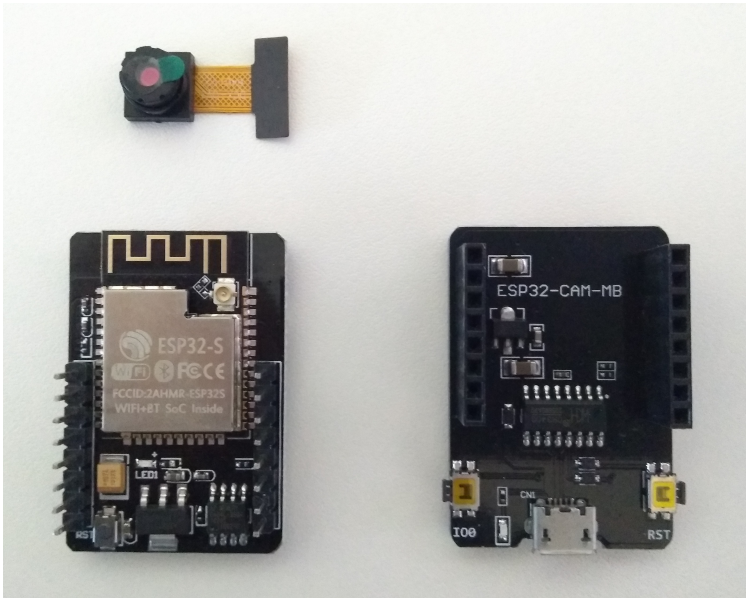
Outro módulo a ser destacado é o ESP32-CAM, que é baseado no ESP32-S e inclui uma câmera de 2M-pixel e um *slot* para cartão micro-SD. Dessa forma, as imagens podem ser armazenadas no micro-SD ou serem enviadas para uma *web page* ou ainda ser feito *streaming* para uma *web page* (CAMERON, 2021, p. 26). Na Figura 14 é ilustrada a disposição da pinagem do ESP32-CAM. Importante ressaltar que o módulo ESP32-CAM não possui conector via *Universal Serial Bus* (USB) (CAMERON, 2021, p. 27). Para ser energizado e atualizado o código fonte de forma prática, existe um módulo chamado ESP32-CAM-MB que permite a conexão via USB. A Figura 15 ilustra os módulos ESP32-CAM-MD, ESP32-CAM e a câmera OV2640, da direita para a esquerda, respectivamente.

Figura 14 – Pinagem do ESP32-CAM



Fonte: Santos & Santos (2021).

Figura 15 – ESP32-CAM, câmera OV2640 e módulo ESP32-CAM-MB



Fonte: Próprio autor (2021).

Como visto na Seção 2.2, os sensores têm papel fundamental na IoT, pois são eles que coletam os dados do ambiente, sendo que os dados coletados normalmente são categorizados em dois grupos: analógico e digital (SWANSON, 2000, p. 3). Para se comunicar com os periféricos o microcontrolador ESP32 possui pinos de entrada e saída, que foram exibidos nas Figuras 12 e 14, sendo alguns deles conversores analógico-digitais, digitais-analógico, UART para comunicação serial assíncrona, sensores de toque capacitivos que detectam variações de carga elétrica, entre outros (ESPRESSIF SYSTEMS, 2021, p. 31-32). Para mais informações acesse a documentação oficial[12].

[12] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/index.html

4.1.6 Transmissão dos dados

Após os dados serem coletados é preciso enviá-los para o servidor, para isso foi utilizada uma conexão sem fio (*wireless*) para facilitar a instalação dos sensores em um cenário real. Como visto na Seção 2.4, o *Wi-Fi* (IEEE 802.11) atualmente é o padrão mais difundido em residências, sendo vantajoso utilizá-lo para a comunicação entre os dispositivos e o servidor.

Pelo fato dos sensores não possuírem um processador ou alguma forma de comunicação de rede, é necessário conectá-los a algum dispositivo que tenha essa capacidade. Para isso foram utilizados microcontroladores já que são dispositivos pequenos e acessíveis financeiramente, com intuito de efetuar uma tarefa específica. Para este trabalho foram utilizados o NodeMCU ESP32 e o ESP32-CAM, pois ambos possuem *Wi-Fi* integrado e bom custo-benefício.

Para a comunicação dos sensores de movimento, temperatura e umidade, fumaça e gás foi utilizado o protocolo MQTT, que teve seu funcionamento detalhado na Seção 2.5. Para tornar a aplicação mais dinâmica, foram utilizados tópicos com dois níveis de hierarquia, sendo que o primeiro nível determina o cômodo e o segundo o assunto. Dessa forma, caso hajam duas ou mais cozinhas, por exemplo, é possível alterar o nome do primeiro nível do tópico para diferenciá-las. Por exemplo:

```
cozinha1/temperatura
cozinha2/temperatura
```

Cada sensor publica seus dados coletados em um determinado tópico, porém o sensor de temperatura e umidade, por coletar mais de uma informação (temperatura e umidade), publica em dois tópicos distintos. Os tópicos utilizados pelos sensores de temperatura e umidade, gás e fumaça e, movimento são, respectivamente, os seguintes:

```
cozinha/temperatura
cozinha/umidade
cozinha/gas_fumaca
entrada/movimento
```

4.1.7 *Firmwares*

Para coletar os dados oriundos dos sensores e transmiti-los ao servidor foram desenvolvidos dois *softwares* distintos, um para controlar os sensores utilizando um NodeMCU ESP32 e outro para disponibilizar o *streaming* de vídeo utilizando o microcontrolador ESP-CAM.

Para o desenvolvimento dos *firmwares* foi utilizada a *Integrated Development Environment* (IDE) Arduino IDE[13]. Nela é possível escrever o código, gerenciar as bibliotecas e placas,

[13] https://www.arduino.cc/en/software

compilar e atualizar o *software* dos dispositivos através do USB. As próximas duas subseções 4.1.7.1 e 4.1.7.2 irão detalhar, respectivamente, cada uma delas.

4.1.7.1 Sensores

Para a solução foi utilizado um microcontrolador NodeMCU ESP32 para controlar os sensores de temperatura e umidade (DHT11), movimento (PIR) e gás e fumaça (MQ-2). O esquema elétrico que conecta os sensores ao NodeMCU pode ser visualizado nas Figuras 40, 41 e 42. Além disso o microcontrolador conecta-se à rede local e publica os dados coletados no MQTT *broker*. Para os sensores DHT11 e MQ-2, o ESP32 coleta os dados a cada dois segundos e os publica no MQTT nos respectivos tópicos:

```
cozinha/temperatura
cozinha/umidade
cozinha/gas_fumaca
```

Para ler os dados do sensor DHT11 foi utilizada a biblioteca DHTStable[14] na versão 1.0.1, pois ela abstrai a complexidade da comunicação direta com o *hardware*. Para o sensor MQ-2 não foi preciso utilizar bibliotecas, pois é possível coletar o dado utilizando o comando `analogRead(MQ2_PINOUT)` que retorna a quantidade de PPM do ambiente.

Para o sensor de movimento PIR foi utilizada uma estratégia diferente para publicar as detecções de movimento no MQTT, já que um movimento pode ser detectado a qualquer momento. Diante disso, foram utilizadas interrupções, pois segundo Microcontrollerslab (2022) elas são usadas para lidar com eventos que não acontecem durante a execução sequencial de um programa. Ainda segundo o autor, interrupções externas ou interrupções via *hardware* são causadas por um módulo de *hardware* externo, quando o sensor de movimento PIR detectar movimento, por exemplo.

Porém, interrupções possuem limitações que acabam gerando a quebra na execução do programa. Segundo ESP8266 Arduino Core (2022) não é possível executar `delays()` ou `yield()`, assim como fazer operações bloqueantes e operações que desabilitam a interrupção já que as interrupções são muito restritivas no tempo de execução, podendo tomar apenas alguns microssegundos. Ao fazer uma publicação no MQTT dentro da função que a interrupção chama, a execução do programa era interrompida pela exceção. Diante disso, para resolver este problema foi utilizada uma variável booleana para representar a detecção ou não do movimento. Logo, toda vez que um movimento for detectado a interrupção atribui o valor `true` para esta variável (8) e há uma condição na função `loop()` que verifica se a variável tem o valor `true` (12), quando esta condição for verdadeira é publicado no MQTT a detecção de movimento (13)

[14] https://github.com/RobTillaart/DHTstable

e atribuído o valor `false` para a variável booleana (14). Após é preciso enviar o fim da detecção de movimento, para isso são aguardados cinco segundos após o envio da detecção de movimento (36) e publicado no MQTT o fim de detecção de movimento (21). O Algoritmo 1 demonstra de forma resumida a lógica do algoritmo em linguagem C.

Algoritmo 1 – Código para detectar movimento com interrupções e enviar via MQTT

```
1  #define PIR_PINOUT 27
2  #define MQTT_QOS 1
3  #define DELAY_MOV 5000
4  boolean movDetec, movDetecOff;
5  unsigned long lastMov = 0;
6
7  void IRAM_ATTR motionDetected() {
8    movDetec = true;
9  }
10
11 void motionDetectedSendMQTT() {
12   if(movDetec) {
13     mqttClient.publish("entrada/movimento", "on", true, MQTT_QOS);
14     movDetec = false;
15     movDetecOff = true;
16     lastMov = millis();
17   }
18 }
19
20 void sendMotionDetectionOff() {
21   mqtt.publish("entrada/movimento", "off", true, MQTT_QOS);
22   movDetecOff = false;
23 }
24
25 void setup() {
26   pinMode(PIR_PINOUT, INPUT_PULLUP);
27   attachInterrupt(digitalPinToInterrupt(PIR_PINOUT),
28                   motionDetected,
29                   RISING);
30   movDetec = false;
31 }
32
33 void loop() {
34   unsigned long now = millis();
35   if (movDetecOff && now - lastMov > DELAY_MOV)  {
36     sendMotionDetectionOff();
37   }
38   motionDetectedSendMQTT();
39 }
```

Fonte: Próprio autor (2022)

Para publicar os dados nos tópicos mencionados anteriormente, primeiramente é preciso ter acesso ao endereço do *broker*, que neste trabalho está hospedado na rede local do domicílio. A Seção 4.2.1 abordará o tema com mais detalhes. Para acessar a rede local foi utilizada a rede sem fio *WiFi*. Para fazer a conexão via *software* no microcontrolador foi utilizada a biblioteca *WiFi*[15].

Para publicar os dados nos tópicos desejados é preciso ter um cliente MQTT. Existem diversas bibliotecas que implementam um cliente MQTT, porém nem todas possibilitam definir níveis de QoS na publicação dos dados, como por exemplo a biblioteca *PubSubClient*[16] que permite publicar mensagens apenas com QoS 0 (KNOLLEARY, 2022). Diante disso, chegou-se à biblioteca *MQTTPubSubClient*[17] que permite publicar as mensagens com QoS 0, 1 e 2.

A QoS 1 foi escolhida pois notou-se a necessidade de possuir confiabilidade quanto a entrega do dado, pois caso for detectado um movimento é preciso ter certeza que esta informação chegue ao *Home Assistant*, porém não verificou-se a necessidade de ter confiabilidade quanto a entrega única do dado, pois na prática caso for entregue duas vezes o dado da detecção de movimento não fará diferença significativa para a aplicação.

Por questões de segurança da informação, para conectar-se ao *broker* é requisitado autenticação por meio de um usuário e senha. Este usuário e senha é definido no *broker*, que na Seção 4.2.1 será abordado com detalhes.

4.1.7.2 ESP-CAM

Para fazer o *streaming* de vídeo foi utilizada uma ESP-CAM. Para ser possível visualizar o vídeo da câmera foi desenvolvido um *firmware* que disponibiliza o *streaming* do vídeo em tempo real em um endereço de IP na rede local. Para disponibilizar o *streaming* na rede local, foi preciso utilizar a biblioteca *WiFi*[18] para fazer a conexão via *WiFi* com a rede local. Sendo que o vídeo não é gravado, apenas transmitido.

Para capturar as imagens oriundas da câmera a Espressif, fabricante da linha ESP, fornece uma biblioteca *open source* chamada `esp32_camera`[19] em que seu objetivo é capturar as imagens da câmera e convertê-las para o formato desejado. A biblioteca fornece exemplos de utilização, sendo que o desenvolvimento do *firmware* deste trabalho foi baseado no exemplo "JPEG HTTP Stream", onde os *frames* são transmitidos no formato JPEG.

Devido ao fato de que o *streaming* de vídeo será utilizado para detectar movimento, quanto maior a resolução e qualidade, maior o número de pixeis a imagem terá e consequentemente maior será o processamento necessário para detectar movimento pela imagem. Como

[15] https://www.arduino.cc/en/Reference/WiFi
[16] https://github.com/knolleary/pubsubclient
[17] https://github.com/hideakitai/MQTTPubSubClient/
[18] https://www.arduino.cc/en/Reference/WiFi
[19] https://github.com/espressif/esp32-camera

quem detecta o movimento é um Raspberry Pi, que além de detectar movimento deve rodar o *Home Assistant*, há um processamento limitado. Diante disso, segundo Santos & Santos (2022) a biblioteca utilizada permite definir uma resolução para o vídeo, sendo que as resoluções possíveis são:

- `FRAMESIZE_UXGA` (1600 x 1200);
- `FRAMESIZE_SXGA` (1280 x 1024);
- `FRAMESIZE_XGA` (1024 x 768);
- `FRAMESIZE_SVGA` (800 x 600);
- `FRAMESIZE_VGA` (640 x 480);
- `FRAMESIZE_CIF` (352 x 288);
- `FRAMESIZE_QVGA` (320 x 240).

Diante do processamento limitado e o fato que a imagem deve ser nítida a uma pessoa, a resolução escolhida foi 640 x 480. A Figura 16 mostra a qualidade da imagem transmitida pela ESP-CAM.

Figura 16 – Imagem do *streaming* de vídeo da ESP-CAM

Fonte: Próprio autor (2022).

O *Home Assistant* acessa o endereço de IP do *streaming* de vídeo, faz detecções de movimento e disponibiliza o vídeo. Para não haver problemas caso o roteador entregue IPs diferentes a cada conexão da ESP-CAM à rede, foi adicionado no *firmware* um IP fixo, dessa forma, toda vez que a ESP-CAM conectar-se à rede local, ela irá receber o mesmo endereço de IP. Outra forma de definir um IP fixo é pelo roteador, porém é necessário ter acesso a ele.

4.2 PROCESSAMENTO E ARMAZENAMENTO DOS DADOS

Ao longo deste estudo, mostrou-se necessário um servidor para receber, processar e armazenar os dados enviados pelos microcontroladores, também para permitir acesso, enviar as notificações para os usuários e interagir com outros sistemas. Diante dos casos de uso destacados na Seção 3.2 e dos trabalhos relacionados, chegou-se ao *Home Assistant*.

Segundo Williams (2018), *Home Assistant* é uma plataforma *open source* que pode ser usada gratuitamente. Ela iniciou como um *hobby*, tendo seu uso direcionado para o controle de lâmpadas Philips Hue[20], que foram lançadas em 2012. Atualmente provê uma plataforma para controle e automação residencial, sendo que o *Home Assistant* não é apenas uma aplicação, mas sim, um sistema embarcado sendo que as integrações, configurações e atualizações podem ser feitas através da interface gráfica (HOME ASSISTANT, 2021a). A companhia que está por trás do *Home Assistant* é a Nabu Casa[21], tendo seu foco na comunidade, sendo que sua receita é gerada por itens de valor-agregado, como a possibilidade de acessar o *Home Assistant* local de forma fácil em qualquer lugar do mundo (OVENS, 2020).

O *Home Assistant* armazena seus dados localmente, realizando a comunicação com os dispositivos de forma local e apenas obtém dados da nuvem caso não houver outra escolha (HOME ASSISTANT, 2021g). Segundo Marcin (2021), uma automação local como o *Home Assistant* tem diversos benefícios comparado a uma solução em nuvem, tais como: a velocidade, já que todo processamento ocorre na casa, sem haver a necessidade de enviar dados para *web-servers*, consequentemente não haverá disputa de banda pelos dispositivos; a compatibilidade, já que não terão proibições de companhias como a *Amazon* bloqueando dispositivos da *Google*, por exemplo; o controle, já que a nuvem limita os serviços aos seus clientes, ao ser usado um servidor local haverá controle total sobre os dispositivos e automações; a confiabilidade caso o serviço em nuvem caia ou perca acesso à *internet* na residência, não será possível acessar o *Home Assistant*.

Home Assistant (2021i) recomenda sua instalação por meio de *hardwares* dedicados, tais como *Raspberry Pi*, *ODROID*, *ASUS Tinkerboard*, entre outros. Também é possível instalá-lo utilizando *containers* ou máquinas virtuais, sendo que mais detalhes sobre cada instalação podem ser acessados na documentação oficial[22]. Para esse trabalho foi utilizado um *Raspberry Pi 3 Model B*[23] pelo fato de possuir os requisitos mínimos e pela disponibilidade do equipamento, sendo que suas especificações são processador *Advanced RISC Machine* (ARM) Quad Core 1.2 GHz; 1 *Giga Byte* (GB) de *Random-access Memory* (RAM); *Wireless* e *Bluetooth* integrados; porta *ethernet*; 40 pinos de *General Purpose Input/Output* (GPIO); 4 portas USB; *slot* para cartão micro-SD; porta HDMI; porta para *Raspberry Pi* câmera; e porta P2 para áu-

[20] https://www.philips-hue.com/pt-br
[21] https://www.nabucasa.com/
[22] https://home-assistant.io/installation/
[23] https://raspberrypi.org/products/raspberry-pi-3-model-b/

dio. Diferentemente dos microcontroladores mostrados na Seção 4.1.5, o *Raspberry Pi* precisa de um sistema operacional para funcionar, ele dever ser instalado em um cartão *micro-SD* e posteriormente inserido no *Raspberry Pi* (RASPBERRY PI, 2021). Por este motivo foi utilizado o *Home Assistant* com seu sistema operacional próprio.

Para instalá-lo, foi necessário um cartão de memória *MicroSD* de 32 GB e um computador com acesso à *internet*. Acessou-se o site oficial do *Home Assistant*, selecionado a instalação para *Raspberry Pi*, inserido o cartão de memória no computador e escrito a imagem do *Home Assistant* no cartão de memória. Após isso, foi inserido o cartão de memória com a imagem do *Home Assistant* no *Raspberry Pi*, que após ser ligado concluiu a instalação dentro de alguns minutos. Ao ser feito o primeiro acesso, é requisitada a criação do usuário administrador, através do qual é possível gerenciar e configurar o *Home Assistant*.

O acesso à interface do usuário se dá através de um navegador de *internet* por meio da *Uniform Resource Locator* (URL) `http://homeassistant.local:8123/`, sendo que a grande maioria das configurações podem ser feitas através da interface do usuário. Porém, também é possível acessar o *Home Assistant* por linha de comando de um computador qualquer que esteja na mesma rede local, utilizando *Secure Shell* (SSH). É importante ter este acesso via SSH pois caso alguma configuração fique errada o serviço do *Home Assistant* poderá ficar incessível via navegador.

O *Home Assistant* possui um banco de dados para gravar o histórico dos dados. Para interligar o *Home Assistant* com banco de dados, é utilizado o SQLAlchemy[24] que é um *Object Relational Mapper* (ORM) que possibilita o uso de alguns bancos de dados, tais como MariaDB[25], MySQL[26], PostgreSQL[27]e SQLite[28]. Por padrão o banco de dados utilizado é o SQLite (HOME ASSISTANT, 2022c). No decorrer deste estudo não notou-se nenhum motivo para mudar o banco de dados, com isso o banco de dados utilizado no *Home Assistant* para essa solução foi o SQLite.

O *Home Assistant* tem a capacidade de gravar em seu banco de dados tudo o que está acontecendo e permite ao usuário navegar pelos dados. Por padrão esta funcionalidade está habilitada, porém é possível desabilitá-la para o histórico não ser salvo. Também é possível desabilitar o armazenamento dos dados de entidades específicas. Todavia, para este trabalho todos os dados recebidos são armazenados. O *Home Assistant* grava o dado recebido apenas se ele for diferente do último recebido. Por exemplo, se for recebido um dado de temperatura de 20ºC e o próximo dado recebido também for 20ºC o segundo dados não será armazenado no banco de dados.

O *Home Assistant* também tem a capacidade de expurgar dados a partir de um deter-

[24] https://www.sqlalchemy.org/
[25] https://mariadb.org/
[26] https://www.mysql.com/
[27] https://www.postgresql.org/
[28] https://www.sqlite.org/

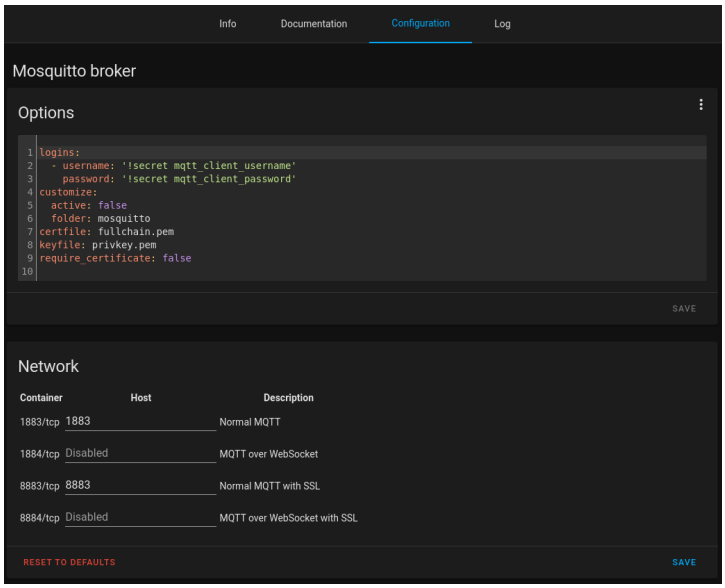
minado período de tempo. Segundo Home Assistant (2022c) essa opção vem habilitada por padrão e esta limpeza é executada toda noite às 4:12 no horário local. Limpar os dados evita que o banco de dados cresça indevidamente, o que ocupa espaço em disco e pode deixar o *Home Assistant* lento. Ainda segundo o autor, se o `auto_purge` for desabilitado é recomentado criar uma automação para limpar os dados periodicamente. Diante disso, para este trabalho foram mantidas as definições padrões da limpeza do banco de dados, mantendo o histórico dos últimos 10 dias.

4.2.1 MQTT *Broker*

Como mencionado na Seção 4.1.6, os sensores fazem a comunicação utilizando o protocolo MQTT, seu funcionamento foi detalhado na Seção 2.5, e como visto ele desacopla o enviador do receptor da mensagem, utilizando um intermediário chamado de *broker*. O *broker* deve estar sendo executado em algum computador em que tanto o enviador quanto o receptor consigam acessá-lo.

O *Home Assistant* possui um *add-on* para um MQTT *broker* chamado *Eclipse Mosquitto*[29], que é um projeto *open source* mantido pela *Eclipse Fundation*[30]. Por esse motivo foi utilizado neste trabalho o *Eclipse Mosquitto* como *broker*, pois caso contrário seria necessária a criação de outro *add-on* para o *broker* desejado. Sua instalação foi feita através da loja de *add-ons* do *Home Assistant*. Após a instalação foi preciso configurar o *broker* na aba *Configuration*. As configurações são feitas através da interface do *Home Assistant* em arquivos `.yaml`. A Figura 17 ilustra a interface de configuração.

Figura 17 – Configuração do MQTT *Broker* pela interface do *Home Assistant*



Fonte: Próprio autor (2022).

[29] https://mosquitto.org/
[30] https://eclipse.org/

Foram mantidas as portas padrão do serviço 1883 para conexão sem criptografia e 8883 para conexão utilizando *Secure Sockets Layer* (SSL). Porém, foram desabilitadas as portas 1884 e 8884 para conexão via *websockets*, já que não foram utilizadas neste trabalho. O endereço do *broker* é o próprio IP do *Raspberry Pi* na rede local na porta 1883 ou 8883. Por questões de segurança foi criado um usuário e senha para publicar os dados coletados pelo microcontrolador. Para os dados sensíveis, *username* e *password*, foram utilizadas *secrets*, dessa forma os dados sensíveis não são enviados para o repositório *git* e ficam centralizados em um único local, que é o arquivo de configuração `secrets.yaml`.

4.2.2 Recebimento dos dados dos sensores

O *Home Assistant* deve receber os dados enviados pelos sensores via MQTT, para isso o *Home Assistant* precisa de um cliente MQTT. O *Home Assistant* não implementa um cliente MQTT em sua configuração padrão, por isso foi preciso adicionar uma integração com um cliente MQTT através do menu integrações. Por meio deste cliente MQTT o *Home Assistant* assina os tópicos que o microcontrolador publica dos dados coletados, sendo que os tópicos foram definidos no arquivo de configuração `configuration.yaml`. A Figura 18 mostra como o arquivo foi configurado.

Figura 18 – Arquivo de configuração do *Home Assistant* com os tópicos assinados

```
33  # Sensores
34  sensor:
35    - name: sensor_cozinha_temperature
36      platform: mqtt
37      state_topic: "cozinha/temperatura"
38      qos: 1
39
40    - name: sensor_cozinha_umidade
41      platform: mqtt
42      state_topic: "cozinha/umidade"
43      qos: 1
44
45    - name: sensor_cozinha_gasfumaca
46      platform: mqtt
47      state_topic: "cozinha/gas_fumaca"
48      qos: 1
49
50    - name: sensor_entrada_movimento
51      platform: mqtt
52      state_topic: "entrada/movimento"
53      qos: 1
54
55  binary_sensor:
56    - platform: mqtt
57      name: Sensor movimento entrada
58      state_topic: "entrada/movimento"
59      payload_on: "on"
60      payload_off: "off"
61      device_class: motion
```

Fonte: Próprio autor (2022).

O *Home Assistant* interpreta cada um destes tópicos como uma entidade, com isso é possível adicioná-los à interface do usuário para mostrar o último dado recebido, bem como o histórico dos dados recebidos. Os dados que são recebidos pelo *Home Assistant* são gravados em seu banco de dados, sendo que o próprio *Home Assistant* faz o armazenamento e gerenciamento do banco de dados.

Para os tópicos de temperatura, umidade e gás/fumaça foram definidos quatro campos: `name` é o identificador único que será concedido à entidade; `platform` define por qual meio o dado será recebido pelo *Home Assistant*; `state_topic` define qual o tópico que o *Home Assistant* ficará monitorando; e `qos` define o nível de QoS na assinatura do tópico.

No sensor de movimento, por ser um sensor binário que deve mostrar se há ou não movimento, foram feitas algumas configurações a mais em comparação aos outros sensores. Foi necessário parametrizar os campos `payload_on` e `payload_off` com os valores equivalentes à verificação de movimento e não verificação de movimento que foram enviados pelo microcontrolador, que como demonstrado na Seção 4.1.7.1, foram o `on` e `off`, respectivamente. Também foi definido o campo `device_class` que define o ícone, cor e *label* conforme o sensor binário esteja ativo ou não (HOME ASSISTANT, 2022a).

4.2.3 Detecção de movimento pela câmera

Para detectar movimento pelo *streaming* oriundo da ESP-CAM foi utilizado o *add-on MotionEye*[31] que está disponível na loja de *add-ons* do *Home Assistant*. Ele é um *frontend open source* usado para a gerência e visualização de câmeras, sendo possível adicionar várias câmeras com diferentes configurações cada, detectar movimento, salvar as imagens das detecções e executar comandos (NIJHOF, 2018).

Para o *MotionEye* exibir o *streaming* de vídeo e fazer as detecções de movimento pelo vídeo foi configurado o endereço IP da câmera na rede local. Como mencionado na Seção 4.1.7.2 foi definido um IP fixo para que não seja pego um IP aleatório a cada nova conexão à rede local.

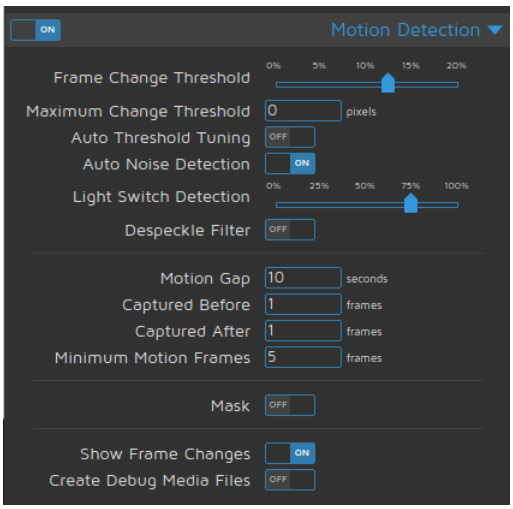
Existe uma série de configurações para definir como a detecção de movimento será feita, entre elas estão: ***Frame Change Threshold*** indica a porcentagem mínima de mudança entre dois *frames* sucessivos; ***Maximum Change Threshold*** define o número de *pixeis* alterados entre *frames* acima dos quais o movimento não é mais acionado, sendo que zero desabilita o limite; ***Auto Noise Detection*** habilita a detecção automática de ruído; ***Light Switch Detection*** define a porcentagem que a imagem precisa alterar para que o evento seja tratado como uma mudança de luz repentina ao invés de um movimento; ***Despeckle Filter*** habilita a aplicação do filtro de *despeckle*[32] antes da detecção de movimento; ***Motion Gap*** define a quantidade de segundos sem movimento para marcar o fim da detecção de movimento; ***Captured Before*** define o número de *frames* a serem capturados antes da detecção de movimento; ***Captured After*** define o número de *frames* a serem capturados depois do fim da detecção de movimento; e ***Minimum Motion Frames*** define o número mínimo de *frames* sucessivos necessários para começar a detecção de movimento. A Figura 19 ilustra como as configurações são dispostas na interface e como elas

[31] https://github.com/hassio-addons/addon-motioneye

[32] *Despeckle* é um filtro utilizado para remover pequenos defeitos em imagens. O filtro altera cada *pixel* da imagem com o valor da mediana calculada em um determinado range de *pixeis* (GIMP, 2022).

foram configuradas para este trabalho.

Figura 19 – Configuração da detecção de movimento do *MotionEye*



Fonte: Próprio autor (2022).

O *MotionEye* permite executar comandos no início e fim de uma detecção de movimento, dessa forma é possível enviar comandos para o *Home Assistant* para indicar o início e fim da detecção. Então, foram criados dois *webhooks*[33], um para sinalizar o início da detecção de movimento e outro para sinalizar o fim da detecção. Uma forma de criar *webhooks* no *Home Assistant* é por meio das automações. A Figura 20 ilustra como as duas automações foram criadas.

Figura 20 – Configuração das automações para detecção de movimento pela câmera

```
- id: "1650133608109"
  alias: Movimento detectado pela câmera - Início
  description: ""
  trigger:
    - platform: webhook
      webhook_id: motioneye-camera1-started
  condition: []
  action:
    - service: input_boolean.turn_on
      data:
        entity_id: input_boolean.camera1_motion_detected
    - service: script.notificacao_movimento_pela_camera
      data: {}
  mode: single

- id: "1650133636349"
  alias: Movimento detectado pela câmera - Fim
  description: ""
  trigger:
    - platform: webhook
      webhook_id: motioneye-camera1-ended
  condition: []
  action:
    - service: input_boolean.turn_off
      data:
        entity_id: input_boolean.camera1_motion_detected
  mode: single
```

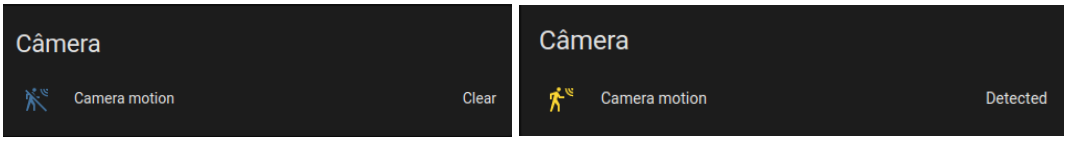
Fonte: Próprio autor (2022).

[33] *Webhook* permite a interação entre aplicações através do uso de *callbacks* (retornos de chamada) personalizados. Seu uso permite que as aplicações se comuniquem de forma automática (MIR, 2021).

Sendo que a *tag* `trigger` define o gatilho para uma determinada ação acontecer, dentro dela é preciso informar qual a plataforma (`platform`), que neste caso será `webhook` e qual é o identificador único desse *webhook* (`webhook_id`), sendo que o `webhook_id` será utilizado para a criação da URL de acionamento do *webhook*. Depois, é preciso informar as ações (`action`) a serem executadas, dentro da *tag* `action` deve ser definido o serviço (`service`) que será chamado e definido os dados (`data`) que serão enviados para o serviço.

Para o início da detecção de movimento foram definidas duas ações, uma para definir o *card* com *Detected* (movimento detectado), ilustrado na Figura 21b e a outra para enviar uma notificação para o *smartphone* dos residentes informando a detecção de movimento, que será detalhada na Seção 4.2.4. Para o fim da detecção foi designada uma ação para definir o *card* como *Clear* (sem movimento), como ilustra a Figura 21a.

Figura 21 – *Card* que exibe se há ou não movimento detectado pela câmera



(a) Sem movimento (b) Com movimento

Fonte: Próprio autor (2022).

Então, quando for detectado movimento pelo *MotionEye*, é enviada uma requisição *Hypertext Transfer Protocol* (HTTP) POST utilizando cURL[34] para o *webhook* de início de detecção de movimento. Ao final da detecção será enviada outra requisição HTTP POST, porém dessa vez para o *webhook* fim de detecção. Os comandos utilizados para fazer a requisição foram respectivamente os seguintes:

```
curl -XPOST homeassistant.local:8123/api/webhook/motioneye-camera1-started
```

```
curl -XPOST homeassistant.local:8123/api/webhook/motioneye-camera1-ended
```

Além disso, a cada detecção de movimento são salvos alguns *frames* do vídeo. Estas imagens serão armazenadas em pastas separadas por câmera e o nome das imagens é composto pela data e hora da detecção de movimento. Estas imagens são armazenadas no cartão de memória micro-SD, todavia, existe a possibilidade de alterar o local onde as imagens são armazenadas. Dessa forma é possível adicionar dispositivos externos para armazená-las.

[34] cURL é uma ferramenta de linha de comando utilizada por desenvolvedores para transferir dados de ou para um servidor (SALADAS, 2019), sendo uma ferramenta *open source* com suporte a diversos protocolos (CRUL, 2022).

4.2.4 Notificações

O *Home Assistant* por meio de seu aplicativo para dispositivos móveis permite o envio de notificações para os *smartphones* que possuam o seu aplicativo instalado e que estejam autenticados em suas devidas contas. Dessa forma, é possível receber notificações pelo *smartphone*. Para enviar as notificações é possível criar automações com determinadas regras de acionamento e definir quais dispositivos irão recebê-las.

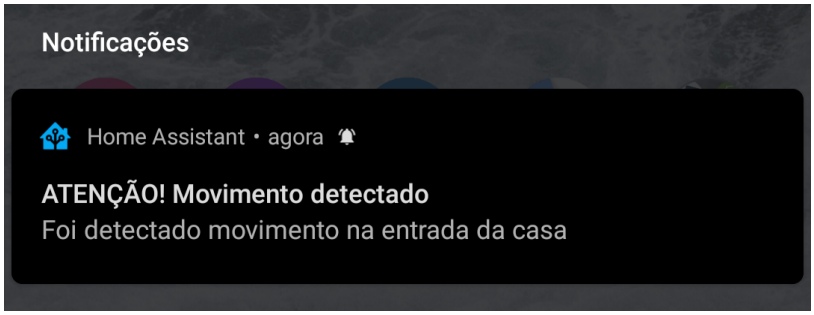
As notificações podem ser parametrizadas por diversos atributos tais como título, mensagem, anexar mídia (foto, vídeo e áudio), abrir uma URL ao clicar sobre a notificação o que permite tanto abrir um *dashboard* específico no aplicativo do *Home Assistant* quanto abrir um *site* pelo navegador, definir níveis de prioridade da notificação, adicionar botões de comando, entre diversas outras opções (HOME ASSISTANT, 2021l).

O aplicativo do *Home Assistant* dispõe de versões para *Android* e *iOS*, que por serem sistemas operacionais distintos possuem algumas funcionalidades específicas no recebimento de notificações, para saber mais sobre cada especificidade é necessário consultar a documentação oficial[35]. Para este trabalho foram enviadas notificações apenas para dispositivos *Android*, já que não houve acesso a um *smartphone* com *iOS*.

Para notificar os residentes por meio de seus *smartphones* foram criadas três automações no *Home Assistant*:

1. **Detecção de movimento pelo sensor PIR**: Quando o *Home Assistant* receber através do MQTT o *payload* `on` no tópico `entrada/movimento`, quer dizer que foi detectado movimento pelo sensor, com isso é disparada uma notificação para os residentes alertando o ocorrido. A Figura 22 ilustra a notificação.

Figura 22 – Notificação de detecção de movimento pelo sensor PIR



Fonte: Próprio autor (2022).

2. **Detecção de movimento pela câmera**: Quando o *Home Assistant* receber o comando através do *webhook* de início de movimento, descrito na Seção 4.2.3, é disparada uma notificação para os residentes alertando-os. A Figura 23 ilustra a notificação.

[35] https://companion.home-assistant.io/docs/notifications/notifications-basic

Figura 23 – Notificação de detecção de movimento pela câmera

Fonte: Próprio autor (2022).

3. **Detecção de gás ou fumaça**: Quando for recebido um valor maior que 1.500 através do MQTT pelo tópico `cozinha/gas_fumaca` é disparada uma notificação para os *smartphones* dos residentes alertando sobre o possível vazamento de gás ou fumaça. O valor 1.500 foi definido diante do estudo feito na Seção 4.1.1 que define que o IDLH é de 2.000 PPM, porém para ser possível executar uma contramedida antes de algum perigo ou dano à vida foi definido um valor abaixo dos 2.000 PPM. A notificação tem um botão para chamar os bombeiros, que ao ser clicado redireciona o usuário ao aplicativo padrão de telefone. A Figura 24 ilustra a notificação.

Figura 24 – Notificação de detecção de gás e fumaça

Fonte: Próprio autor (2022).

Foi definida a permanência das notificações até que o usuário clique sobre a mesma, que faz o aplicativo do *Home Assistant* ser aberto. Também foi definido o nível mais alto de prioridade (*high*), pois com isso, faz com que a notificação seja disparada no *smartphone* mesmo que ele esteja ocioso.

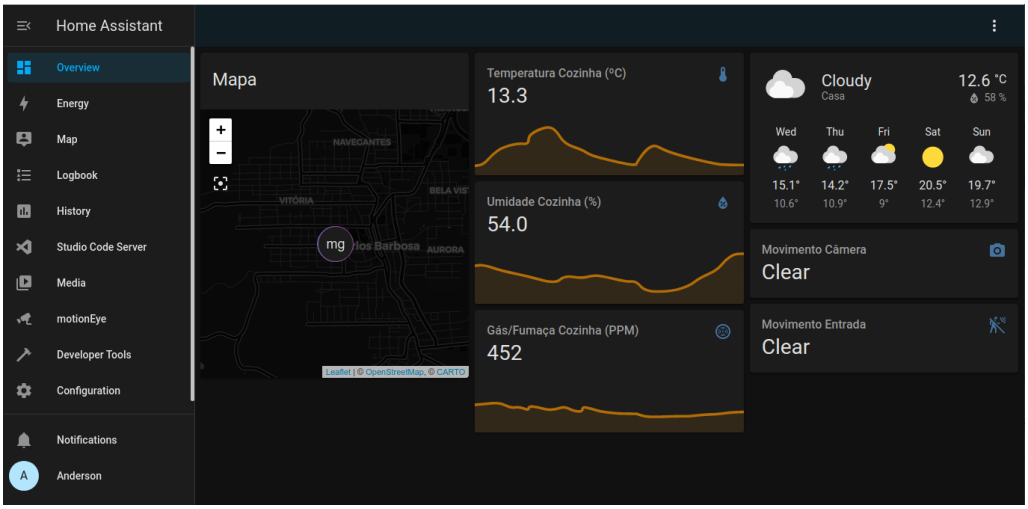
4.3 ACESSO AOS DADOS

O *Home Assistant* provê uma interface gráfica baseada em *dashboards* denominada *Lovelace*. Segundo Home Assistant (2021k), ela é rápida, customizável e é uma poderosa forma para o usuário gerenciar sua casa utilizando seu *smartphone* ou *desktop*. É possível definir diversos *dashboards*, sendo que eles são gerenciados pela interface do usuário, porém também

é possível modificá-los através dos arquivos `.yaml`. *Lovelace* possui 29 diferentes *cards*, sendo que é possível desenvolver novos *cards* customizados[36].

Para este trabalho foi criado um *dashboard* principal para mostrar os dados coletados dos sensores, bem como gráficos que exibem o histórico diário dos dados oriundos dos sensores de temperatura e umidade, e gás e fumaça. Também foram adicionados dois *cards* para exibir a detecção de movimento pela câmera e pelo sensor de movimento PIR. A Figura 25 ilustra o *dashboard* desenvolvido.

Figura 25 – *Dashboard* principal do *Home Assistant*



Fonte: Próprio autor (2022).

O *Home Assistant* possui por padrão uma tela para mostrar o histórico dos dados em que é possível filtrar o histórico por período e por entidade. A Figura 26 mostra como o histórico é apresentado para o usuário.

Figura 26 – Tela de histórico dos dados no *Home Assistant*

Fonte: Próprio autor (2022).

Como apresentado na Seção 4.2.3 foi utilizado o *add-on* do *Home Assistant MotionEye* para detectar movimento pela câmera, porém além de detectar movimento ele permite visualizar o *streaming* de vídeo. Dessa forma ao abri-lo, pelo menu lateral, é exibido o vídeo como mostra a Figura 27.

36 https://developers.home-assistant.io/docs/frontend/custom-ui/custom-card/

Figura 27 – Exibição do *streaming* de vídeo pelo *MotionEye*

Fonte: Próprio autor (2022).

Diante do fato que o *Home Assistant* está em uma rede local, não é possível acessá-lo de forma externa, todavia mostrou-se necessário o acesso remoto já que diante de um vazamento de gás, por exemplo, o residente deve ser notificado mesmo não estando em casa. Diante disso, para solucionar este problema existem algumas formas para viabilizar o acesso externo, sendo algumas delas:

1. **Nabu Casa**: Possui um recurso chamado *Remote UI* que permite conectar de forma remota à instância local do *Home Assistant*. Ele funciona criptografando todas as comunicações entre o navegador e a instância local, sendo que a criptografia é provida pelo certificado *Let's Encrypt*. De modo geral, a instância local do *Home Assistant* conecta-se a um servidor *proxy*, que opera sobre a camada TCP e encaminha todos os dados criptografados para à instância local (NABU CASA, 2021). De forma sucinta, a *Nabu Casa* gera uma URL que provê acesso à instância local. Este serviço em maio de 2022 possui um valor mensal de 6,50 dólares.

2. **ngrok**: Permite expor à *internet* um *webserver* rodando em sua máquina local informando qual porta o *webserver* está escutando (NGROK, 2021). O *ngrok* gera uma URL para o usuário externo acessar, dependendo do tipo da conta contratada pode ser gerada uma URL aleatória ou fixa.

3. ***Dynamic* DNS (DDNS)**: Outra forma de expor a instância local é utilizando DDNS, para isso é necessário um *host*, que deve receber o IP público da instância local fornecida pelo provedor. Pelo fato do endereço IP mudar com frequência, é preciso de um *software* para atualizar o endereço de IP no *host* de DDNS (N-ABLE, 2019). Dessa forma, possibilita o acesso à instância local sempre pela mesma URL. Alguns *hosts* são Duck DNS e No-Ip. Esta forma necessita do acesso ao roteador da rede local, pois é preciso criar uma regra de *port forwarding*[37] para o endereço de IP do *Rasberry Pi* na rede local.

[37] *Port forwarding*, ou redirecionamento de porta em português, é a forma de fazer com que um computador fique acessível a outros computadores por meio da *internet*, mesmo que ele esteja por trás de um roteador ou *firewall* (PORT FORWARD, 2017).

Ao ser analisado qual das três soluções melhor se adéqua ao trabalho viu-se que para implementar o DDNS é necessário acesso ao roteador para fazer redirecionamento de porta, porém nem todos os roteadores possuem essa funcionalidade e nem todos os provedores de *internet* fornecem acesso aos roteadores em comodato. O ngrok em sua versão gratuita gera uma URL aleatória cada vez que o sistema é iniciado, não sendo o ideal já que seria preciso alterar a URL de acesso a cada reiniciamento do sistema. Já na solução da Nabu Casa não é preciso acesso ao roteador, a URL é fixa e os dados são criptografados, sendo que quem faz todo o trabalho é a própria Nabu Casa. Além disso é fornecido um teste grátis de 31 dias. Diante disso, para este trabalho foi utilizado o serviço *Remove UI* da Nabu Casa.

Para a Nabu Casa fornecer acesso externo ao *Home Assistant* foi necessário criar uma conta e associa-la à instância do *Home Assistant*. Após ser feita a associação, é gerada uma URL fornecida pela Nabu Casa para acessar a instância local de forma remota através da *internet*.

4.4 MODO DE CONTINGÊNCIA

Pelo fato do *Home Assistant* utilizar um servidor local, não será possível acessá-lo caso falte energia ou *internet*. Diante disso, para os residentes serem notificados quando a instância do *Home Assistant* está inacessível e terem as informações sobre o que pode ter ocasionado o problema, é necessário que eles tenham acesso aos últimos dados coletados pelos sensores, bem como as imagens das últimas detecções de movimento.

Para isso, viu-se necessária a utilização de um banco de dados na nuvem, em que são armazenados os dados coletados dos sensores e as imagens das detecções de movimento a cada determinado período de tempo. Por exemplo, a cada cinco minuto são enviados os últimos dados coletados dos sensores para o banco de dados em nuvem. O *Home Assistant* é responsável por conectar-se ao banco de dados na nuvem e inserir os dados periodicamente.

Também, viu-se necessária a criação de um aplicativo móvel que deve buscar os dados armazenados no banco de dados em nuvem e disponibilizá-los para análise do usuário e também fazer requisições com recorrência em segundo plano para a *Application Programming Interface* (API) do *Home Assistant*, em que o intuito é verificar se a instância está com acesso externo. Caso não haja, é disparada uma notificação para o dispositivo em que o aplicativo está instalado, com o objetivo de avisar o residente que o *Home Assistant* está sem acesso remoto. Dessa forma, o residente pode analisar os últimos dados coletados e se for necessário executar uma contramedida. Nas próximas Seções 4.4.1 e 4.4.2 será detalhada a replicação dos dados para a nuvem e o desenvolvimento do aplicativo móvel.

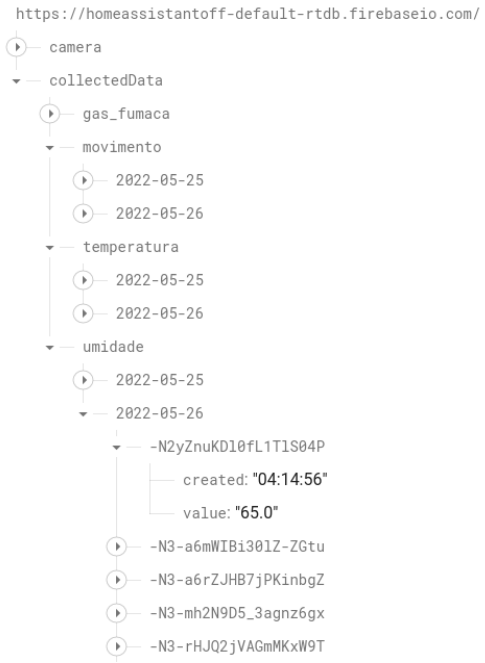
4.4.1 Replicação dos dados para a nuvem

Pelo fato do *Raspberry* armazenar tipos distintos de dados provenientes de sensores e câmeras, notou-se a demanda da utilização de bancos de dados distintos para armazenar as ima-

gens dos dados dos sensores. Para armazenar as imagens foi utilizado o *Storage*[38], que é uma solução do *Firebase*[39] específica para armazenamento de arquivos, em que para cada arquivo salvo é gerada uma URL específica que pode ser acessada pela *internet*. Já para armazenar os dados provenientes dos sensores e as URLs das imagens salvas no *Storage*, foi utilizado o banco de dados *NoSQL*[40] *Realtime Database*[41] que é uma solução do *Firebase* para armazenar e sincronizar dados na nuvem em tempo real, que permanecem disponíveis quando o aplicativo está *offline* (FIREBASE, 2021). Por ser um banco de dados *NoSQL* os dados podem ser armazenados em hierarquias no formato *json*.

Diante do fato que cada sensor é independente, os dados podem ser coletados de formas diferentes. Diante disso, a separação dos dados no *Realtime Database* foi por sensor, em que cada sensor tem seus dados armazenados de forma independente, tendo em comum o nodo pai `collectedData`. Os dados para cada sensor possuem o mesmo padrão, que armazena o valor (*value*) e a hora da coleta (*created*). Diante do fato que os dados dos sensores de temperatura, umidade e gás/fumaça são coletados a cada dois segundos pelos sensores, haverá uma quantidade razoável de dados. Diante disso, viu-se a necessidade de separar os dados por dia, para que no aplicativo seja possível filtrar os dados pela data. A Figura 28 ilustra como os dados estão dispostos no banco de dados *Realtime Database*.

Figura 28 – Disposição dos dados no *Realtime Database*



Fonte: Próprio autor (2022).

[38] https://firebase.google.com/products/storage?hl=pt-br

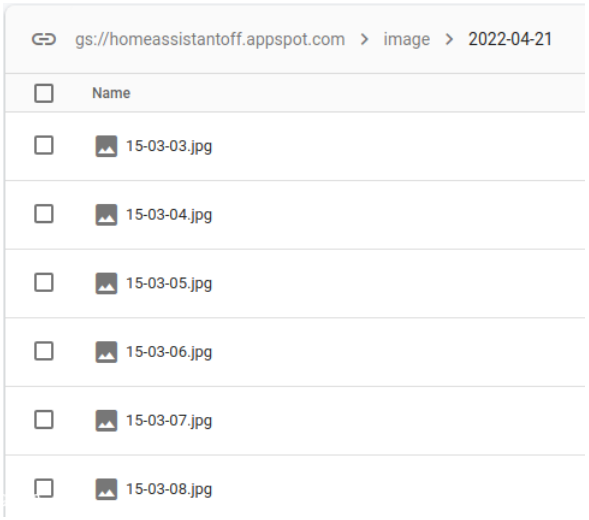
[39] https://firebase.google.com/?hl=pt-br

[40] *NoSQL* refere-se a bancos de dados não relacionais, em que os dados são armazenados em um formato diferente das tabelas relacionais. São utilizados em *big data* e aplicativos *web* em tempo real, já que suas principais vantagens são a alta escalabilidade e alta disponibilidade (ORACLE, 2022).

[41] https://firebase.google.com/products/realtime-database

As imagens das detecções de movimento são salvas no banco de dados *Storage* e são separadas por dia, com isso para cada dia é criada uma nova pasta. As imagens são salvas com o padrão hora-minuto-segundo em seus nomes. A Figura 29 ilustra a disposição das imagens no *Storage*.

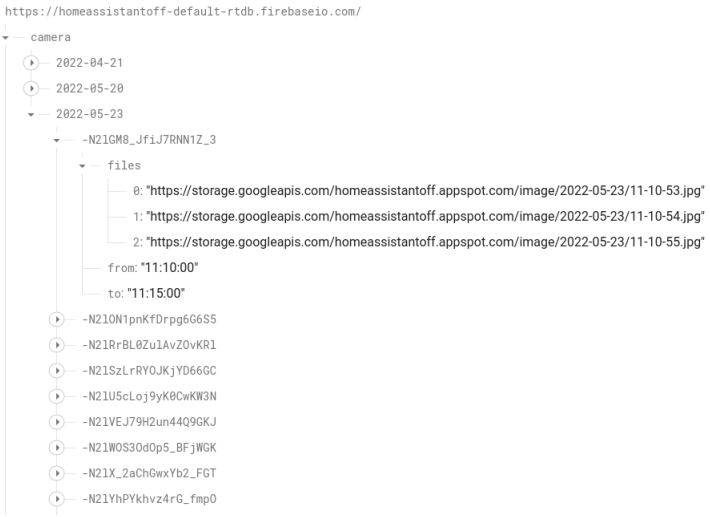
Figura 29 – Disposição das imagens armazenadas no *Storage*



Fonte: Próprio autor (2022).

Para disponibilizar as imagens no aplicativo, que será detalhado na Seção 4.4.2, viu-se necessário armazenar as URLs das imagens no banco de dados *Realtime Database*, para ser possível filtrá-las por data e hora. Os dados foram salvos por dia, sendo que a cada envio é criado um nodo que armazena as URLs das imagens e o período de datas da detecção de movimento que o nodo representa. A Figura 30 ilustra como os dados das imagens de detecção de movimento estão dispostos no *Realtime Database*.

Figura 30 – URL das imagens no *Realtime Database*



Fonte: Próprio autor (2022).

O *Home Assistant* não possui nenhuma integração com o *Firebase*, todavia ele permite criar integrações customizadas. Com isso, para replicar os dados dos sensores e as imagens das detecções de movimento foi desenvolvida uma integração na linguagem *Python* para enviar os dados do *Home Assistant* para o *Firebase*. Este *software* foi desenvolvido utilizando o *Visual Studio Code*[42], acessando o *Home Assistant* através de uma conexão Samba. Para criar a integração customizada no *Home Assistant* foi preciso criar dois arquivos:

1. **Manifesto de integração**: Segundo Home Assistant (2022b) toda integração deve ter um arquivo de manifesto para especificar informações básicas sobre a integração. Este arquivo é armazenado com a nomenclatura `manifest.json` dentro do diretório da integração. Nele devem ser especificadas algumas informações, tais como: *domain*, que consiste em um nome único para distinguir a integração; *name* é o nome da integração; *version*, campo para versionamento da integração; *dependencies* é utilizado para o *Home Assistant* instalar dependências que a integração necessita; *codeowner* é utilizado para identificar o desenvolvedor da integração; e *iot_class* define como a integração faz suas conexões. A Figura 31 ilustra como o arquivo `manifest.json` foi definido para este trabalho.

Figura 31 – Arquivo de manifesto da integração

```
{
    "domain": "ha2firebase",
    "name": "Home Assistant To Firebase",
    "codeowners": ["@aimperatori"],
    "requirements": ["firebase_admin==5.2.0"],
    "iot_class": "cloud_push",
    "version": "1.0.1"
}
```

Fonte: Próprio autor (2022).

   Para enviar os dados ao *Firebase* é necessário utilizar a biblioteca `firebase_admin`. O *Home Assistant* utiliza o gerenciador de pacotes *pip*[43] para instalar suas dependências. Então, pelo fato da biblioteca do *Firebase* estar disponível via *pip*[44] foi possível adicionar a biblioteca definindo-a no *requirements*, como mostra a Figura 31.

2. **Arquivo** `__init__.py`: Nele é que deve ser desenvolvido o código fonte da integração. O código deve ser desenvolvido na linguagem *Python* e deve conter um método chamado `setup` que é chamado quando o *Home Assistant* é inicializado. Neste método podem ser registrados serviços que executam um método definido, sendo que estes serviços podem ser chamados por automações.

Para o desenvolvimento da integração foram registrados dois serviços para replicar os dados no *Firebase*, um para enviar os dados dos sensores (temperatura, umidade, gás/fumaça

[42] https://code.visualstudio.com/
[43] https://pypi.org/
[44] https://pypi.org/project/firebase-admin/

e movimento) e outro para enviar as imagens das detecções de movimento. As próximas duas Seções 4.4.1.1 e 4.4.1.2 irão detalhar o funcionamento de cada uma delas.

4.4.1.1 Dados dos sensores

Como visto na Seção 4.2 o *Home Assistant* armazena seus dados em um banco de dados local que neste trabalho foi utilizado o *SQLite*, com isso para enviar os dados à nuvem primeiro é preciso acessar os dados dos sensores no *Home Assistant*.

Para acessar os dados no *SQLite* foi utilizada a biblioteca *sqlite3* que pelo fato do *Home Assistant* utilizá-la, não foi necessária sua inserção nos *requirements* do arquivo de manifesto. Para obter os dados foram desenvolvidas quatro consultas para coletar os dados de temperatura, umidade, gás/fumaça e movimento dos últimos cinco minutos do banco de dados do *Home Assistant*. Para a consulta dos dados de gás/fumaça foi definida uma regra para serem replicados os dados que tiverem um valor maior que 1000, com isso são enviados à nuvem apenas os dados que sejam relevantes para alguma possível ameaça à residência.

Para cada dado que retorna do banco de dados é feita sua transformação em *json* e enviado para o *Realtime Database* em seu respectivo dia. Para os quatro tipos de dados coletados foi definido um padrão para o *json* que é composto por dois campos: `value` contém o dado coletado pelo sensor e `created` contém a hora da inserção do dado no banco de dados do *Home Assistant*. O exemplo a seguir mostra um *json* de temperatura:

```
{
    "value": "14.3",
    "created": "14:13:56"
}
```

Importante ressaltar que o *Home Assistant* armazena as datas e horas com o fuso horário UTC+0, porém no Brasil o horário oficial é UTC-3, com isso ao ser buscado as datas no banco de dados do *Home Assistant* é feita a conversão de horário para UTC-3.

Para executar o serviço criado, foi utilizada uma automação que possui recorrência de cinco minutos na sua execução. Dessa forma, o *Home Assistant* dispara esta automação de forma automática que por sua vez chama o serviço da integração criada para enviar os dados coletados dos últimos cinco minutos.

4.4.1.2 Imagens das detecções de movimento

Como visto na Seção 4.2.3, o *MotionEye* armazena as imagens das detecções de movimento em um determinado diretório separadas por dia, elas são salvas com o padrão na nomenclatura hora-minuto-segundo.

Para obter estas imagens foi desenvolvido um código que cria o padrão de nomenclatura da hora da execução do serviço até cinco minutos atrás, para verificar se há uma imagem na pasta do dia atual, com nome condizente. Cada imagem obtida é enviada para o *Storage*, que por sua vez retorna a URL de acesso à imagem. Esta URL é concatenada a um *json* junto às outras URLs. Após enviar todas as imagens para o *Storage* e ter o *json* com todas as URLs das imagens enviadas, é adicionado ao *json* o *range* de datas nas *tags* `from` e `to`, para definir qual período aquelas imagens representam. Após isso, o *json* é enviado para o *Realtime Database* junto ao nodo que representa o dia atual, tendo como nodo pai `camera`. A Figura 30 ilustra as URLs das imagens salvas no *Realtime Database*.

Então, para executar este serviço com recorrência foi criada uma automação que é executada de forma automática pelo *Home Assistant* a cada cinco minutos, enviando as imagens para o *Storage* e enviando as URLs para o *Realtime Database*.

4.4.2 Aplicação móvel

Para ser possível visualizar os dados armazenados no banco de dados e enviar notificações para o usuário quando a instância do *Home Assistant* estiver *offline*, foi desenvolvido um aplicativo para dispositivos móveis, através do qual o residente pode visualizar os dados que foram replicados para o banco de dados na nuvem.

O aplicativo foi desenvolvido para *Android* utilizando a linguagem *Kotlin*[45]. O desenvolvimento foi feito através da IDE *Android Studio*[46] que é o ambiente de desenvolvimento oficial para aplicativos *Android*. A linguagem *Kotlin* foi escolhida pois ela é recomendada e utilizada pela própria Google, que tem o objetivo de utilizar e aperfeiçoar cada vez mais esta linguagem (ANDROID, 2021a).

A arquitetura utilizada nesta aplicação foi a *Model View ViewModel* (MVVM), que é separada em três camadas, em que a camada *Model* é responsável pela abstração das fontes de dados, sendo que ela trabalha junto com a camada *ViewModel* para salvar e obter os dados. A camada *View* é responsável pelos elementos visuais e informar à camada *ViewModel* as ações dos usuários. Já a camada *ViewModel* tem a atribuição de expor os dados relevantes para a camada *View*, interligando as camadas *Model* e *View*.

Para ler os dados do banco de dados mencionado na Seção 4.4.1 foi utilizado o *Software Development Kit* (SDK) do *Realtime Database* fornecido pelo *Firebase*. Ele fornece uma biblioteca para ler e escrever no banco de dados. Segundo Firebase (2021) suas características são: sincronismo de dados em que toda vez que um dado é alterado, todos os dispositivos conectados irão receber a atualização, o que faz com que os dados listados no aplicativo sejam atualizados automaticamente, sem a necessidade de recarregar a tela; e armazenamento local dos dados que faz com que os dados estejam disponíveis mesmo quando o dispositivo estiver *offline*, quando

[45] https://kotlinlang.org/
[46] https://developer.android.com/studio

a conexão é restabelecida os dados são sincronizados. Para exibir as imagens e ter a capacidade de ampliá-las, foram utilizadas as bibliotecas *Glide*[47] e *Subsampling Scale Image View*[48], respectivamente.

4.4.2.1 Interface do aplicativo

O aplicativo é composto por oito telas, sendo que a tela principal ilustrada na Figura 32 exibe na parte superior o *status* da instância do *Home Assistant*. No canto superior direito há o menu para acesso às configurações do aplicativo e na parte central da tela cinco botões para acessar os dados de temperatura, umidade, gás/fumaça, detecção de movimento e imagens das detecções de movimento da câmera, respectivamente.

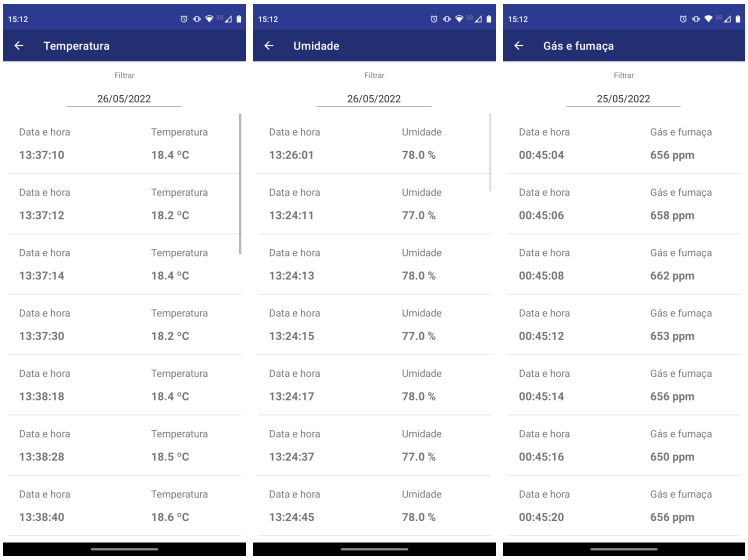
Figura 32 – Tela principal do aplicativo

Fonte: Próprio autor (2022).

As telas dos dados de temperatura, umidade e gás/fumaça seguem o mesmo padrão, listando os dados em ordem decrescente pela hora e filtradas pela data escolhida pelo usuário. A Figura 33 ilustra as três telas.

47 https://github.com/bumptech/glide
48 https://github.com/davemorrissey/subsampling-scale-image-view

Figura 33 – Telas de disposição dos dados de temperatura, umidade e gás/fumaça



(a) Temperatura (b) Umidade (c) Gás e fumaça

Fonte: Próprio autor (2022).

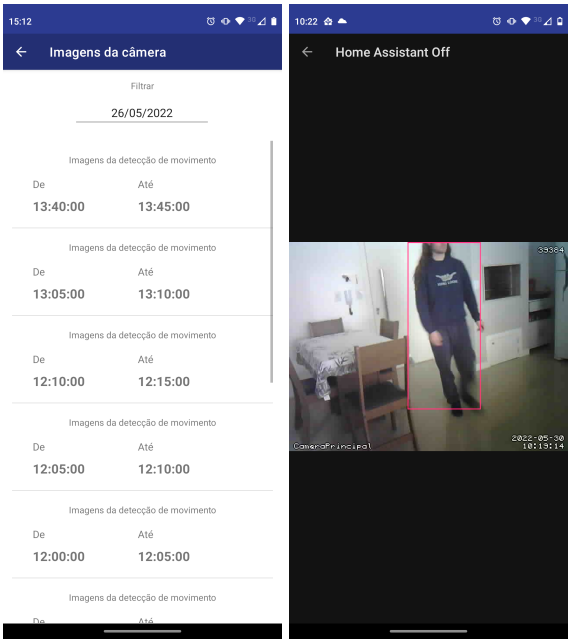
Para a tela de detecção de movimento é listada de forma decrescente a hora em que foi feita a detecção, sendo que os dados são filtráveis por data. A Figura 34 ilustra a tela.

Figura 34 – Tela das detecções de movimento

Fonte: Próprio autor (2022).

Já na tela das imagens de movimento foram disponibilizadas as imagens agrupadas em cinco minutos, sendo que a listagem dos grupos é feita por dia. Para a exibição das imagens foi utilizado um *slider* horizontal que é possível arrastar para o lado para avançar/voltar as imagens. A Figura 35a ilustra a listagem dos períodos e a Figura 35b mostra a visualização das imagens.

Figura 35 – Telas de exibições das imagens das detecções dos movimentos

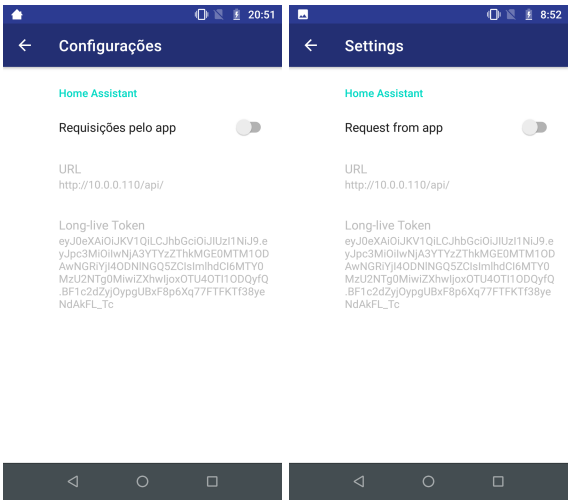


(a) Listagem do agrupamento das imagens (b) *Slider* das imagens de movimento

Fonte: Próprio autor (2022).

O aplicativo foi desenvolvido utilizando o recurso de *strings*[49], com isso, o aplicativo tem a capacidade de alterar seu idioma conforme o idioma definido nas configurações do sistema do *smartphone*. Para este trabalho as *strings* foram escritas em português brasileiro e inglês, todavia é possível adicionar novas traduções ao aplicativo sem precisar alterar seu código-fonte. A Figura 36 mostra a diferença da mesma tela com diferentes idiomas.

Figura 36 – Diferença no idioma do aplicativo conforme configuração



(a) Português (b) Inglês

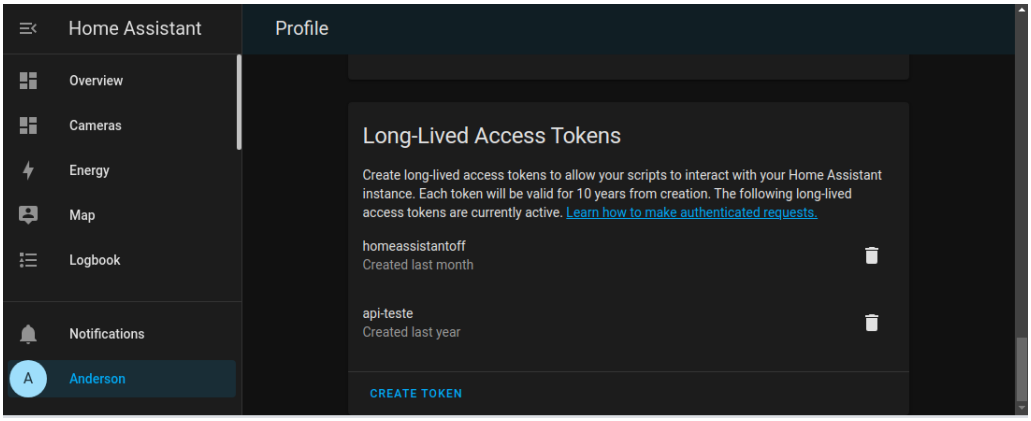
Fonte: Próprio autor (2022).

49 https://developer.android.com/guide/topics/resources/string-resource?hl=pt-br

4.4.2.2 Requisições ao *Home Assistant*

Para verificar se a instância do *Home Assistant* está *online*, são feitas requisições para a API do *Home Assistant* pela URL `http://<url-acesso-homeassistant>/api/`. Esta chamada deve acompanhar o cabeçalho `Authorization: Bearer ABCDEFGH` em que `ABCDEFGH` é uma *long-lived access token* que pode ser gerada por um usuário que tenha acesso ao *Home Assistant* (HOME ASSISTANT, 2022d). A Figura 37 ilustra a criação da *token* pela interface do usuário.

Figura 37 – Criação de uma *Long-lived Token* no *Home Assistant*



Fonte: Próprio autor (2022).

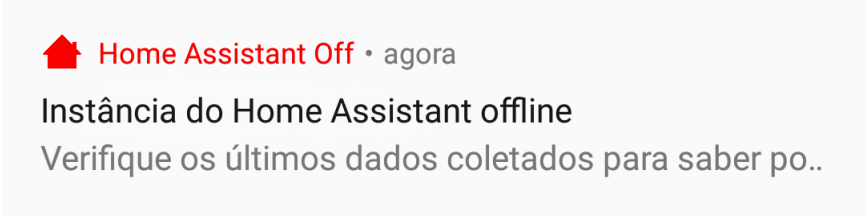
Quando é efetuada a requisição à API e o *Home Assistant* retornar um HTTP *code* 200 significa que a instância está *online*, caso contrário representa que a instância está *offline*. Quando for detectado que a instância está *offline* é gerada uma notificação para informar o residente que a instância está *offline* e que ele não irá receber notificações pelo aplicativo do *Home Assistant*.

Uma forma de validar o *status* da instância do *Home Assistant* é centralizando em um *webserver*. Para desenvolver um *webserver* que valida se a instância do *Home Assistant* está ativa ou não, é preciso que o *webserver* envie as notificações para os *smartphones* que tenham o aplicativo instalado. Dessa forma os *smartphones* precisam assinar um *push notification* para receber as notificações. Para efetuar os *push notifications* é possível desenvolver uma aplicação exclusiva para esta solução, o que gastaria horas de desenvolvimento e posteriormente um valor para a hospedagem do serviço na nuvem. Outra forma seria contratar algum serviço de *push notification* tais como: *Firebase Cloud Messaging*[50] e *Apple Push Notification Service*[51] que tem um custo conforme cada fornecedor. Todavia, em ambos os casos há um gasto mensal com os serviços necessários. Outra limitação seria quanto à exibição do *status* da instância do *Home Assistant*, para o que o *webserver* deveria ter uma API que retorne o *status*. Dessa forma, os *smartphones* teriam que fazer requisições recorrentes a esta API.

[50] https://firebase.google.com/docs/cloud-messaging
[51] https://developer.apple.com/notifications/

Diante disso, a forma que melhor se adéqua para esta solução é o próprio aplicativo fazer as requisições periódicas para validar se a instância está *online* ou não, em que o aplicativo faz as requisições para a API do *Home Assistant* a cada um minuto. Caso não haja uma resposta em 30 segundos, o usuário recebe uma notificação alertando que a instância do *Home Assistant* está inacessível, sendo que essa notificação irá aparecer ininterruptamente até que volte o acesso à instância do *Home Assistant*. A Figura 38 mostra como a notificação aparece para o usuário.

Figura 38 – Notificação *Home Assistant offline*



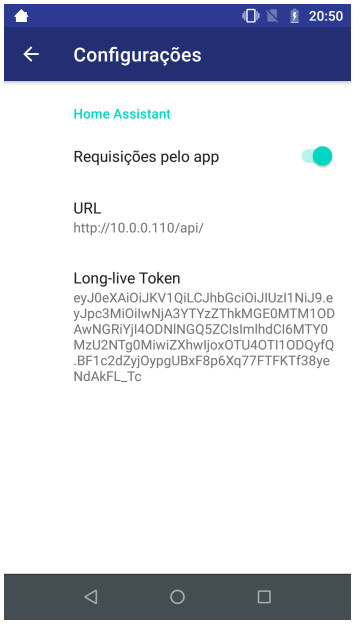
Fonte: Próprio autor (2022).

Para definir períodos de requisições recorrentes ao *Home Assistant* pelo aplicativo foi utilizada a classe `AlarmManager`, sendo que ela possui diversos métodos para definir o disparo dos alarmes com recorrência. As verificações à instância do *Home Assistant* não precisam ser disparadas em horários precisos, por isso foi utilizado o método `setInexactRepeating` que dispara as repetições de forma inexata, com isso ele consegue ter uma maior eficiência energética do que as recorrências tradicionais, que disparam o alarme na hora exata. O sistema consegue ajustar a entrega dos alarmes para serem disparados de forma simultânea, evitando acordar o dispositivo mais do que o necessário (ANDROID, 2022). Além disso, é preciso definir qual tipo de alarme deve ser utilizado, sendo que para este trabalho foi escolhido o tipo `ELAPSED_REALTIME_WAKEUP` que ativa o dispositivo e dispara o alarme após o tempo especificado desde a inicialização do dispositivo. Ele foi utilizado já que o sistema pode estar dormindo e por ser uma notificação importante o *smartphone* deve disparar a notificação independente do seu estado. Para fazer as requisições à API do *Home Assistant* utilizando HTTP foi usada a biblioteca `Volley`[52].

Para o aplicativo ser utilizado por diversos usuários, viu-se necessária a criação de parâmetros que permitam especificar a URL da instância do *Home Assistante*, a *Long-Live Token* e a possibilidade de desabilitar as requisições caso o usuário desejar. Para isso, foram criados três parâmetros, sendo que eles foram desenvolvidos utilizando a biblioteca `Preference`[53] do *Android*. A biblioteca `Preference` tem por padrão a capacidade de armazenar e gravar os dados parametrizados em pares de chave-valor simples em um arquivo armazenado na sessão do aplicativo que apenas o aplicativo tem acesso. Além disso, a biblioteca consegue abstrair a complexidade sendo possível obter os dados parametrizados de forma simples (ANDROID, 2021b). A Figura 39 ilustra a tela de configuração do aplicativo.

[52] https://developer.android.com/training/volley
[53] https://developer.android.com/training/data-storage/shared-preferences?hl=pt-br

Figura 39 – Tela de parâmetros do aplicativo para as requisições ao *Home Assistant*



Fonte: Próprio autor (2022).

4.5 ESTIMATIVA DO CUSTO PARA A APLICAÇÃO DA PROPOSTA

Para que os usuários possam instalar em suas residências a aplicação sugerida neste estudo, há uma estimativas de valores a serem investidos. O custo da aplicação proposta foi dividido em dois, sendo que o primeiro refere-se à aquisição dos componentes do sistema e o segundo, trata-se do custo mensal para manter a aplicação em funcionamento.

A compra dos dispositivos necessários para o sistema totaliza, aproximadamente R$ 1.494,60, sendo que o valor divide-se em: um *Raspberry Pi* 3B que custa R$ 1.299,00[54]; um microcontrolador NodeMCU ESP32 no valor de R$72,90[55]; um microcontrolador ESP-CAM no valor de R$67,00[56]; um sensor de temperatura e umidade DHT11 que custa R$16,90[57]; um sensor de gás e fumaça MQ-2 por R$18,90[58]; e um sensor de movimento PIR no valor de R$19,90[59]. Importante ressaltar que todos os itens aqui citados podem apresentar variação de preço em diferentes *sites* ou lojas.

Em relação ao custo mensal da aplicação, os usuários terão que dispender de seus orçamentos diferentes valores relacionados às funcionalidades necessárias para o sistema operar. Um deles é o serviço de acesso externo oferecido pela empresa Nabu Casa que possui o custo de U$ 6,50 (dólares) ao mês. Já o custo dos banco de dados da nuvem varia conforme a demanda.

54 https://www.amazon.com.br/Placa-Raspberry-Quadcore-1-2ghz-Bluetooth/dp/B01CD5VC92/
55 https://www.filipeflop.com/produto/modulo-wifi-esp32-bluetooth
56 https://shopee.com.br/ESP32-CAM-ESP32-com-camera-microcontrolador-ESP-NodeMCU-arduino-compat%C3%ADvel-i.356273232.8903906216
57 https://www.filipeflop.com/produto/sensor-de-umidade-e-temperatura-dht11
58 https://www.filipeflop.com/produto/sensor-de-gas-mq-2-inflamavel-e-fumaca
59 https://www.filipeflop.com/produto/mini-sensor-de-movimento-presenca-pir

O *Realtime Database* libera de forma gratuita 1 GB, após isso é cobrado U$ 5,00 (dólares) a cada GB. Já o *Storage* possui 5 GB de forma gratuita, após isso é cobrado U$ 0,026 (dólares) para cada GB utilizado. Dessa forma, o valor total estimado é de no mínimo U$ 6,50 (dólares) ou R$ 34,16 (cotação de julho de 2022), caso seja utilizado o plano gratuito do banco de dados na nuvem, podendo aumentar conforme a demanda de armazenamento.

4.6 CONCLUSÕES DO CAPÍTULO

Ao final do desenvolvimento deste capítulo pode ser concluído que o *Home Assistant*, além de ser um *software open source* que permite desenvolver funcionalidades novas por qualquer pessoa, possui uma comunidade ativa por isso possui diversas funcionalidades desenvolvidas tanto pela equipe mantenedora quanto por sua comunidade, sendo vantajoso utilizá-lo ao invés de desenvolver uma aplicação nova.

Ao implementar o nível de QoS 1 para a publicação dos tópicos no MQTT *Broker* através do NodeMCU ESP-32, viu-se a limitação que a maioria das bibliotecas de cliente MQTT não implementam níveis de QoS 1 e 2 para a publicação dos tópicos. Porém, para a subscrição dos tópicos elas implementam os três níveis.

O *MotionEye* tem a capacidade de detectar movimento por *streaming* em tempo real. Todavia, requer um certo poder computacional dependendo da qualidade do vídeo. Para este trabalho quem processa as detecções de movimento pelo *MotionEye* é um *Raspberry Pi Model* 3B, que além disso executa em paralelo o serviço do *Home Assistant*. Dessa forma, houve uma limitação quanto à detecção de movimento quando a resolução da ESP-CAM estava no máximo (1600x1200), apresentando travamento no *streaming* de vídeo quando era detectado movimento. Para atenuar este problema, foi necessária a diminuição da resolução para 640x480 e da qualidade da imagem transmitida pela ESP-CAM.

Para a replicação dos dados dos sensores e as imagens das detecções de movimento para a nuvem foi necessária a criação de uma integração customizada no *Home Assistant* já que não havia uma integração com o *Firebase*.

Ao ser replicado os dados para a nuvem, notou-se uma grande quantidade de dados oriundos do sensor de gás e fumaça, já que ele coleta dados do ambiente a cada dois segundos e possui uma pequena porém, constante variação nos dados coletados. Dessa forma, para evitar o envio de dados irrelevantes para a nuvem, foi criada uma regra para enviar os dados que tenham valor maior que 1000 PPM, já que o sensor apresentou uma variação entre 600 e 700 PPM em um ambiente sem a presença de gás e fumaça.

5 CONCLUSÕES FINAIS

Ao final do desenvolvimento desse estudo, conclui-se que a aplicação proposta mostrou-se viável, pois conforme visto ao longo do trabalho, foi possível desenvolver uma solução com capacidade de comunicar a coleta de dados oriundos de sensores para um servidor local. Por sua vez, o servidor disponibiliza essas informações para análise dos usuários e envia notificações para os residentes a fim de prover segurança domiciliar. Com a coleta contínua do dado referente ao índice de gás e fumaça no ambiente, é possível fazer validações para que quando for detectado um valor acima de 1500 PPM, o residente seja avisado por meio de notificações em seu *smartphone*. Dessa forma, é possível prevenir danos à saúde e aos bens materiais já que o residente será notificado para efetuar as contramedidas cabíveis ao momento.

Os usuários do sistema podem acessar os dados referentes à sua moradia em tempo real, o que facilita a prevenção de incidentes domésticos e pode inibir tentativas de furtos, já que por meio do acompanhamento das informações, o usuário poderá efetuar as contramedidas necessárias, como por exemplo contatar a polícia ou o corpo de bombeiros.

Por meio deste estudo, ficou evidente que a efetividade do sistema de segurança domiciliar necessita de um acesso remoto por meio da *internet* pra que o residente consiga acessar os dados e receber as notificações quando não estiver em sua residência. Entretanto, é importante frisar que a exposição das informações pessoais e da residência à *internet*, necessita de precauções, tais como a utilização de criptografia na comunicação e autenticação dos usuários, pois uma vez que os dados estão na *web*, poderão ser alvo de possíveis ataques cibernéticos.

Pelo fato desta solução ser voltada à segurança domiciliar, notou-se a necessidade de redundância no acesso aos dados, já que o servidor é local e em caso de falta de energia ou *internet* o usuário não receberá as notificações, o que impedirá seu conhecimento sobre um possível incêndio ou furto.

Para detectar a inacessibilidade à instância do *Home Assistant* é necessário um verificador. A utilização do aplicativo móvel como verificador de acesso externo ao *Home Assistant*, pode ocasionar requisições demasiadas na API do *Home Assistant* caso hajam muitos dispositivos fazendo-as. Por outro lado, ao ser utilizada uma aplicação específica como verificadora de acesso externo ao *Home Assistant*, é preciso que ela esteja hospedada em um local fora da residência, na nuvem por exemplo, acarretando em despesas extras. Dessa forma, apesar do aplicativo móvel poder ocasionar requisições demasiadas, na vida real é pouco provável que isso aconteça, já que os residentes normalmente não possuem muitos *smartphones*. Dessa forma, para que o residente saiba se o servidor local está ou não operante, foi desenvolvido um aplicativo para verificar a instância local e notificar caso ela esteja *offline*. Além disso, esse aplicativo disponibiliza os dados coletados que estão replicados no banco de dados em nuvem, para que o

residente possa analisá-los para identificar algum possível problema.

Pelo fato do MQTT desacoplar o enviador do receptor da mensagem, para este trabalho não importa como e nem por quem os dados sejam coletados, pois para o *Home Assistant* o necessário é receber os dados pelos tópicos determinados. Com isso, é possível dispor de cópias idênticas das configurações do *Home Assistant*, porém utilizar sensores com diferentes qualidades, amplitudes e precisões, conforme a necessidade e condição financeira de cada residência. A câmera apesar de não utilizar o protocolo MQTT possui a mesma vantagem, pois disponibiliza o *streaming* de vídeo em um endereço IP. Com isso, apesar deste trabalho ter usado uma ESP-CAM, é possível utilizar qualquer câmera IP.

Além disso, foi possível desenvolver esta solução utilizando *softwares* e *hardwares open source*. Com isso, foi possível desenvolver um sistema livre para ser integrado com diversos tipos de dispositivos, sem limitações impostas por grandes corporações, que limitam o consumidor a comprar dispositivos da sua marca.

Como alternativas de trabalhos futuros sobre possibilidades que não foram exploradas neste estudo estão relacionados os seguintes pontos:

1. Tornar a aplicação modular para que outros sensores possam ser acoplados à aplicação de forma facilitada;
2. Criação de automações para interrupção no fornecimento de gás caso seja detectado vazamento de gás;
3. Alarme sonoro para detecção de intrusos;
4. Canal de comunicação direta com as autoridades (polícia, bombeiros) caso ocorra algum incidente;
5. Para aumentar a robustez da aplicação podem ser criadas redundâncias no *link* de *internet* e redundâncias no fornecimento de energia dos microcontroladores e do *Raspberry Pi*;
6. Melhorar a segurança da aplicação, utilizando SSL para a transmissão dos dados via MQTT e adicionar autenticação ao *Firebase* para restringir o acesso aos dados apenas aos residentes do domicílio.

REFERÊNCIAS

ANDROID. Abordagem kotlin do android. 2021. Disponível em: <https://developer.android.com/kotlin/first?hl=pt-br>. Acesso em: 01 maio 2022.

____. Configurações: Usar valores salvos. 2021. Disponível em: <https://developer.android.com/guide/topics/ui/settings/use-saved-values>. Acesso em: 01 maio 2022.

____. Alarmmanager. 2022. Disponível em: <https://developer.android.com/reference/android/app/AlarmManager?hl=pt-br>. Acesso em: 01 maio 2022.

ASHTON, K. That 'internet of things' thing. 22 Jun 2009. Disponível em: <https://www.rfidjournal.com/that-internet-of-things-thing>. Acesso em: 18 ago. 2021.

ASSOCIAÇÃO BRASILEIRA DAS EMPRESAS DE SISTEMAS ELETRÔNICOS DE SEGURANÇA. **Segurança inteligente**. São Paulo, SP, 2021. Disponível em: <https://abese.org.br/revista-seguranca-inteligente-18/>. Acesso em: 07 ago. 2021.

BOLZANI, C. A. M. Desmistificando a domótica. Jun 2007. Disponível em: <https://caiobolzani.com/artigos/art01_07.pdf>. Acesso em: 19 ago. 2021.

BRUXEL, I. L. **Sistema de Segurança com Detecção e Acompanhamento de Movimento Automatizado**. 48 f. Trabalho de Conclusão do Curso (Engenharia da Computação) — Centro de Ciências Exatas e Tecnológicas, Centro Universitário Univates, Lajeado, 2015.

CAMERON, N. **Electronics Projects with the ESP8266 and ESP32**: Building web pages, applications, and wifi enabled devices. Edinburgo, UK: Apress Media LLC, 2021. 26-40 p.

CDC. L.p.g: Immediately dangerous to life or health concentrations (idlh). 1994. Disponível em: <https://www.cdc.gov/niosh/idlh/68476857.html>. Acesso em: 26 set. 2021.

CECCON, M. Escapar de intoxicação com monóxido de carbono exige decisão rápida. **Gazeta do povo**, 23 Maio 2019. Disponível em: <https://www.gazetadopovo.com.br/viver-bem/saude-e-bem-estar/intoxicacao-com-monoxido-de-carbono/>. Acesso em: 08 ago. 2021.

CLOUDFLARE. **What does buffering mean?** 2021. Disponível em: <https://www.cloudflare.com/pt-br/learning/video/what-is-buffering/>. Acesso em: 5 set. 2021.

____. **What is live streaming?** 2021. Disponível em: <https://www.cloudflare.com/pt-br/learning/video/what-is-live-streaming/>. Acesso em: 5 set. 2021.

____. **What is streaming?** 2021. Disponível em: <https://www.cloudflare.com/pt-br/learning/video/what-is-streaming/>. Acesso em: 5 set. 2021.

COBB, J. **Prepper's Home Defense**: Security strategies to protect your family by any means necessary. [S.l.]: UlySSeS PreSS, 2012. 13-27 p.

CONWAY, D. G. **The Home Security Handbook**. [S.l.]: How To Content, 2005. 6-15 p.

____. **The A-Z of Home Security**: How to keep your home and family safe from crime. 2. ed. [S.l.]: How To Content, 2009. 1-8 p.

CRUL. command line tool and library. 2022. Disponível em: <https://curl.se/>. Acesso em: 18 abr. 2022.

DAWOUD, D. S.; DAWOUD, P. **Microcontroller and Smart Home Networks**. Gistrup, Dinamarca: River Publishers, 2020. 513-538 p.

DUARTE, O. C. M. B. Ip security. 2003. Disponível em: <https://www.gta.ufrj.br/grad/03_1/ip-security/>. Acesso em: 21 ago. 2021.

ESP8266 ARDUINO CORE. My esp crashes running some code. how to troubleshoot it? 2022. Disponível em: <https://arduino-esp8266.readthedocs.io/en/latest/faq/a02-my-esp-crashes.html#other-causes-for-crashes>. Acesso em: 20 fev. 2022.

ESPRESSIF SYSTEMS. **ESP32 WROOM 32**. 2021. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf>. Acesso em: 26 ago. 2021.

EVOLVIT. Cloud vs local servers: Weighing up the pros and cons. 2019. Disponível em: <https://evolvit.co.uk/it-support/server-support/cloud-vs-local-servers-weighing-up-the-pros-and-cons/>. Acesso em: 07 set. 2021.

FILIPEFLOP. Sensor de presença e movimento microondas rcwl-0516. 2021. Disponível em: <https://www.filipeflop.com/produto/sensor-de-presenca-e-movimento-microondas-rcwl-0516/>. Acesso em: 02 out. 2021.

____. Sensor de umidade e temperatura am2302 dht22. 2021. Disponível em: <https://www.filipeflop.com/produto/sensor-de-umidade-e-temperatura-am2302-dht22/>. Acesso em: 26 set. 2021.

____. Sensor de umidade e temperatura dht11. 2021. Disponível em: <https://www.filipeflop.com/produto/sensor-de-umidade-e-temperatura-dht11/>. Acesso em: 26 set. 2021.

____. Sensores de movimento, o que são? 2021. Disponível em: <https://www.filipeflop.com/categoria/sensores/movimento-e-proximidade/>. Acesso em: 02 out. 2021.

FIREBASE. Firebase realtime database. 2021. Disponível em: <https://firebase.google.com/docs/database>. Acesso em: 12 fev. 2022.

FOROUZAN, B. A. **Comunicação de Dados e Redes de Computadores**. Nova York, EUA: AMHG Editora Ltda, 2010. 990 p.

FOWLER, M. **UML Distilled**: A brief guide to the standard object modeling language. 3. ed. [S.l.]: Addison-Wesley Professional, 2003. 102-117 p.

GIMP. Despeckle. 2022. Disponível em: <https://docs.gimp.org/2.10/en/plug-in-despeckle.html>. Acesso em: 15 abr. 2022.

GIT. About. 2022. Disponível em: <https://git-scm.com/about>. Acesso em: 21 abr. 2022.

HILLAR, G. C. **MQTT Essentials**: A lightweight iot protocol. Birmingham, UK: Packt Publishing, 2021. 27-75 p.

HIVEMQ. **MQTT & MQTT 5 Essentials**: A comprehensive overview of mqtt facts and features for beginners and experts alike. Bavaria, Germany, 2021. 5-39 p.

HOME ASSISTANT. Architecture. 2021. Disponível em: <https://developers.home-assistant.io/docs/architecture_index/>. Acesso em: 23 out. 2021.

____. Authentication. 2021. Disponível em: <https://developers.home-assistant.io/docs/auth_index>. Acesso em: 26 out. 2021.

____. Automating home assistant. 2021. Disponível em: <https://www.home-assistant.io/docs/automation/>. Acesso em: 24 out. 2021.

____. Configuration.yaml. 2021. Disponível em: <https://www.home-assistant.io/docs/configuration/>. Acesso em: 24 out. 2021.

____. Core architecture. 2021. Disponível em: <https://developers.home-assistant.io/docs/architecture/core/>. Acesso em: 23 out. 2021.

____. Home assistant add-ons. 2021. Disponível em: <https://www.home-assistant.io/addons/>. Acesso em: 24 out. 2021.

____. Home assistant features. 2021. Disponível em: <https://www.home-assistant.io/>. Acesso em: 24 out. 2021.

____. Home assistant operating system. 2021. Disponível em: <https://developers.home-assistant.io/docs/operating-system>. Acesso em: 23 out. 2021.

____. Installation. 2021. Disponível em: <https://www.home-assistant.io/installation/>. Acesso em: 25 out. 2021.

____. Integration architecture. 2021. Disponível em: <https://developers.home-assistant.io/docs/architecture_components/>. Acesso em: 24 out. 2021.

____. Lovelace. 2021. Disponível em: <https://www.home-assistant.io/lovelace/>. Acesso em: 27 out. 2021.

____. Notification: Introduction. 2021. Disponível em: <https://companion.home-assistant.io/docs/notifications/notifications-basic/>. Acesso em: 27 out. 2021.

____. Scripts. 2021. Disponível em: <https://www.home-assistant.io/integrations/script/>. Acesso em: 24 out. 2021.

____. Yaml. 2021. Disponível em: <https://www.home-assistant.io/docs/configuration/yaml/>. Acesso em: 24 out. 2021.

____. Binary sensor. 2022. Disponível em: <https://www.home-assistant.io/integrations/binary_sensor/>. Acesso em: 19 mar. 2022.

____. Integration manifest. 2022. Disponível em: <https://developers.home-assistant.io/docs/creating_integration_manifest/>. Acesso em: 23 maio 2022.

____. Recorder. 2022. Disponível em: <https://www.home-assistant.io/integrations/recorder>. Acesso em: 08 maio 2022.

____. Rest api. 2022. Disponível em: <https://developers.home-assistant.io/docs/api/rest/>. Acesso em: 13 fev. 2022.

INSTITUTO DIGITAL. Módulo sensor mq-2 de gás inflamável e fumaça. 2021. Disponível em: <https://www.institutodigital.com.br/produto/modulo-sensor-de-gas-inflamavel-e-fumaca-mq-2/>. Acesso em: 02 out. 2021.

INTELBRAS. Saiba como funcionam os tipos de sensores de alarme. 2018. Disponível em: <https://blog.intelbras.com.br/saiba-como-funcionam-os-tipos-de-sensores-de-alarme/>. Acesso em: 02 out. 2021.

IPSENSE. On premise e cloud servers: entenda as principais diferenças. 2017. Disponível em: <https://www.ipsense.com.br/blog/on-premise-e-cloud-servers-entenda-as-principais-diferencas/>. Acesso em: 08 set. 2021.

KNOLLEARY. Arduino client for mqtt. 2022. Disponível em: <https://github.com/knolleary/pubsubclient/tree/v2.8>. Acesso em: 27 mar. 2022.

MAGRANI, E. **A Internet das Coisas**. [S.l.]: FGV Editora, 2018. 24 p.

MAHALLE, P. N. *et al.* **The Convergence of Internet of Things and Cloud for Smart Computing**. [S.l.]: CRC Press, 2021. 43-46 p.

MARCHESAN, M. **Sistema de Monitoramento Residencial Utilizando a Plataforma Arduino**. 62 f. Trabalho de Conclusão do Curso (Tecnologia em Redes de Computadores) — Colégio Técnico Industrial de Santa Maria, Universidade Federal de Santa Maria, Santa Maria, 2012.

MARCIN. What is home assistant and what it can do? 2021. Disponível em: <https://futurehousestore.co.uk/what-is-home-assistant-and-what-it-can-do>. Acesso em: 24 out. 2021.

MICROCONTROLLERSLAB. Esp32 external interrupts using arduino ide. 2022. Disponível em: <https://microcontrollerslab.com/esp32-external-interrupts-tutorial-arduino-ide/>. Acesso em: 26 mar. 2022.

MICROSOFT. O que é computação em nuvem? 2021. Disponível em: <https://azure.microsoft.com/pt-br/overview/what-is-cloud-computing/>. Acesso em: 06 out. 2021.

MIR, T. What is a webhook and how to use it? 2021. Disponível em: <https://www.geeksforgeeks.org/what-is-a-webhook-and-how-to-use-it/>. Acesso em: 13 jun. 2022.

N-ABLE. Dynamic dns overview. 2019. Disponível em: <https://www.n-able.com/blog/dynamic-dns-overview>. Acesso em: 11 nov. 2021.

NABU CASA. Remote ui. 2021. Disponível em: <https://www.nabucasa.com/config/remote/>. Acesso em: 27 out. 2021.

NGROK. Documentation. 2021. Disponível em: <https://ngrok.com/docs>. Acesso em: 27 out. 2021.

NIJHOF, F. Home assistant community add-on: motioneye. 2018. Disponível em: <https://community.home-assistant.io/t/home-assistant-community-add-on-motioneye/71826>. Acesso em: 18 abr. 2022.

NODEMCU. **ESP32 WROOM 32**. 2015. Disponível em: <https://github.com/nodemcu/nodemcu-devkit-v1.0/blob/master/Documents/NODEMCU_DEVKIT_V1.0_PINMAP.png>. Acesso em: 28 ago. 2021.

ORACLE. O que é nosql? 2022. Disponível em: <https://www.oracle.com/br/database/nosql/what-is-nosql/>. Acesso em: 13 fev. 2022.

OVENS, S. Why i use home assistant for open source home automation: Home automation can be a slippery slope. the right open source tools can get you on firmer footing. 2020. Disponível em: <https://opensource.com/article/20/11/home-assistant>. Acesso em: 25 out. 2021.

PINHEIRO, P. Perigos da inalação de fumaça em incêndios. **MD Saúde**, 2021. Disponível em: <https://www.mdsaude.com/pneumologia/fumaca-incendio/>. Acesso em: 08 ago. 2021.

PORT FORWARD. How to forward a port. 2017. Disponível em: <https://portforward.com/>. Acesso em: 17 maio 2022.

POSITIVO. Kit casa segura. 2021. Disponível em: <https://www.positivocasainteligente.com.br/kit-casa-segura/>. Acesso em: 02 out. 2021.

RASPBERRY PI. Getting started. 2021. Disponível em: <https://www.raspberrypi.org/documentation/computers/getting-started.html>. Acesso em: 29 ago. 2021.

RED HAT. O que é middleware e para que ele serve? 2021. Disponível em: <https://www.redhat.com/pt-br/topics/middleware/what-is-middleware>. Acesso em: 09 set. 2021.

RESCORLA, E.; DIERKS, T. **The Transport Layer Security (TLS) Protocol Version 1.2**. RFC Editor, 2008. RFC 5246. (Request for Comments, 5246). Disponível em: <https://rfc-editor.org/rfc/rfc5246.txt>.

ROBU.IN. Mq series gas sensor. 2020. Disponível em: <https://robu.in/mq-series-gas-sensor/>. Acesso em: 26 set. 2021.

____. Mq-2 smoke lpg butane hydrogen gas sensor detector module. 2021. Disponível em: <https://robu.in/product/mq-2-mq2-smoke-gas-lpg-butane-hydrogen-gas-sensor-detector-module/>. Acesso em: 26 set. 2021.

SALADAS, J. What is curl and how does it relate to apis? 2019. Disponível em: <https://developer.ibm.com/articles/what-is-curl-command/>. Acesso em: 18 abr. 2022.

SANTOS, J. S. **Detector de Vazamentos de Gás com Aviso por SMS**. 93 f. Trabalho de Conclusão do Curso (Engenharia da Computação) — Faculdade de Tecnologia e Ciências Sociais Aplicadas, Centro Universitário de Brasília, Brasília, 2012.

SANTOS, P. M. P. **Internet das coisas**: O desafio da privacidade. 8 f. Dissertação (Mestrado em Sistemas de Informação Organizacionais) — Escola Superior de Ciências Empresariais, Instituto Politécnico de Setúbal, Setúbal, 2016.

SANTOS, R.; SANTOS, S. Esp32-cam ai-thinker pinout guide: Gpios usage explained. 2021. Disponível em: <https://randomnerdtutorials.com/esp32-cam-ai-thinker-pinout/>. Acesso em: 28 ago. 2021.

____. Change esp32-cam ov2640 camera settings: Brightness, resolution, quality, contrast, and more. 2022. Disponível em: <https://randomnerdtutorials.com/esp32-cam-ov2640-camera-settings/>. Acesso em: 21 abr. 2022.

SINOPOLI, J. **Advanced Technology for Smart Buildings**. [S.l.]: ARTECH HOUSE, 2016. 1-8 p.

STALLINGS, W. **Arquitetura e Organização de Computadores**. 10. ed. São Paulo, SP: Person Education do Brasil, 2017. 25-27 p.

STEVAN, S. L.; FARINELLI, F. A. **DOMÓTICA - Automação Residencial e Casas Inteligentes com Arduíno e ESP8266**. [S.l.]: Saraiva Educação S.A., 2018. 2-4 p.

SWANSON, D. C. **Signal Processing for Intelligent Sensor Systems**. Nova York, EUA: Marcel Dekker Inc., 2000. 3 p.

UBUNTU. Install and configure samba. 2021. Disponível em: <https://ubuntu.com/tutorials/install-and-configure-samba#1-overview>. Acesso em: 27 dez. 2021.

WILLIAMS, A. Home assistant lets you automate your smart home without giving up privacy: A diy smart home platform to consider. 2018. Disponível em: <https://www.the-ambient.com/features/home-assistant-automation-privacy-582>. Acesso em: 25 out. 2021.

YAML. Yaml 1.2. 2021. Disponível em: <https://yaml.org/>. Acesso em: 24 out. 2021.

YUAN, M. Conhecendo o mqtt. 04 Out 2017. Disponível em: <https://developer.ibm.com/br/articles/iot-mqtt-why-good-for-iot/>. Acesso em: 20 ago. 2021.

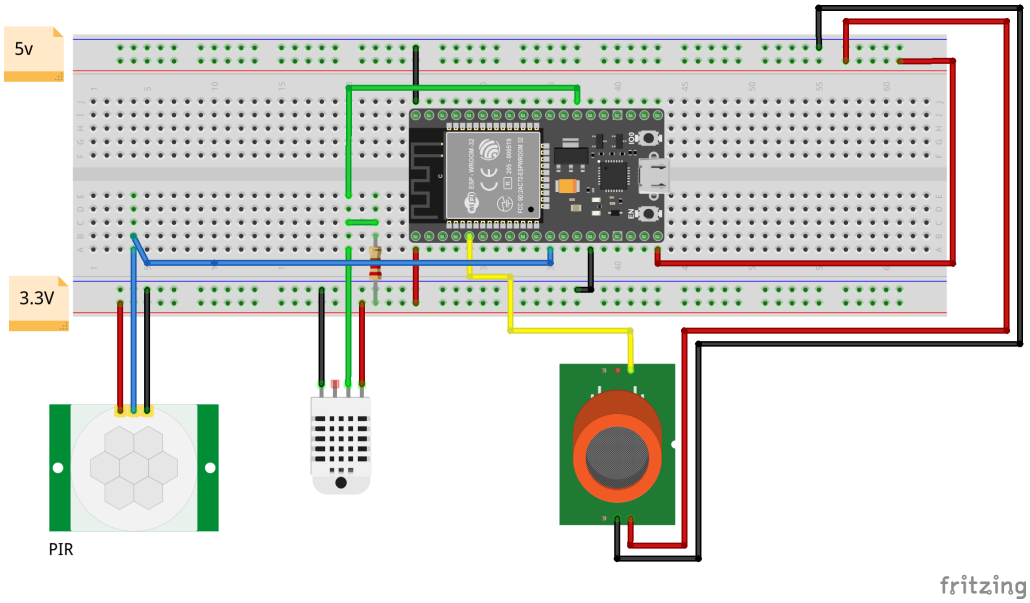
____. Getting to know nodemcu and its devkit board. 7 ago 2017. Disponível em: <https://developer.ibm.com/tutorials/iot-nodemcu-open-why-use/>. Acesso em: 26 ago. 2021.

APÊNDICE A – CIRCUITOS

A Figura 40 ilustra o protótipo virtual do esquema elétrico utilizado para conectar os sensores PIR, DHT11 e MQ-2 ao microcontrolador NodeMCU ESP32. Já a Figura 41 mostra o esquema elétrico do sistema e a Figura 42 ilustra o protótipo físico desenvolvido neste trabalho.

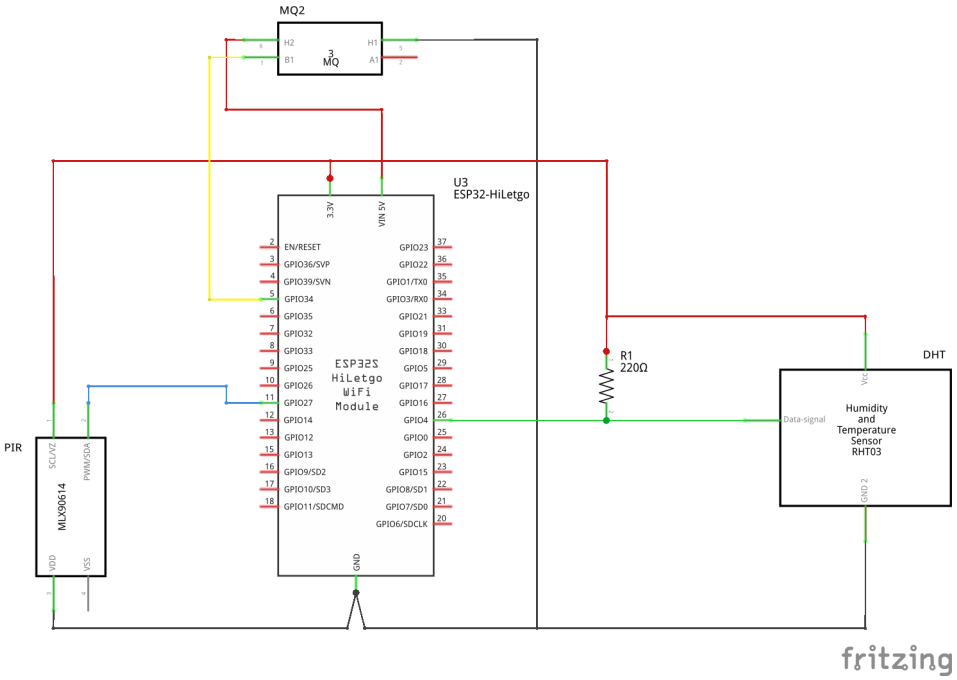
O sensor PIR foi conectado à carga de 3.3 volts e a porta de dados foi conectada à porta 27 do NodeMCU. O sensor DHT11 também foi conectado a 3.3 volts e a porta de dados precisou ser ligada a um resistor de 1K Ω e à porta 4 do NodeMCU. O sensor MQ-2 precisou ser conectado em 5 volts para funcionar corretamente e foi conectado à porta 34 do NodeMCU para a transferência dos dados.

Figura 40 – Protótipo virtual do esquema elétrico



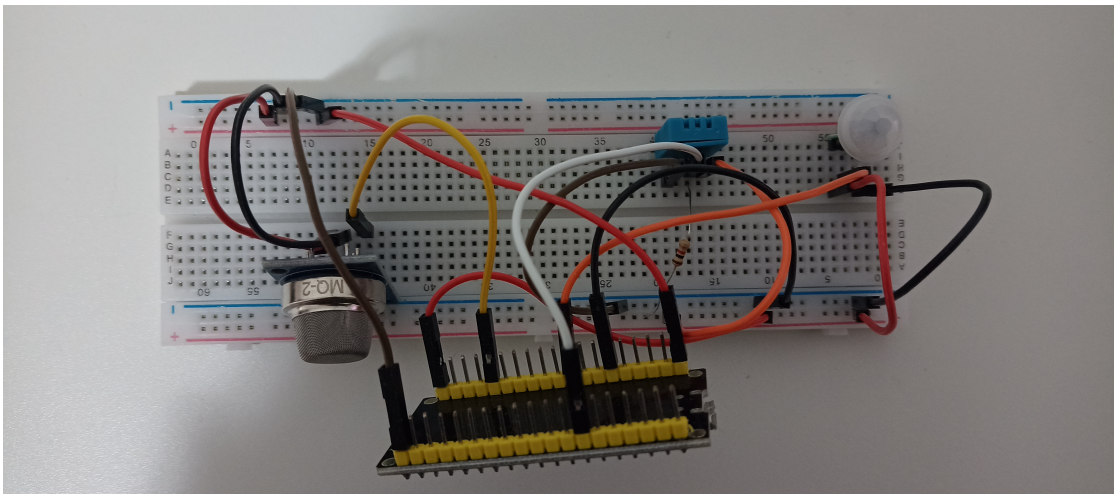
Fonte: Próprio autor (2022).

Figura 41 – Esquema elétrico dos sensores



Fonte: Próprio autor (2022).

Figura 42 – Protótipo físico do esquema elétrico



Fonte: Próprio autor (2022).